OASIS

Service Component Architecture (SCA) Tutorial : Part 1

Anish Karmarkar – Oracle
Ashok Malhotra – Oracle
David Booz - IBM

SCA

- A vendor-, technology-, language-neutral model for the creation of business systems using SOA by the composition and deployment of new and existing service components

Part 1 Outline

- SCA: Introduction
- SCA Assembly
- SCA Java Common Annotations
- SCA Java Common APIs
- SCA Spring Component Implementation
- SCA Java Component Implementation
- SCA BPEL Component Implementation
- SCA Web Service Binding
- SCA JMS Binding
- SCA EJB Session Bean Binding
- Summary
- Demo

Part 2 Outline (Afternoon Session)

- SCA Policy Framework
  - Intents
  - Policy sets
  - Java Annotations for policy framework
  - Mapping intents to policy sets
  - Intents for
    - Reliable messaging
    - Security
  - Demo

Introduction: Outline

- SCA: Why?
- SCA: What?
- SCA: How and Where?

Introduction: Outline

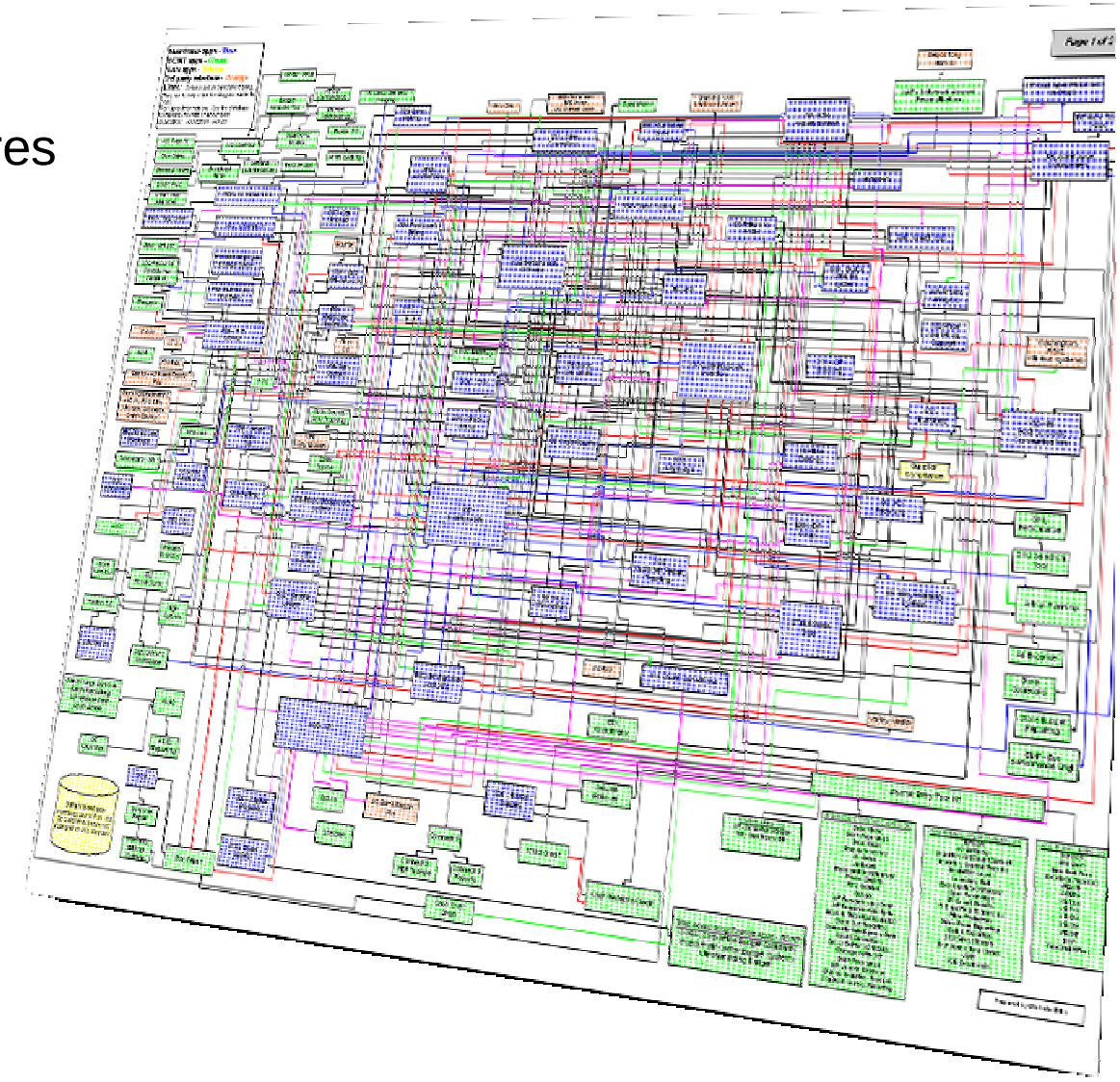
- SCA: Why?
  - Business drivers
  - What we have today
  - Where do we want to go
  - Service-oriented architecture (SOA)
  - SOA core elements
- SCA: What?
- SCA: How and Where?

Business Drivers

- Flexible businesses require flexible IT
  - Globalization demands greater flexibility
  - Global supply chain integration
  - Business processes:
    daily changes vs. yearly changes
  - Growth through flexibility is at
    the top of the CEO agenda
  - Reusable assets can cut costs
    by up to 20%
  - Crucial for flexibility and becoming
    an On Demand Business

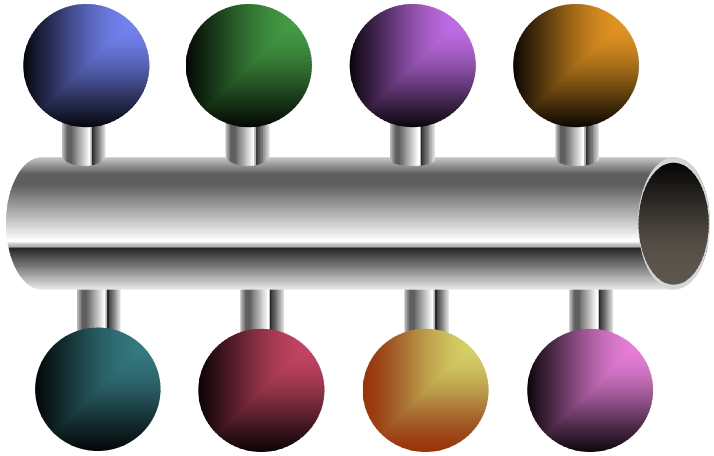
What We have Today

- Complexity
- Rigid, brittle architectures
- Inability to evolve



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

Service-oriented Architechture (SOA)

- SOA derives its technical strategy and vision 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"

SOA Core Elements

- **Service Assembly**
  - technology- and language- independent representation of the composition of services into business solutions
- **Service Component**
  - technology- and language-independent representation of a service which can be composed with other services

Introduction: Outline

- SCA:Why?
- **SCA: What?**
  - **SCA: Simplified Programming Model for SOA**
  - **SCA: What It Is Not**
  - **SCA: High Level View**
  - **SCA Elements**
  - **Assembly Model Concepts**
  - **SCA Interaction Model**
  - **SCA Client and Implementations**
  - **SCA Bindings**
  - **SCA Policy Framework**
    - Policy, Profiles and QoS
    - Attaching Policies and Mapping to Policy Sets
    - Interaction and Implementation Policies
- How and Where?

OASIS

SCA: Simplified Programming Model for SOA

- What is SCA:
  - 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. EJBs, Java POJOs, BPEL process, COBOL, C++, PHP …

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

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

SCA: What is it NOT

- Does not model individual ***workflows***
  - use BPEL or other workflow languages
- Is not ***Web services***
  - SCA can use / may use Web services, but can also build solutions with no Web services content
- Is not tied to a specific runtime environment
  - distributed, hetergeneous, large, small
- Does not force use of specific programming languages and technologies
  - aims to encompass many languages, technologies

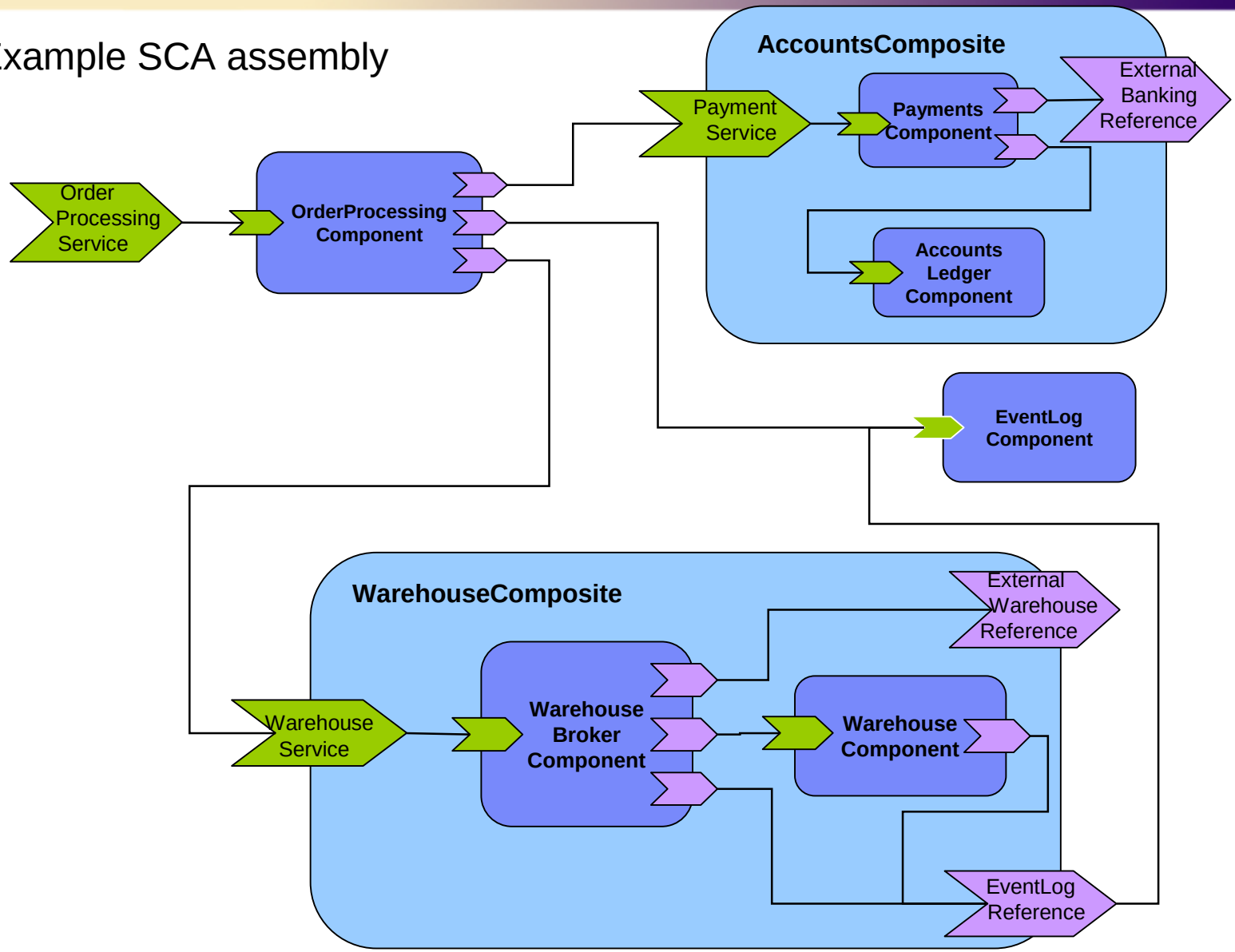
SCA – High Level View

- ***Unified declarative model*** describing service assemblies
  - dependency resolution and configuration
  - declarative policies for infrastructure services
    - Security, Transactions, Reliable messaging
- ***Business-level model*** for implementing services
  - service components with service interfaces
  - no technical APIs like JDBC™, JCA™, JMS™, …
- ***Binding model*** for multiple access methods and infrastructure services
  - WSDL, SOAP over HTTP, JMS™/messaging, Java™ RMI/IIOP…

SCA Elements

- ***Assembly*** Model
  - how to define structure of composite applications
- ***Component Implementation*** specifications
  - how to write business services in particular languages
  - Java, Spring, C++, BPEL, PHP….
- ***Binding*** specifications
  - how to use access methods
  - Web services, JMS, RMI-IIOP, REST…
- ***Policy Framework***
  - how to add infrastructure services to solutions
  - Security, Transactions, Reliable messaging…

