Service Component Architecture: An Overview

Jim Marino

BEA System

Office of the CTO

27 March 2006

bea

Think liquid.

Agenda

- SCA Collaboration Overview
- SCA Technical Overview

bea Think liquid.

Agenda

- SCA Collaboration Overview
- SCA Technical Overview

bea Think liquid.

SCA in