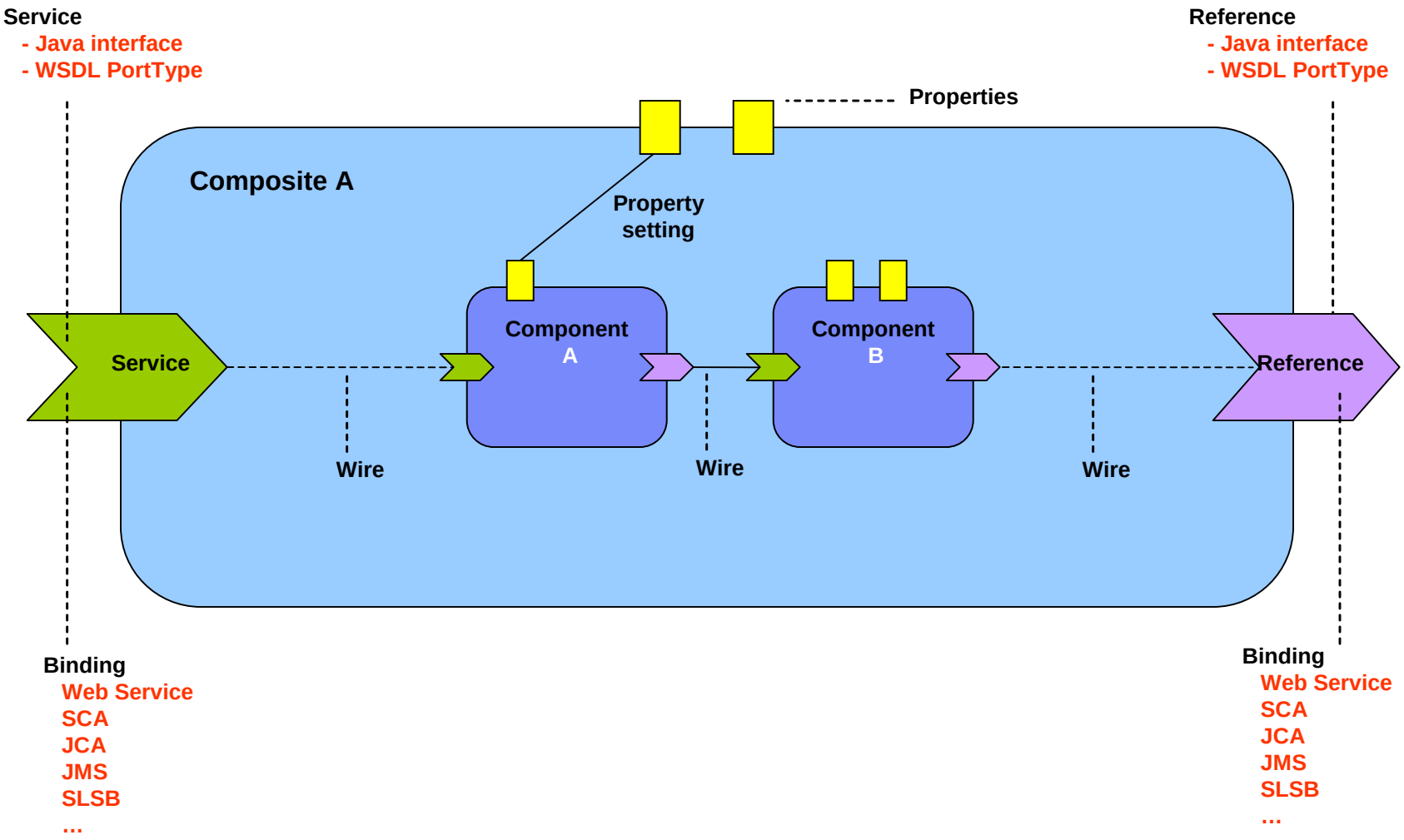
Example SCA assembly



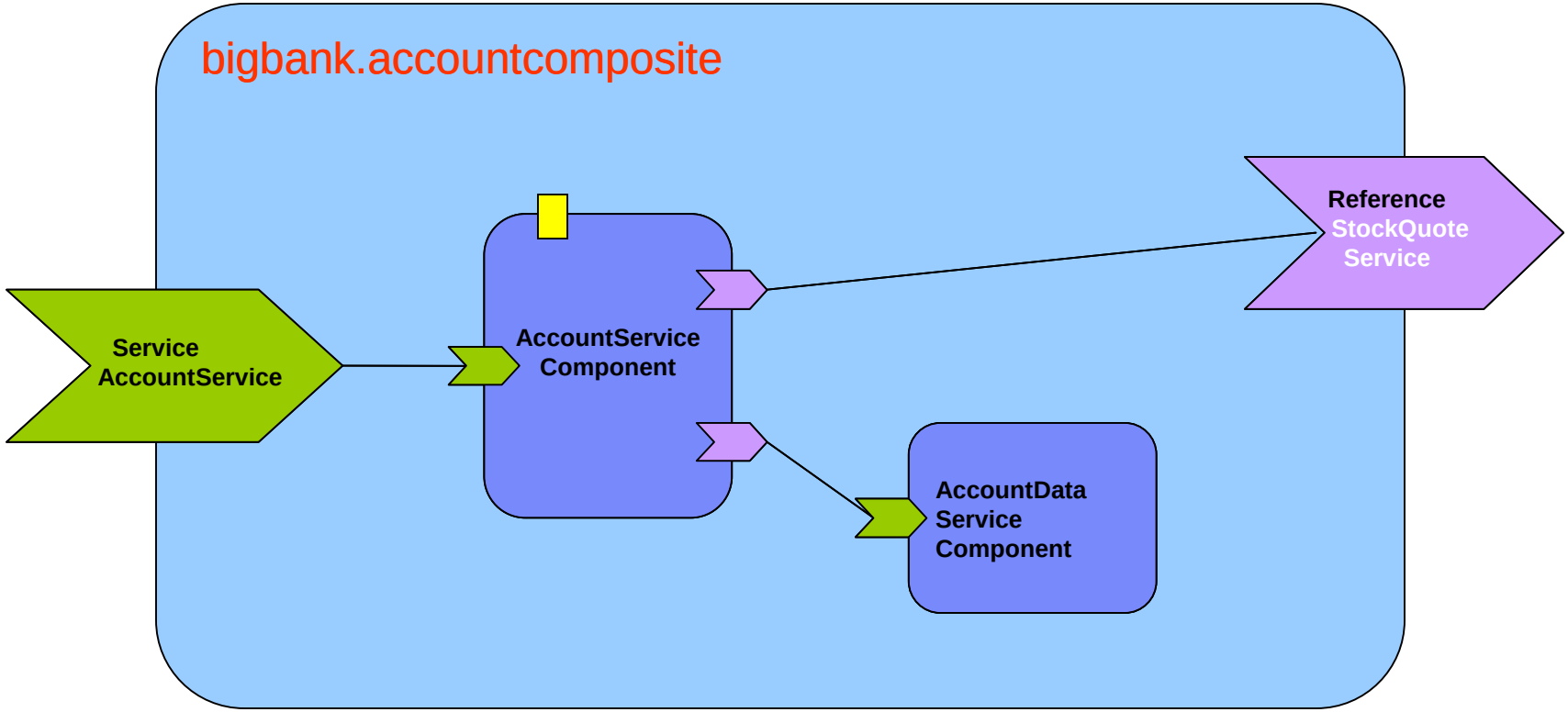
Assembly Model Concepts

- Component
- Implementation
- Composite
- Service
- Reference
- Wire
- ComponentType
- ConstrainingType
- Domain
- Contribution

Composite


Service
- Java interface
- WSDL PortType
Reference
- Java interface
- WSDL PortType
Properties
Composite A
Property setting
Component A
Component B
Service
Reference
Wire
Wire
Wire
Binding
Web Service
SCA
JCA
JMS
SLSB
...
Binding
Web Service
SCA
JCA
JMS
SLSB
...

Example


bigbank.accountcomposite
Service
AccountService
AccountService
Component
Reference
StockQuote
Service
AccountData
Service
Component

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
          name="bigbank.accountcomposite" >

  <service name="AccountService">
     <interface.java interface="services.account.AccountService"/>
     <binding.ws port="http://www.bigbank.com/AccountService#
        wsdl.endpoint(AccountService/AccountServiceSOAP)"/>
     <reference>AccountServiceComponent</reference>
  </service>

  <component name="AccountServiceComponent">
     <implementation.java class="services.account.AccountServiceImpl"/>
     <property name="currency">EURO</property>
     <reference name="accountDataService" target="AccountDataServiceComponent"/>
     <reference name="stockQuoteService" target="StockQuoteService"/>
  </component>

  <component name="AccountDataServiceComponent">
     <implementation.java class="services.accountdata.AccountDataServiceImpl"/>
  </component>

  <reference name="StockQuoteService">
     <interface.java interface="services.stockquote.StockQuoteService"/>
     <binding.ws port="http://www.quickstockquote.com/StockQuoteService#
        wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
  </reference>
</composite>
```

SCA Interaction Model

- ***Synchronous*** & ***Asynchronous*** service relationships
- ***Conversational*** services
  - stateful service interactions
- Asynchronous support
  - “non-blocking” invocation
  - asynchronous client to synchronous service
  - callbacks

SCA Client and Implementation Specifications

- Specifies how service components and service clients are built
- Specific to a particular language or framework or language- or framework-specific APIs
- Extensible
- Currently defined C&I specifications:
  - BPEL
  - Java
  - Spring Framework
  - C++

SCA Bindings

- Specific to particular:
  - Access Method / Protocol / Transport
  - Serialization
  - Framework
- Apply to services and references
- Typically added during deployment
- Currently defined bindings:
  - Web service binding
  - JMS binding
  - EJB Session Bean binding

Policies Framework and Infrastructure Capabilities

- ***Infrastructure*** has many configurable capabilities
  - Security: Authentication and Authorization
  - Security: Privacy, Encryption, Non-Repudiation
  - Transactions, Reliable messaging, etc.
  - Complex sets of configurations across multiple domains of concern

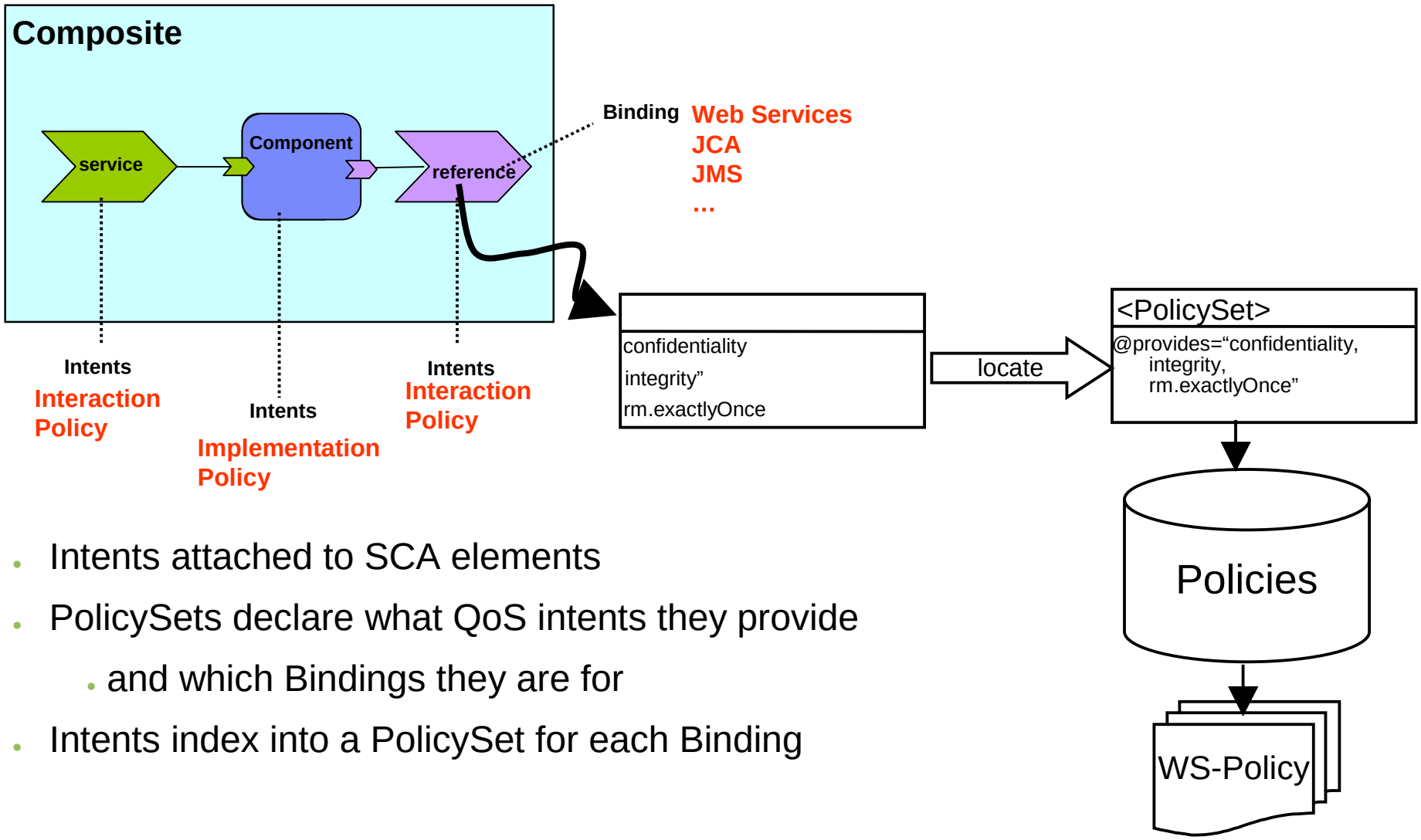
- SCA abstracts out complexity with a ***declarative model***
  - no implementation code impact
  - simplify usage via declarative policy intents
  - simple to apply, modify
  - complex details held in PolicySets

OASIS

Policies, Profiles and Quality of Service

- Framework consists of:
  - SCA policy ***intent***
    - Each represent a single abstract QoS intent
    - may be ***qualified***
  - SCA ***policy sets***
    - Represent a collection of concrete policies to realize an abstract QoS intent
  - WS-Policy
    - A syntax for concrete policies in policy sets
    - others possible…

Attaching Profiles and mapping to PolicySets



- Intents attached to SCA elements
- PolicySets declare what QoS intents they provide
  - and which Bindings they are for
- Intents index into a PolicySet for each Binding

Interaction and Implementation Policies

- ***Interaction policies*** affect the contract between a service requestor and a service provider
  - Things that affect the interaction between them, such as message contexts, wire formats, etc.
  - eg. authentication, confidentiality, integrity
  - eg rm.atLeastOnce, rm.ordered
- ***Implementation policies*** affect the contract between a component and its container
  - Things that affect how the container should manage the component environment, such as transaction monitoring, access control, etc.
  - eg tx.transaction

Introduction: Outline

- SCA:Why?
- SCA: What?
- SCA: How and Where?
  - Open SOA (OSOA) Collaboration
  - OSOA & Evolution of Specifications
  - Implementations
  - Useful Pointers

OASIS

The Open SOA (OSOA) Collaboration

- SCA specs being evolved by group of collaborators
  - BEA
  - Cape Clear
  - IBM
  - Interface21
  - IONA
  - Oracle
  - Primeton
  - Progress Software
  - Red Hat
  - RogueWave
  - SAP
  - Siemens
  - Software AG
  - Sun
  - Sybase
  - Tibco

OASIS

OSOA Collaboration (cont.)

- Several working groups have been working for the past year
- Consists of the following Working Groups:
  - Assembly
    - Assembly specification
  - Policy
    - Policy Framework
  - Bindings
    - WS binding
    - JMS binding
  - C++
  - Java
    - Java Common Annotation & API
    - Java Component Implementation
    - Spring Component Implementation
    - EJB Session Bean binding

OSOA Collaboration (cont.)

- OSOA is not a standards body
- Innovate rapidly and deliver the specification set to the community
- Royalty Free
- Public website for specifications, white papers, news, etc
  - http://www.osoa.org
  - comment and feedback welcome
  - OSOA Supporters group

OSOA & Evolution of SCA Specs

- Pre-1.0 version of the specifications
  - 0.9 version available since November 2005
  - Newer point version updated to the osoa.org website
- 1.0 version: Released March 2007
- Planned Standardization
  - Will be done in OASIS
  - Special OCSA member section
    - http://www.oasis-opencsa.org/
  - Technical Committees to follow

OSOA & Evolution of SCA Specs (cont.)

- SCA 1.0 (released March 2007) contains:
  - SCA Assembly Model
  - SCA Policy Framework
  - SCA BPEL Client and Implementation
  - SCA C++ Client and Implementation
  - SCA Java Common Annotations and APIs
  - SCA Java Component Implementation
  - SCA Spring Component Implementation
  - SCA Web Service Binding
  - SCA EJB Session Bean Binding
  - SCA JMS Binding

Implementations

- Open source: Apache and Eclipse
- Vendor: BEA, IBM, Oracle, SAP, Tibco, Iona, RogueWave, …

Useful Pointers

- SCA specifications and related material
  - http://www.osoa.org
- Contact:
  - anish.karmarkar@oracle.com
  - ashok.malhotra@oracle.com
  - booz@us.ibm.com

OASIS

Useful Pointers (cont.)

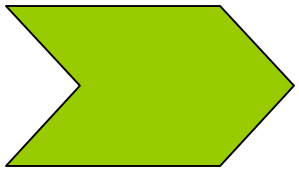
- SCA Assembly Model
  - http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf?version=1
  - Michael Beisiegel, Henning Blohm, Dave Booz, Mike Edwards, Oisin Hurley, Sabin Ielceanu, Alex Miller, Anish Karmarkar, Ashok Malhotra, Jim Marino, Martin Nally, Eric Newcomer, Sanjay Patil Greg Pavlik, Martin Raepple, Michael Rowley,Ken Tam, Scott Vorthmann, Peter Walker, Lance Waterman
- SCA Policy Framework
  - http://www.osoa.org/download/attachments/35/SCA_Policy_Framework_V100.pdf?version=1
  - Michael Beisiegel, Dave Booz, Ching-Yun Chao, Mike Edwards, Sabin Ielceanu, Anish Karmarkar, Ashok Malhotra, Eric Newcomer, Sanjay Patil, Michael Rowley, Chris Sharp, Ümit Yalçinalp
- SCA Java Component Implementation
  - http://www.osoa.org/download/attachments/35/SCA_JavaComponentImplementation_V100.pdf?version=1
  - Ron Barac, Michael Beisiegel Henning Blohm, Dave Booz, Jeremy Boynes, Ching-Yun Chao, Adrian Colyer, Mike Edwards, Hal Hildebrand, Sabin Ielceanu, Anish Karmarkar, Daniel Kulp, Ashok Malhotra, Jim Marino, Michael Rowley, Ken Tam, Scott Vorthmann, Lance Waterman
- SCA Java Common Annotations and APIs
  - http://www.osoa.org/download/attachments/35/SCA_JavaAnnotationsAndAPIs_V100.pdf?version=1
  - Ron Barack, Michael Beisiegel, Henning Blohm, Dave Booz, Jeremy Boynes, Ching-Yun Chao, Adrian Colyer, Mike Edwards, Hal Hildebrand, Sabin Ielceanu, Anish Karmarkar, Daniel Kulp, Ashok Malhotra, Jim Marino, Michael Rowley, Ken Tam, Scott Vorthmann, Lance Waterman
- SCA Spring Component Implementation
  - http://www.osoa.org/download/attachments/35/SCA_SpringComponentImplementationSpecification-V100.pdf?version=1
  - Michael Beisiegel, Dave Booz, Adrian Colyer, Hal Hildebrand, Jim Marino, Ken Tam
- Client and Implementation Model Specification for WS-BPEL
  - http://www.osoa.org/download/attachments/35/SCA_ClientAndImplementationModelforBPEL_V100.pdf?version=1
  - Martin Chapman, Sabin Ielceanu, Dieter Koenig, Michael Rowley, Ivana Trickovic,Alex Yiu
- Cient and Implementation Model for C++
  - http://www.osoa.org/download/attachments/35/SCA_ClientAndImplementationModel_Cpp-V100.pdf?version=2
  - Andrew Borley, David Haney, Oisin Hurley, Todd Little, Brian Lorenz, Conor Patten, Pete Robbins, Colin Thorne
- Web Service Binding
  - http://www.osoa.org/download/attachments/35/SCA_WebServiceBinding_V100.pdf?version=2
  - Simon Holdsworth, Sabin Ielceanu, Anish Karmarkar, Mark Little, Sanjay Patil, Michael Rowley
- EJB Session Bean Binding
  - http://www.osoa.org/download/attachments/35/SCA_EJBSessionBeanBinding_V100.pdf?version=1
  - Ron Barack, Henning Blohm, Dave Booz, Rashmi Hunt, Michael Keith, Michael Rowley
- JMS Binding
  - http://www.osoa.org/download/attachments/35/SCA_JMSBinding_V100.pdf?version=2
  - Rajith Attapattu, Henning Blohm, Simon Holdsworth
  - Eric Johnson, Anish Karmarkar, Michael Rowley

SCA Assembly: Outline

- SCA Assembly: Concepts
- SCA Assembly Symbols
- Bigbank Example
- Recursive Composition
- Local v. Remote Interface
- Bidirectional Interfaces
- Conversational Interfaces
- Autowiring
- Composite Inclusion
- Reuse in Assembly
- Top-Down Design: constrainingType
- Extensibility
- Packaging & Deployment: Domain
- Packaging & Deployment: Contributions
- Summary

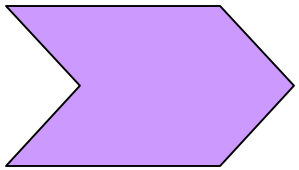
SCA Assembly: Concepts

- Service: externally accessible software functionality of an implementation



Service

- Reference: dependency on an external service



Reference

SCA Assembly: Concepts (cont.)

- Wire: connects references to services

Wire

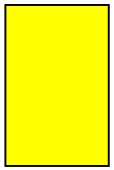
- Interface: description of business functions
  - Eg: java interface, WSDL 1.1 portType, WSDL interface
- Binding: access mechanisms used by services and reference
  - Eg: WSDL binding, JMS binding, EJB Session bean binding

SCA Assembly: Concepts (cont.)

- Component Implementation :
  - Configurable piece of software that provides some business function (Java class, BPEL process, Spring application context)
  - Support for *different implementation technologies*
    - e.g. Java™, Spring, BPEL, C++, PHP, XSLT…
    - implementation type *extensibility*
  - *composite* can also be used as an *implementation*
  - Provides business function via one or more *services*
  - Uses other services through service *references*
  - Service and references typed by *interfaces*
  - *Scoped*
    - Runtime managed state and message routing

SCA Assembly: Concepts (cont.)

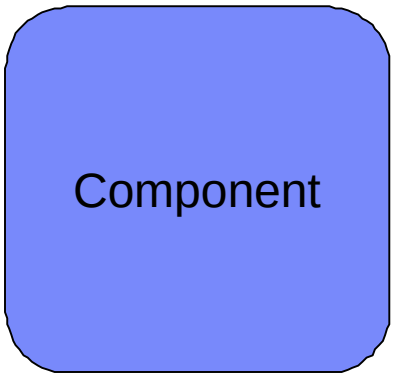
- Property: allow for the configuration of an implementation with externally set data values
  - Eg: an implementation that allows the value for ‘currency’ property to be set to any of the world currencies



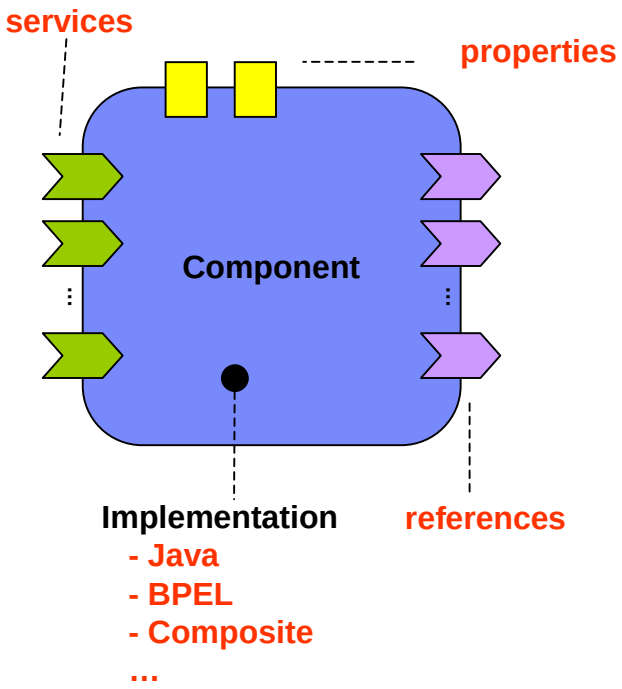
Property

SCA Assembly: Concepts (cont.)

- ***Component***
  - ***Configured*** instance of ***implementation*** within a Composite
  - more than one component can use same implementation
  - ***Provides*** and ***consumes services***
  - Sets implementation ***properties***
  - Sets service ***references*** by ***wiring*** them to services
    - wiring to services provided by other components or by references of the composite

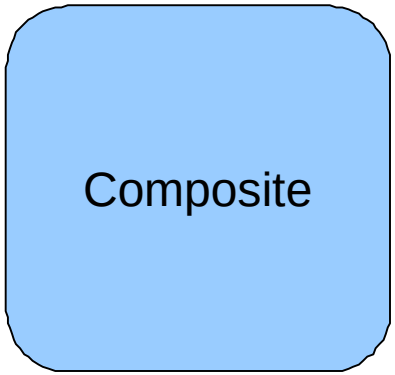


SCA Assembly Concepts (cont.)



SCA Assembly: Concepts (cont.)

- Composite: basic unit of composition.
  - ***Assembly*** of ***service components*** developed and deployed together
  - Consisting of:
    - public ***services***
    - service implementations organized as ***components***
    - required services as ***references***
    - ***wires*** connect components, services, and references
    - ***properties***
  - May be used as ***implementation*** of components at next higher layer



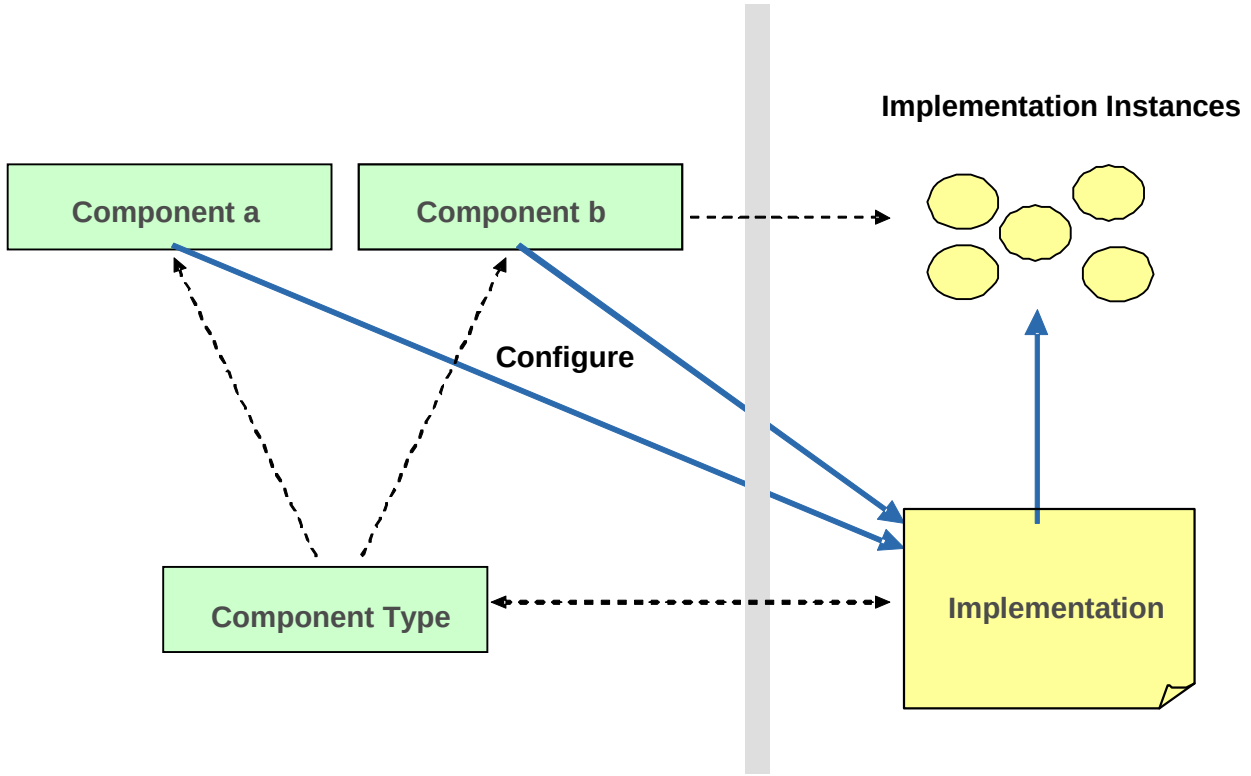
SCA Assembly: Concepts (cont.)

- ComponentType: description of an implementation
  - Specifies all the references, services, properties of an implementation
- ConstrainingType: template for an implementation, component or composite that specifies the shape in terms of references, services, properties

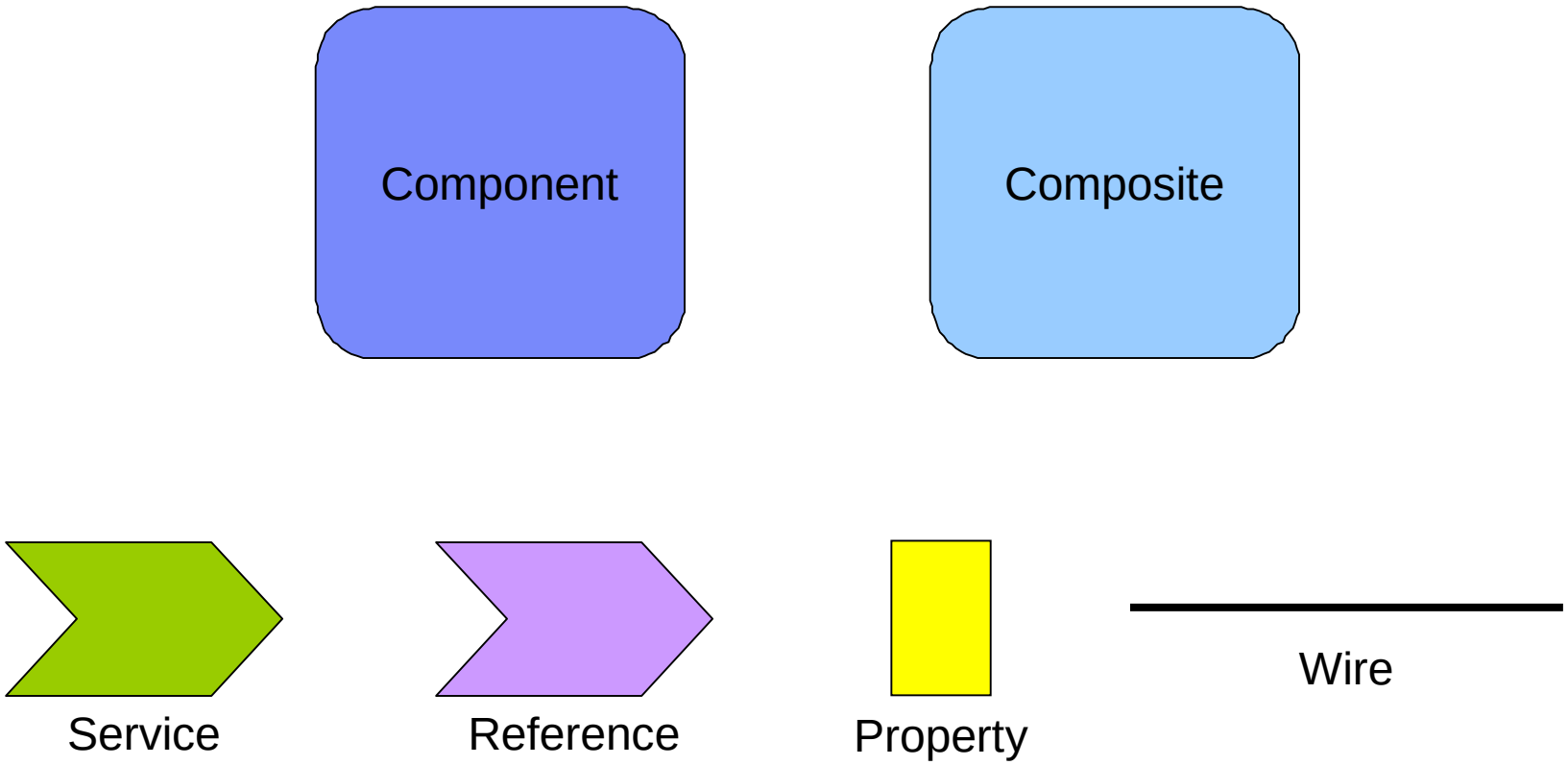
SCA ComponentType

```
<componentType>

    <service name="...”>
        ...
    </service>
    ...
    <service name="...”>
        ...
    </service>


    <reference name="...”>
        ...
    </reference>
    ...
    <reference name="...">
        ...
    </reference>


    <property name="...">
        ...
    </property>
    ...
    <property name="...">
        ...
    </property>

</componentType>
```

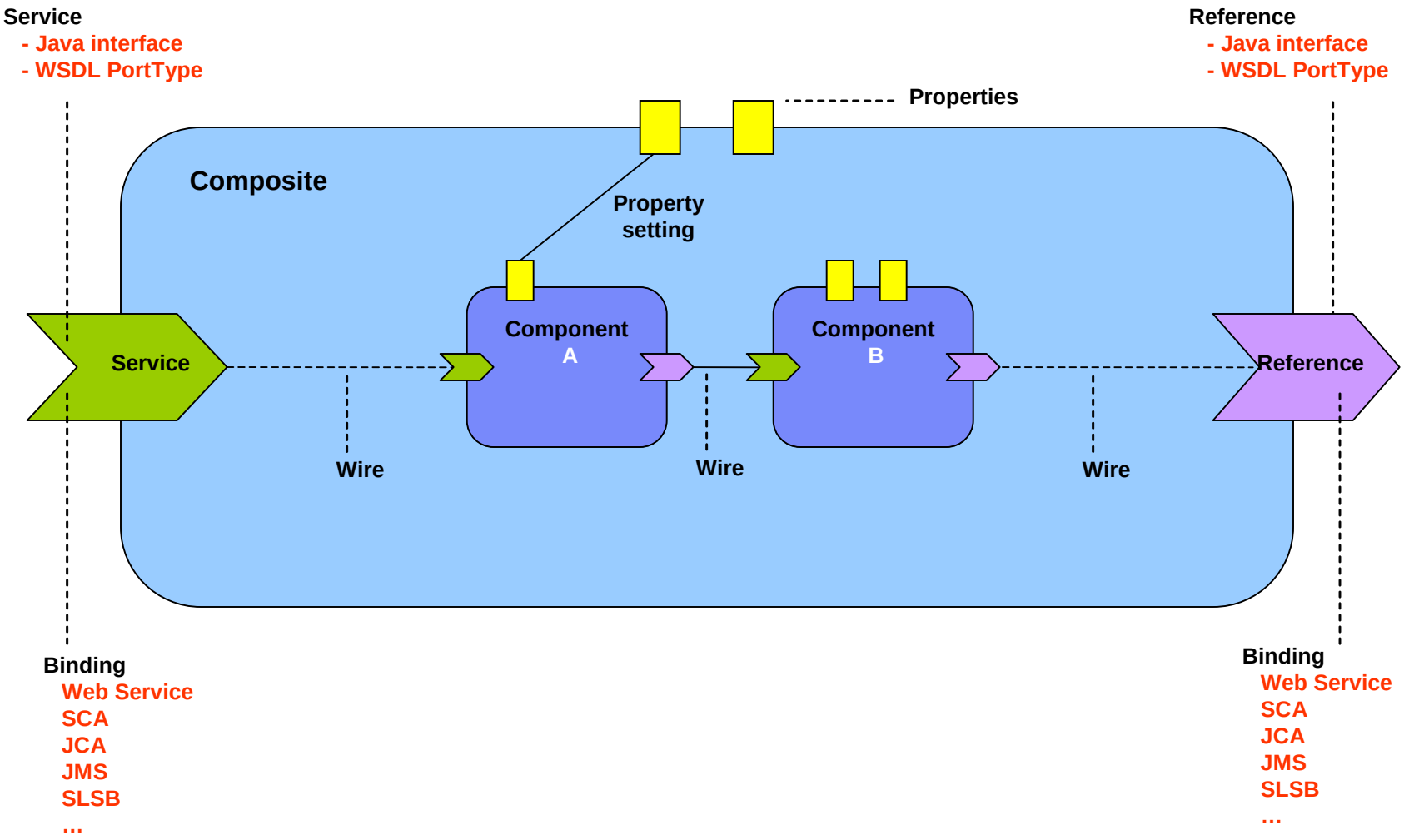
SCA Assembly Concepts (cont.)



SCA Assembly Symbols

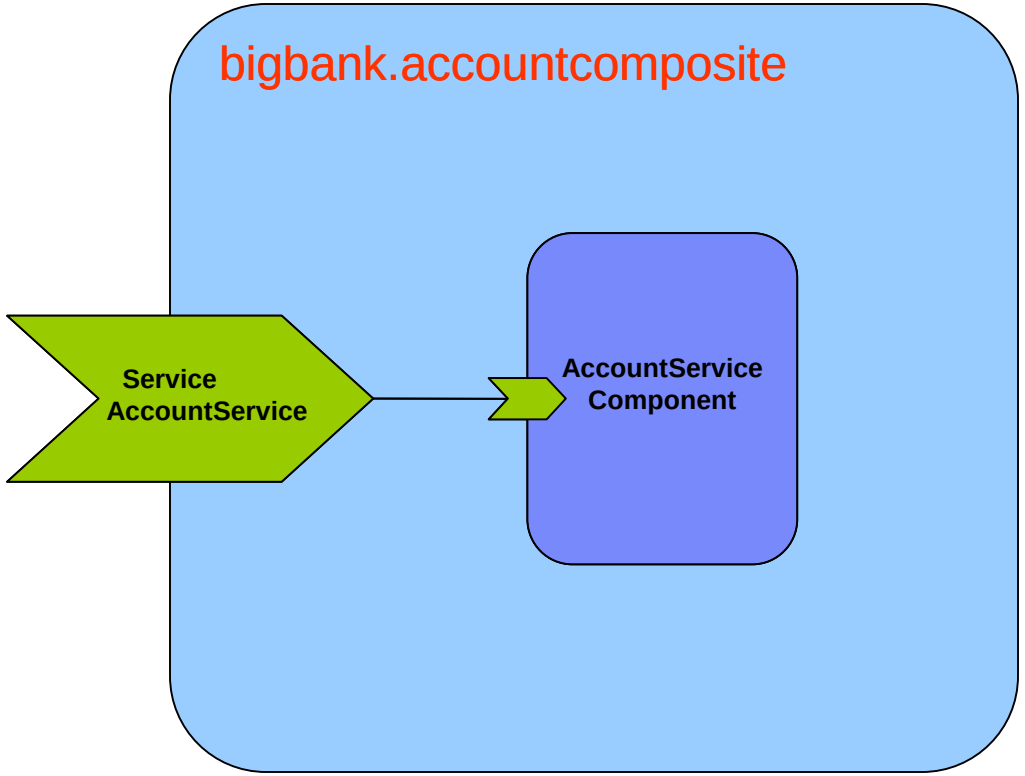


Composite


Service
- Java interface
- WSDL PortType
Reference
- Java interface
- WSDL PortType
Properties
Composite
Property setting
Component A
Component B
Service
Reference
Wire
Wire
Wire
Binding
Web Service
SCA
JCA
JMS
SLSB
...
Binding
Web Service
SCA
JCA
JMS
SLSB
...

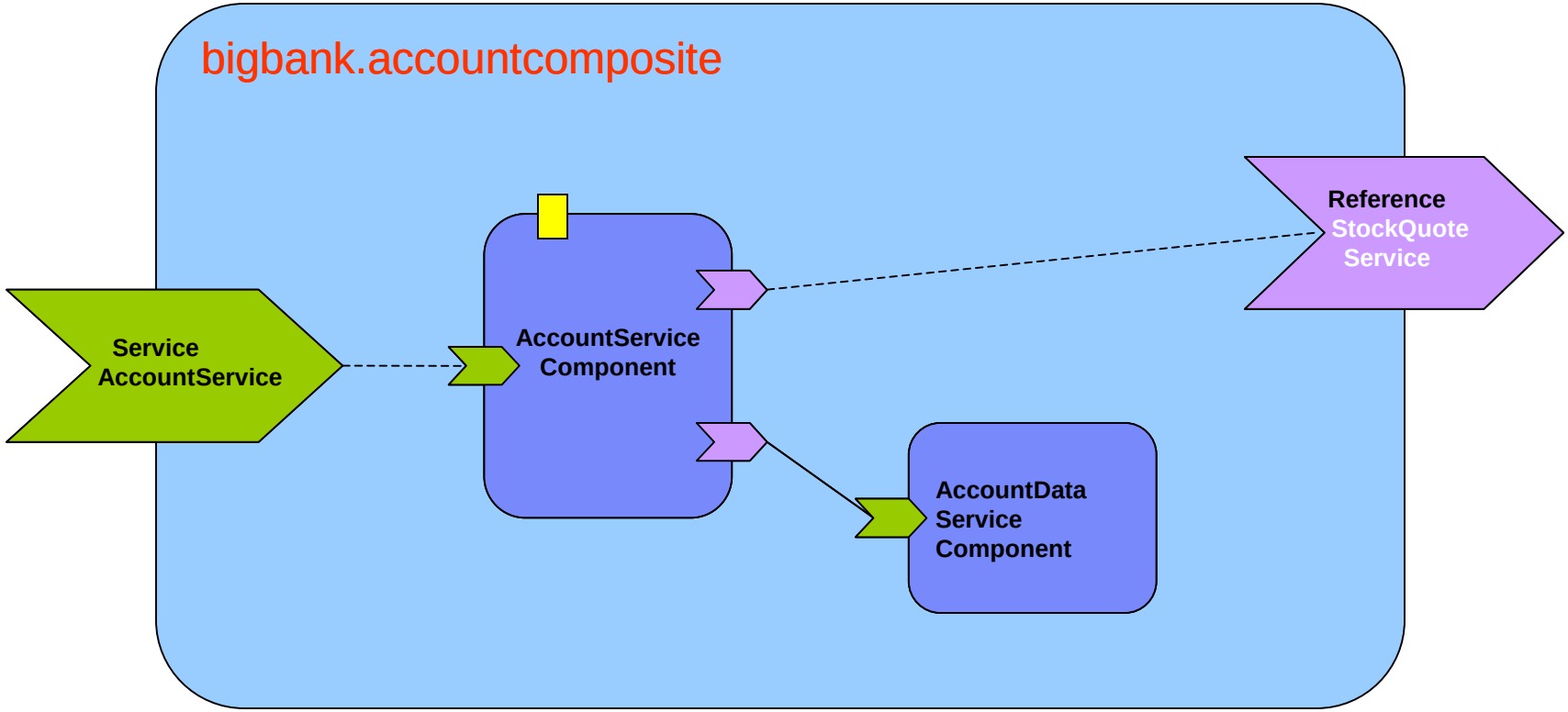
```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           targetNamespace="http://example.org“ name=“..." >

        <service name=“..." promote=“A"/>

        <component name=“A">
                <implementation ... />
                <property name=“...">...</property>
                <service name=“...">...</service>
                <reference name="" target="B"/>
        </component>
        <component name=“B">
                <implementation ... />
                <property name=“...">...</property>
                <property name=“...">...</property>
                <service name=“...">...</service>
                <reference name="“ .../>
        </component>


         <property name="...">...</property>
         <property name="...">...</property>


        <reference name="..."
                promote=“B"/>

</composite>
```

Bigbank Composite (1 service, 1 component)



```
<?xml version="1.0" encoding="ASCII"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
         targetNamespace="http://example.org"
          name="bigbank.accountcomposite" >

  <service name="AccountService" promote="AccountServiceComponent">
     <interface.java interface="services.account.AccountService"/>
     <binding.ws port="http://www.example.org/AccountService#
        wsdl.endpoint(AccountService/AccountServiceSOAP)"/>
  </service>

  <component name="AccountServiceComponent">
     <implementation.java class="services.account.AccountServiceImpl"/
  </component>

</composite>
```

Bigbank Composite (multiple components, service, ref & prop



```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           targetNamespace="http://example.org"
           name="bigbank.accountcomposite" >

    <service name="AccountService" promote="AccountServiceComponent">
        <interface.java interface="services.account.AccountService"/>
        <binding.ws port="http://www.example.org/AccountService#
          wsdl.endpoint(AccountService/AccountServiceSOAP)"/>
    </service>

    <component name="AccountServiceComponent">
        <implementation.java class="services.account.AccountServiceImpl"/>
        <reference name="StockQuoteService"/>
        <reference name="AccountDataService"
                   target="AccountDataServiceComponent/AccountDataService"/>
        <property name="currency">EURO</property>
    </component>

    <component name="AccountDataServiceComponent">
        <implementation.bpel process="“QName"/>
        <service name="AccountDataService">
            <interface.java interface="services.accountdata.AccountDataService"/>
        </service>
    </component>

    <reference name="" promote="AccountServiceComponent/StockQuoteService">
        <interface.java interface="services.stockquote.StockQuoteService"/>
        <binding.ws port="http://example.org/StockQuoteService#
          wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
    </reference>
<composite>
```

Recursive Composition

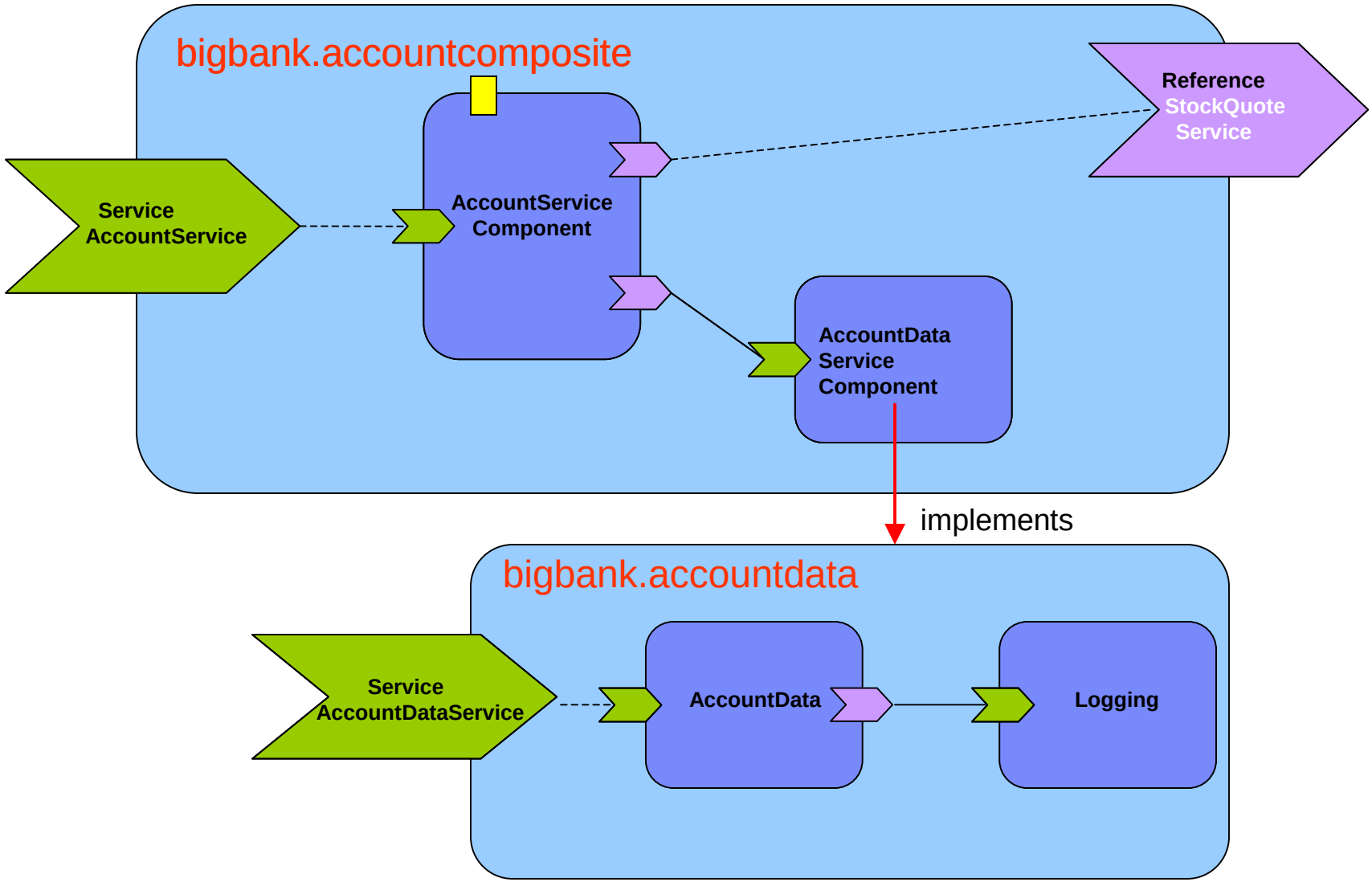
- Composites and Components look the same
  - Configured references
  - Configured properties
  - Configured services
- Composites have associated ComponentType
- Recursive composition
  - A composite can be used (nested) by another (higher-level) composite as a component implementation
  - Promotes reuse of assemblies
  - Uses “implementation.composite” as the component implementation
  - A component can be implemented by “simple” implementation or by a composite

AccountDataServiceComponent's ComponentType

- AccountDataServiceComponent's ComponentType

```
<componentType>
    <service name="AccountDataService">
        <interface.java interface="services.accountdata.AccountDataService"/>
    </service>
</componentType>
```

Implementing AccountDataService Using A Composite



bigbank.AccountData Composite

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           targetNamespace="http://example.org"
           name="bigbank.AccountData" >

    <service name="AccountDataService" promote="AccountData">
        <interface.java interface="services.accountdata.AccountService"/>
    </service>

    <component name="AccountDataServiceComponent">
        <implementation.bpel process="..."/>
        <reference name="LoggingService"
                   target="LoggingServiceComponent"/>
    </component>

    <component name="LoggingServiceComponent">
        <implementation.spring location="..."/>
    </component>

<composite>
```

OASIS

bigbank.account Composite (recursion)

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           targetNamespace="http://example.org"
           name="bigbank.accountcomposite" >
    <service name="AccountService" promote="AccountServiceComponent">
        <interface.java interface="services.account.AccountService"/>
        <binding.ws port="http://www.example.org/AccountService#
          wsdl.endpoint(AccountService/AccountServiceSOAP)"/>
    </service>
    <component name="AccountServiceComponent">
        <implementation.java class="services.account.AccountServiceImpl"/>
        <reference name="StockQuoteService"/>
        <reference name="AccountDataService"
                   target="AccountDataServiceComponent/AccountDataService"/>
        <property name="currency">EURO</property>
    </component>

    <component name="AccountDataServiceComponent">
        <implementation.composite name="bb:bigBank.AccoutData"/>
        <service name="AccountDataService">
            <interface.java interface="services.accountdata.AccountDataService"/>
        </service>
    </component>

    <reference name="" promote="AccountServiceComponent/StockQuoteService">
        <interface.java interface="services.stockquote.StockQuoteService"/>
        <binding.ws port="http://example.org/StockQuoteService#
          wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
    </reference>
<composite>
```

Local v. Remotable Interfaces

- Supports multiple components within a single process or separate processes
- Local interface
  - Pass-by-reference
  - Tightly-coupled
  - Fine-grained
- Remote interface
  - Pass-by-value (with pass-by-reference override)
  - Loosely-coupled
  - Coarse-grained
- Java interface
  - Local: default
  - Remotable: using @remotable annotation
- WSDL portType/interface: always remote
- ‘local’ attribute override on the <composite> element

Bidirectional Interfaces (Callbacks)

- Useful for asynchronous messaging
- Support for callbacks using Java interfaces

```
<interface.java interface="services.invoicing.ComputePrice"
        callbackInterface="services.invoicing.InvoiceCallback"/>
```

- Support for callbacks using WSDL portTypes/interfaces

```
<interface.wsdl interface="http://example.org/inv#
                        wsdl.interface(ComputePrice)"
                callbackInterface="http://example.org/inv#
                        wsdl.interface(InvoiceCallback)"/>
```

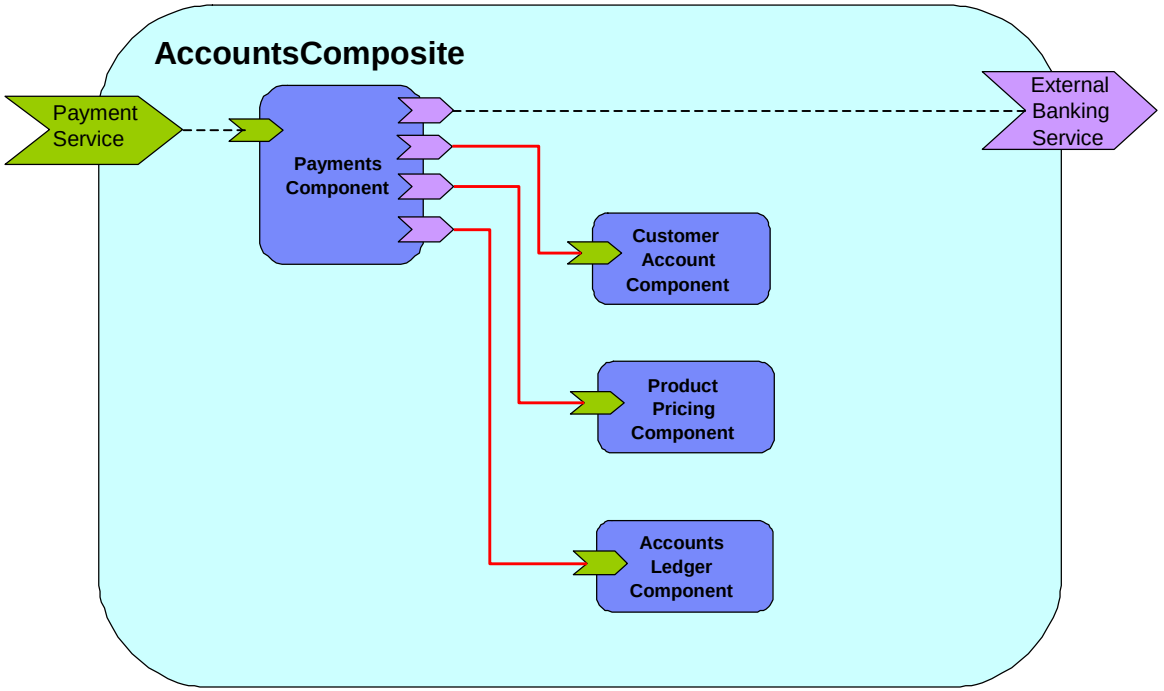
Conversational Interfaces

- Frees application programmer from conversation/correlation management
- Imposes requirements on bindings
- Interfaces marked as conversational using SCA Policy intent
- Specific operations can be marked as “endsConversation”
- WSDL extensions for “conversational” and “endsConversation”

```
<portType name="LoanService" sca:requires="conversational" >
       <operation name="apply">
          <input message="tns:ApplicationInput"/>
          <output message="tns:ApplicationOutput"/>
       </operation>
       <operation name="cancel" sca:endsConversation="true" >
       </operation>
       ...
</portType>
```

Autowiring

- Allows component references to be wired to component services automatically (without explicit wires)
- Matches references to services based on compatible interfaces, bindings, policy intents/sets



AccountsComposite without Autowiring

```
<composite name="AccountComposite" ...>
    <service name="PaymentService" promote="PaymentsComponent"/>
    <component name="PaymentsComponent">
      <implementation ... />
        <service name="PaymentService"/>
        <reference name="CustomerAccountService“ target="CustomerAccountComponent"/>
        <reference name="ProductPricingService" target="ProductPricingComponent"/>
        <reference name="AccountsLedgerService" target="AccountsLedgerComponent"/>
        <reference name="ExternalBankingService"/>
    </component>
    <component name="CustomerAccountComponent">
        <implementation ... />
    </component>
    <component name="ProductPricingComponent">
        <implementation ... />
    </component>
    <component name="AccountsLedgerComponent">
        <implementation ... />
    </component>
    <reference name="ExternalBankingService“
   promote="PaymentsComponent/ExternalBankingService"/>
</composite>
```

AccountsComposite with Autowiring

```
<composite name="AccountComposite" ...>
    <service name="PaymentService" promote="PaymentsComponent"/>
    <component name="PaymentsComponent" autowire="true" >
      <implementation ... />
        <service name="PaymentService"/>
        <reference name="CustomerAccountService“/>
        <reference name="ProductPricingService"/>
        <reference name="AccountsLedgerService"/>
        <reference name="ExternalBankingService"/>
    </component>
    <component name="CustomerAccountComponent">
        <implementation ... />
    </component>
    <component name="ProductPricingComponent">
        <implementation ... />
    </component>
    <component name="AccountsLedgerComponent">
        <implementation ... />
    </component>
    <reference name="ExternalBankingService“
   promote="PaymentsComponent/ExternalBankingService"/>
</composite>
```

Composite Inclusion

- Allows composite reuse through inclusion
  - Equivalent to inlining of all composite children
  - Uses <sca:include> element

```
<composite name="IncludingComposite" ...>
    <include name="myns:IncludedComposite"/>
    <service name="myservice" promote="mycomponent" ... />
</composite>

<composite name="IncludedComposite" ...>
    <component name="mycomponent" .../>
</composite>
```

- Resulting Composite:

```
<composite name="IncludingComposite" ...>
    <component name="mycomponent" .../>
    <service name="myservice" promote="mycomponent" ... />
</composite>
```

Reuse in SCA

- Inclusion
- Recursive composition
- Implementation reuse through configurable components
- Reusable services through composite references

Top-Down Design: constrainingType

- constrainingType
  - Implementation independent
  - Specifies the shape -- constraints in terms of services/references/properties
  - composites, components, componentType and implementations can be constrained using the “constrainingType” attribute
  - Allows an architect to specify constrainingTypes which can be used by developers as a template
  - SCA provides runtime validation of artifacts with its constrainingType

constrainingType Example

```
<constrainingType name="myCT" ... >
    <service name="MyValueService">
        <interface.java interface="services.myvalue.MyValueService"/>
    </service>
    <reference name="customerService">
        <interface.java interface="services.customer.CustomerService"/>
    </reference>
    <property name="currency" type="xsd:string"/>
</constrainingType>
```

```
<component name="MyValueServiceComponent" constrainingType="myns:CT" >
    <implementation.bpel process="..."/>
    <service name="MyValueService">
        <interface.java interface="services.myvalue.MyValueService"/>
        <binding.jms .../>
    </service>
    <reference name="customerService" target="CustomerService">
        <binding.ws ...>
    </reference>
    <property name="currency">EURO</property>
</component>
```

Extensibility in SCA

- Designed for extensibility
- Extensible artifacts:
  - Implementation types
    - <implementation.*>
  - Interface types
    - <interface.*>
  - Binding types
    - <binding.*>

Packaging and Deployment: Domains

- Composites deployed, configured into ***SCA Domain***
- Domain contains ***components***, ***services***, ***references***, ***wires***
  - configured using ***composites***
- Composites make deployment simpler
  - individual composites created, deployed independently
  - may contain only wires, components or externally provided services or references
- Defines the boundary of visibility for SCA
- Represents the complete SCA runtime
  - Potentially distributable
- Typically represents an area of functionality controlled by a single organization/division
  - E.g.: accounts
- Represented by a virtual composite
- Contains installed contributions
- Abstract services for management of the domain

Packaging and Deployment: Contributions

- Package containing artifacts necessary for SCA
  - SCA defined artifacts
    - E.g.: composites, constrainingType, etc
  - Non-SCA defined artifacts
    - E.g.: WSDL, XML schema, Java classes, object code etc
- Packaging must be hierarchical
- Metadata included in the “META-INF” directory

```
<contribution xmlns=http://www.osoa.org/xmlns/sca/1.0>
    <deployable composite="xs:QName"/>*
    <import namespace="xs:String" location=”xs:AnyURI”?/>*
    <export namespace="xs:String"/>*
</contribution>
```

- Interoperable packaging format: ZIP
- Other formats possible: filesystem directory, OSGi bundle, JAR file

Assembly: Summary

- SCA Assembly models systems composed of reusable services
- A model for service-based system:
  - construction
  - assembly
  - deployment
- Heterogeneity
  - Multiple languages
  - Multiple container technologies
  - Service access methods
- Metadata driven

Java Common Annotations

- Java Annotations for generating corresponding componentType
- Common across all Java-related specifications
- Implementation annotations
  - @Service
  - @Reference
  - @Property
  - @Scope @Init @Destroy @EagerInit
  - @ConversationID @ConversationAttributes
  - @ComponentName
  - @Constructor
- Interface annotations
  - @AllowsPassByReference
  - @Callback
  - @Remotable
  - @Conversational
  - @Oneway

Java Annotation Example

```
package services.account;
...
public class AccountServiceImpl implements AccountService {
    @Property
    private String currency = "USD";

    @Reference
    private AccountDataService accountDataService;

    @Reference
    private StockQuoteService stockQuoteService;

    ...

    public AccountReport getAccountReport(String customerID) {
        ...
    }

    ...
}
```

Java Annotation Example: componentType

```
<componentType xmlns="http://www.osoa.org/xmlns/sca/1.0"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="AccountService">
        <interface.java interface="services.account.AccountService"/>
    </service>

    <reference name="accountDataService">
        <interface.java interface="services.accountdata.AccountDataService"/>
    </reference>

    <reference name="stockQuoteService">
        <interface.java interface="services.stockquote.StockQuoteService"/>
    </reference>

    <property name="currency" type="xsd:string" default="USD"/>

</componentType>
```

Java Common APIs

- Common across all Java-related specifications
- APIs for:
  - Component context
  - Request context
  - Callable reference
  - Service reference
  - Conversation
  - Exceptions

Java Component Implementation

- Implementation type for POJO
  - Uses <implementation.java class="java-class-name">
- Uses the common annotations and APIs

```
<component name="AccountServiceComponent">
    <implementation.java class="services.account.AccountServiceImpl"/>
    ...
</component>
```

Spring Framework Component Implementation

- Integration at a coarse-grained level
- Spring application context can be used as a component implementation
- Two ways:
  - No SCA-related metadata in Spring
  - SCA-related metadata specified as Spring beans
    - sca:service
    - sca:reference
    - sca:property
- Uses <implementation.spring location="spring-application-context"/>

```
<component name="AccountServiceComponent">
    <implementation.spring location="/spring/application-context/"/>
    ...
</component>
```

Bigbank implementation using Spring

- AccountDataService component

```
<beans>
    <bean id="AccountDataService" class="bigbank.account.AccountDataService"/>
</beans>
```

- AccountService component

```
<beans>

    <bean id="X">
        <property name="aPropertyName" ref="Y"/>
    </bean>

    <bean id="Y">
        <property name="SQRef" ref="StockQuoteService"/>
        <property name="ADRef" ref="AccountDataService"/>
    </bean>

    <sca:service name="AccountService" type="bigbank.account.AccountService" target="X"/>

    <sca:reference name="StockQuoteService" type="bigbank.account.StockQuoteService"/>

    <sca:reference name="AccountDataService" type="bigbank.account.AccountDataService"/>

<beans>
```

BPEL Component Implementation

- SCA and BPEL are complementary
  - BPEL provides business orchestration view of the component
  - SCA provides a compositional view of interconnection among service components
- Supports WS-BPEL 1.1 and 2.0
- Requires WSDL interfaces
- SCA service = partnerLink with a single role belonging to the BPEL process
- SCA reference = partnerLink with a single role belonging to a partner
- When partnerLink defines two role, directionality defines whether it is a service or a reference
- SCA extensions
  - Attribute “sca:property” on a variable declaration defines a property
  - Element “sca:multiReference” on a variable declaration defines a multivalued reference

BPEL Component Implementation (cont.)

- Uses <implementation.bpel process="bpel-process-QName"/>

```
<component name="AccountServiceComponent">
    <implementation.bpel process="myns:Process1"/>
    ...
</component>
```

Web Service Binding

- WSDL-based
- Supports WSDL 1.1 and WSDL 2.0
- Two ways to specify a WS binding
  - Reference an existing WSDL binding/service/endpoint/port element
  - Specify metadata to synthesize a SOAP-based WSDL binding

```
<binding.ws wsdlElement="xs:anyURI"?
            wsdli:wsdlLocation="list of xs:anyURI"?>
    <wsa:EndpointReference>...</wsa:EndpointReference>*
</binding.ws>
```

Web Service Binding Examples

- Point to an existing WSDL document

```
<binding.ws wsdlElement="http://www.stockquote.org/StockQuoteService#
                         wsdl.service(StockQuoteService)"
            wsdli:wsdlLocation="http://www.stockquote.org/StockQuoteService
            http://www.stockquote.org/StockQuoteService.wsdl" />
```

- Synthesize WSDL

```
<binding.ws uri="http://www.sqs.com/StockQuoteService"/>
```

  - Defaults to SOAP/HTTP binding
  - Defaults to document/literal

JMS Binding

- Based on JMS API
- Allows JMS headers and user properties to be set on a per-operation basis
- Support for callbacks and conversations
- Default data binding and operation selection
- Uses <binding.jms .../>
- Example:

```
<binding.jms>
    <destination name=”StockQuoteServiceQueue”/>
    <connectionFactory name=”StockQuoteServiceQCF”/>
    <resourceAdapter name=”com.example.JMSRA”/>
</binding.jms>
```

EJB Session Bean Binding

- Support stateless and stateful session beans
- Stateful session bean implies conversational
- Covers both exposure and consumption usecases
- Support EJB 2.x and 3.0
- Uses <binding.ejb .../>
- Example

```
<binding.ejb uri="corbaname:rir:#ejb/JobBankServiceHome“
        homeInterface="com.app.jobbank.JobBankServiceHome“
        ejb-link-name="jobbankEJB.jar#JobBankComponent"/>
```

Future Work

- Work will continue in the OSOA collaboration
  - SCA Eventing Model
  - SCA Client and Implementation Model for PHP
    - other scripting languages being investigated
  - SCA Client and Implementation Model for COBOL
  - JCA Binding
  - Java EE Integration
  - other implementation languages & frameworks may follow
- SCA 1.0 to be standardized in OASIS

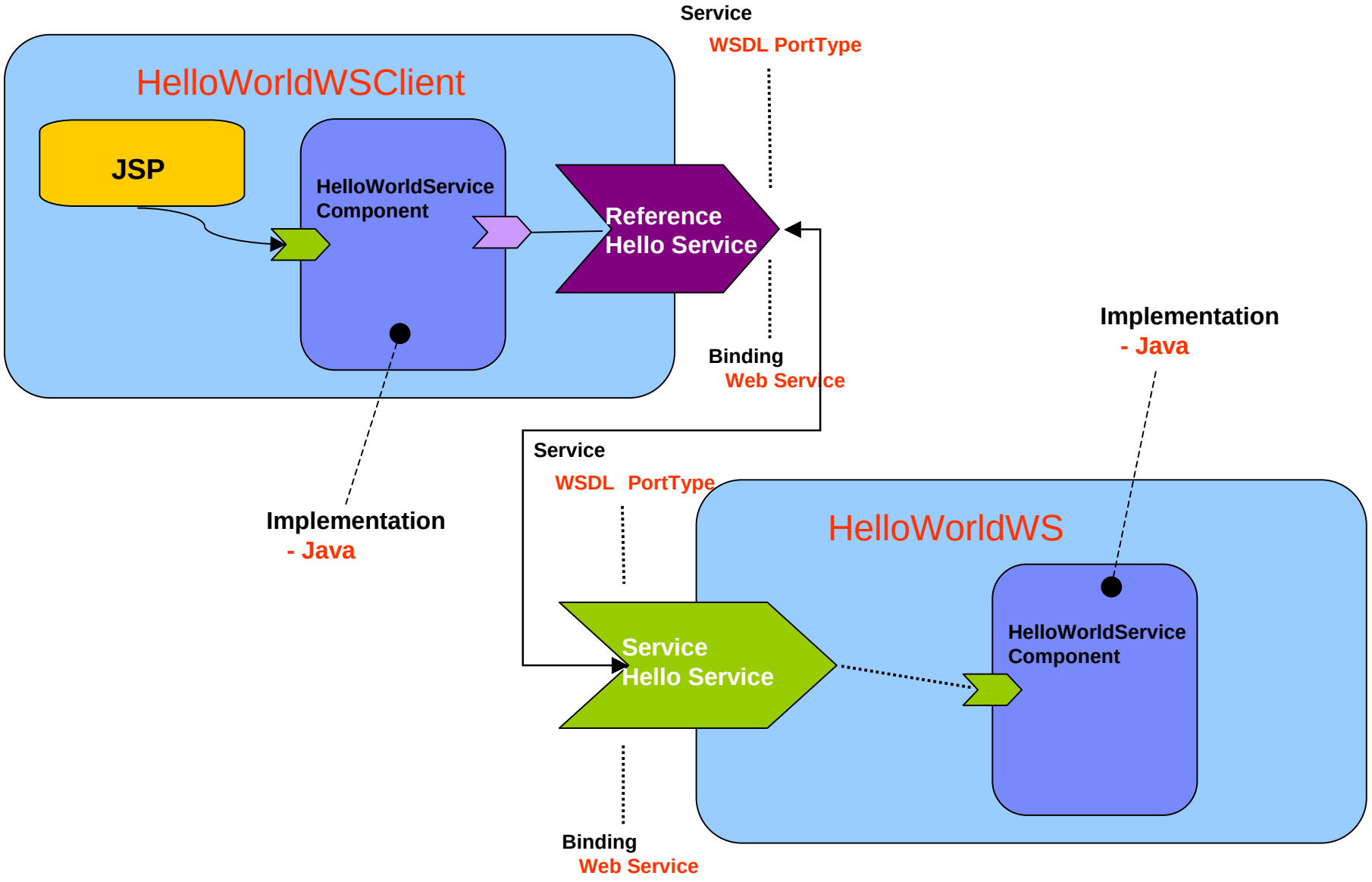
Summary

- SCA models systems built using a Service Oriented Architecture
  - supports Service Implementation, Service Assembly
  - open to many kinds of service implementation
  - open to many types of service access
  - declarative intent & policy approach to application of Security & Transaction

Demo

- IBM Demo:
  - Simple HelloWorld sample with a callback (SCA V0.95)
- Oracle Demos:
  - Oracle Fabric (SCA 0.95) Overview
  - Simple SayHello Example Using Jdeveloper (Synchronous)
  - Simple HelloWorld Example (Asynchronous)
  - OrderBooking Application

HelloWorld Sample with Callback



HelloWorld Sample with Callback - Client

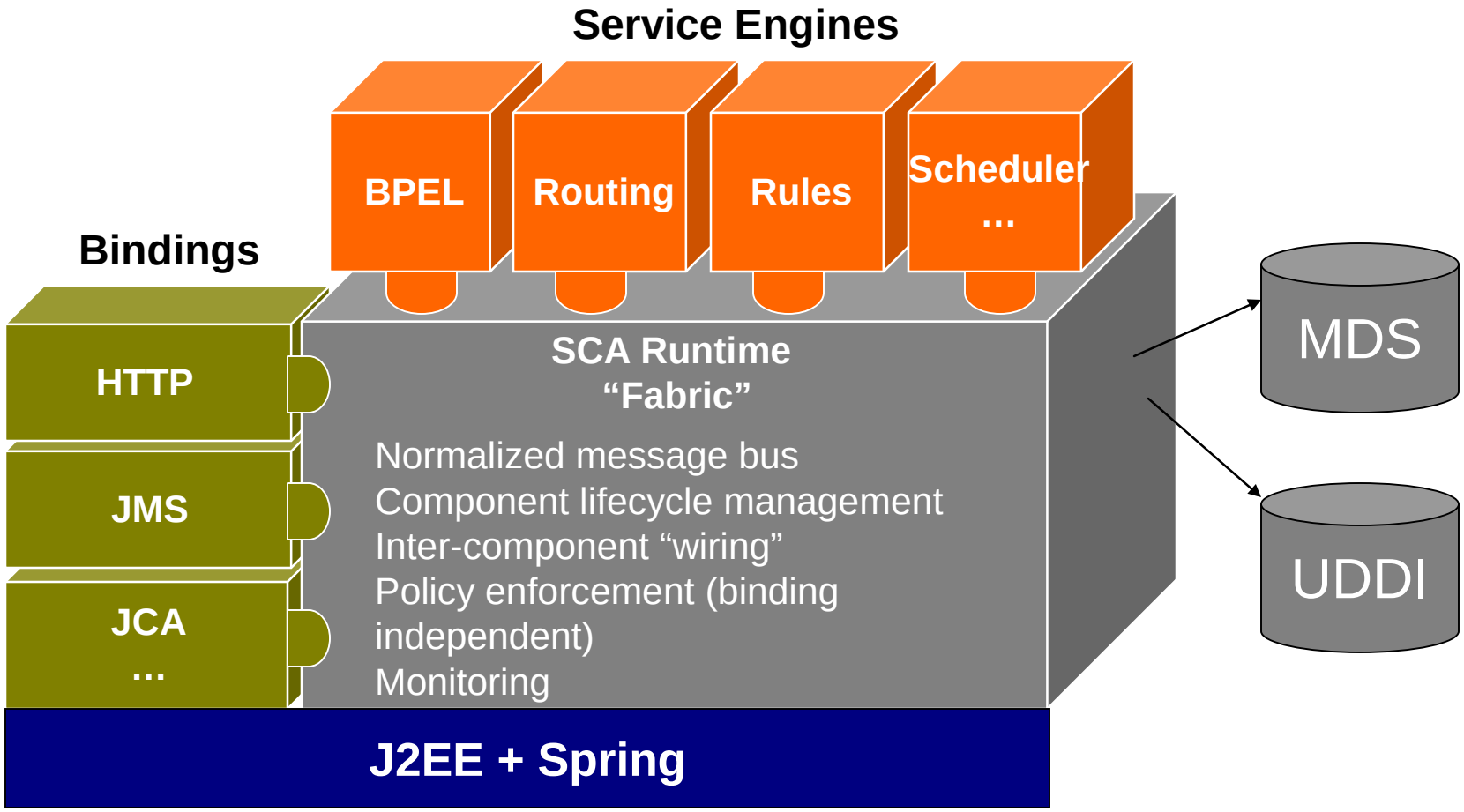
```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           xmlns:system="http://tuscany.apache.org/xmlns/system/1.0-SNAPSHOT"
           name="helloworldwsclient">
    <component name="HelloWorldServiceComponent">
      <implementation.java class="helloworld.HelloWorldServiceComponent"/>
      <reference name="helloWorldService">HelloWorldService</reference>
    </component>

    <reference name="HelloWorldService">
        <interface.wsdl xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
            interface="http://helloworld#wsdl.interface(HelloWorld)"
            callbackInterface="http://helloworld#wsdl.interface(HelloWorldCallback)"
            wsdli:wsdlLocation="http://helloworld wsdl/helloworld.wsdl" />

        <binding.ws endpoint="http://helloworld#
            wsdl.endpoint(HelloWorldService/HelloWorldSoapPort)"
            location="wsdl/helloworld.wsdl" />
    </reference>
</composite>
```

HelloWorld Sample with Callback - Service

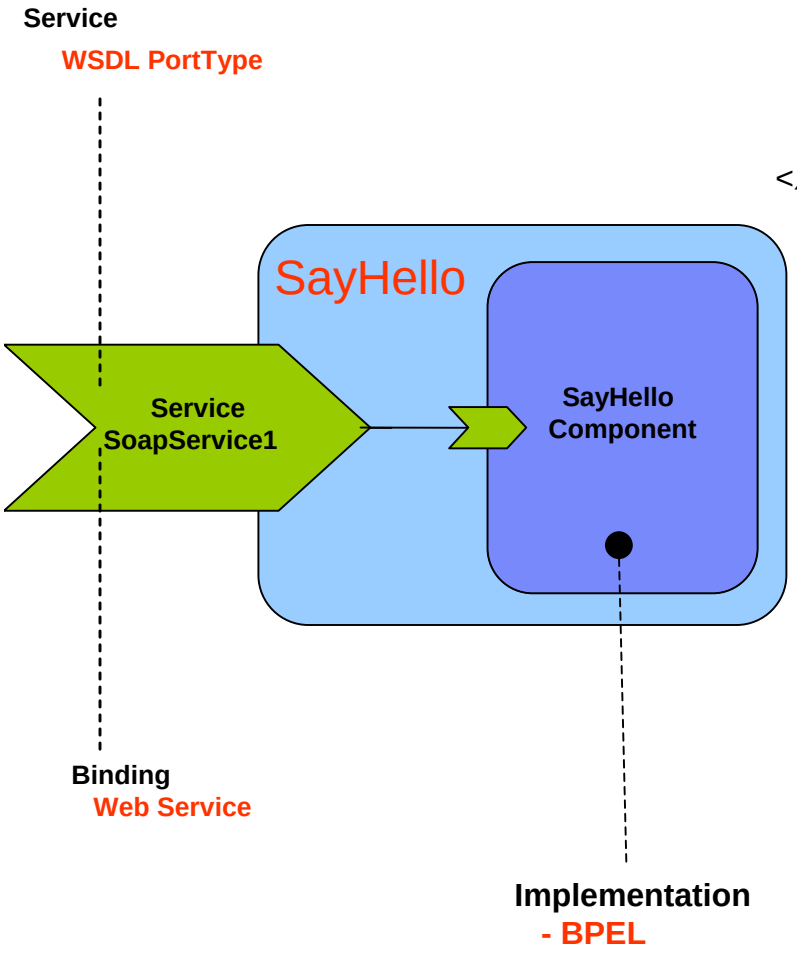
```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
           name="helloworldws">

    <service name="hwwsasync">
        <interface.wsdl xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
            interface="http://helloworld#wsdl.interface(HelloWorld)"
            callbackInterface="http://helloworld#wsdl.interface(HelloWorldCallback)"
            wsdli:wsdlLocation="http://helloworld WEB-INF/wsdl/helloworld.wsdl" />

        <binding.ws endpoint="http://helloworld#
            wsdl.endpoint(HelloWorldService/HelloWorldSoapPort)"
            location="WEB-INF/wsdl/helloworld.wsdl" />
        <reference>HelloWorldServiceComponent</reference>
    </service>

    <component name="HelloWorldServiceComponent">
        <implementation.java class="helloworld.HelloWorldImpl" />
    </component>

</composite>
```

OASIS

Demo: Fabric Runtime (SCA 0.95) Overview



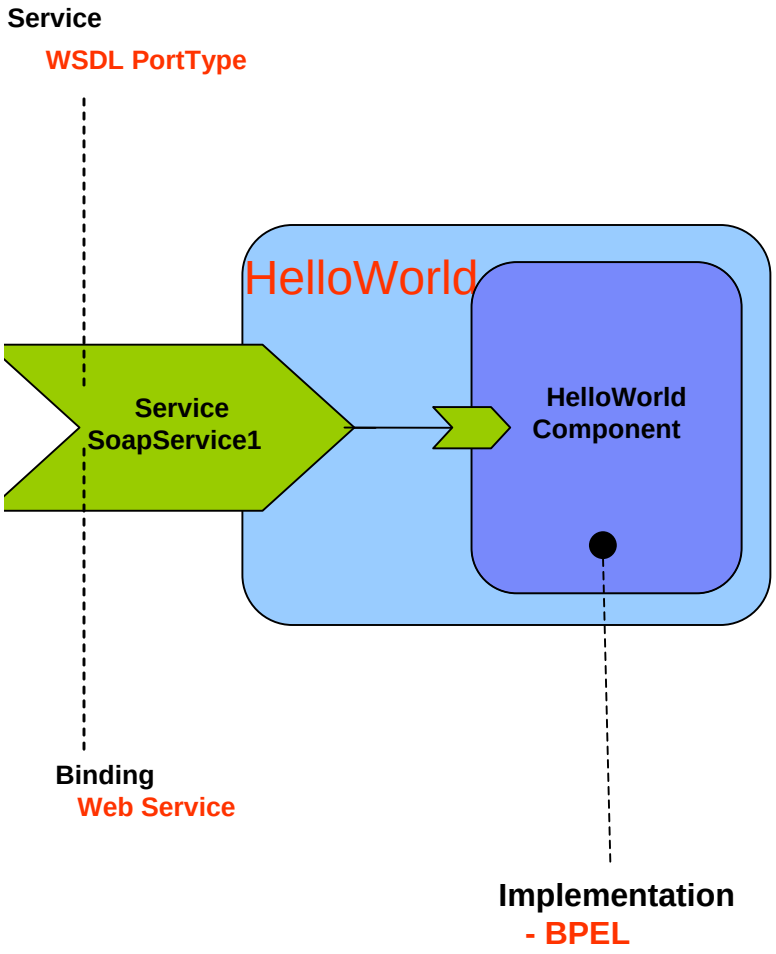
SayHello Example (Synchronous)




```
<componentType ...>
  <service name="client">
    <interface.wsdl
      interface="http://xmlns.oracle.com/SayHello#
       wsdl.interface(SayHello)"/>
  </service>
</componentType>
```

```
<composite name="SayHello“ ... >
  <service name="SoapService1">
    <interface.wsdl
      interface="http://xmlns.oracle.com/SayHello#
       wsdl.interface(SayHello)"/>
    <binding.ws
        port="http://xmlns.oracle.com/SayHello#
         wsdl.endpoint(SoapService1/SayHello_pt)"/>
  </service>
  <component name="SayHello">
    <implementation.bpel src="SayHello.bpel"/>
  </component>
  <wire>
    <source.uri>SoapService1</source.uri>
    <target.uri>SayHello/client</target.uri>
  </wire>
</composite>
```

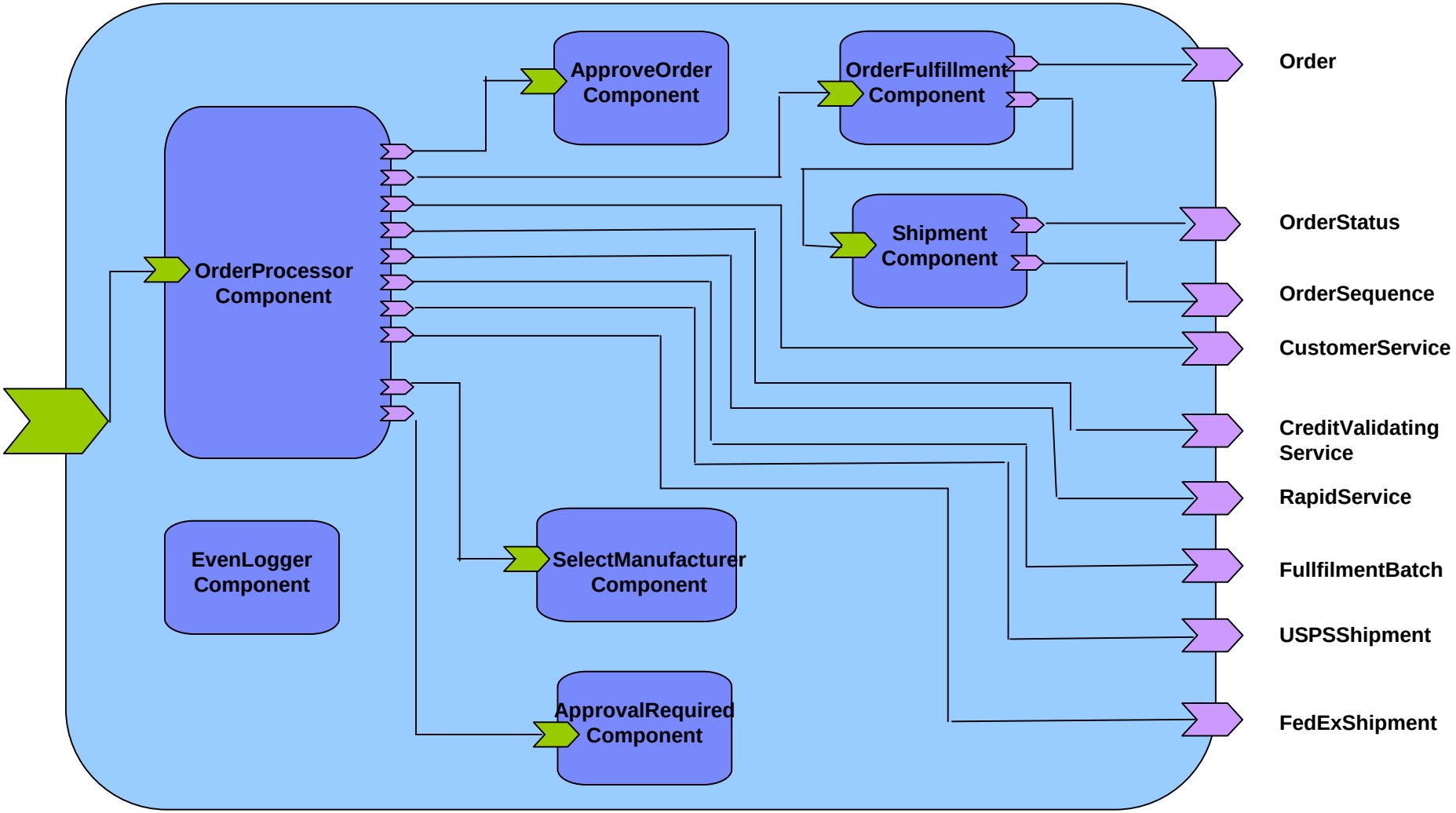
HelloWorld Example (Asynchronous)




```
<componentType ...>
  <service name="client">
    <interface.wsdl
      interface="http://samples.otn.com/helloworld#
          wsdl.interface(HelloWorld)"
      callbackInterface="http://samples.otn.com/helloworld#
          wsdl.interface(HelloWorldCallback)"/>
    </service>
</componentType>
```

```
<composite name="HelloWorld“ ...>
  <service name="client">
    <interface.wsdl
      interface="http://samples.otn.com/helloworld#
        wsdl.interface(HelloWorld)"
      callbackInterface="http://samples.otn.com/helloworld#
        wsdl.interface(HelloWorldCallback)" />
    <binding.ws port="http://samples.otn.com/helloworld#
                  wsdl.endpoint(HelloWorld/HelloWorld)"/>
  </service>
  <component name="HelloWorld">
    <implementation.bpel src="HelloWorld.bpel"/>
  </component>
  <wire>
    <source.uri>client</source.uri>
    <target.uri>HelloWorld/client</target.uri>
  </wire>
</composite>
```

OrderBooking Application



```
<composite name="OrderProcessing" ...>
    <service name="Client">
        <interface.wsdl interface="http://www.globalcompany.com/ns/OrderBooking#
                    wsdl.interface(SOAOrderBooking)"/>
        <binding.ws port="http://www.globalcompany.com/ns/OrderBooking#
                    wsdl.endpoint(OrderBooking/OrderBookingPort)"/>
    </service>
    <component name="OrderFulfillment">
        <implementation.mediator src="OrderFulfillment.mplan"/>
    </component>
    <component name="ApproveOrder">
        <implementation.workflow src="ApproveOrder.task"/>
    </component>
    <component name="ApprovalRequired">
        <implementation.decision src="DecisionService.decs"/>
    </component>
    <component name="SelectManufacturer">
        <implementation.bpel src="SelectManufacturer.bpel"/>
    </component>
    <component name="EventLogger">
         <implementation.eventAgent
                    className="oracle.integration.platform.blocks.event.agent.LoggingEventAgent"/>
         <business-events>
             <subscribeAll consistency="guaranteed"/>
         </business-events>
    </component>
    <reference name="Order">
            <interface.wsdl interface="http://xmlns.oracle.com/pcbpel/adapter/db/Order/#
                    wsdl.interface(Order_ptt)"/>
            <binding.jca config="OrderService_db.jca"/>
    </reference>

    ...

</composite>
```

Thank you!

Questions?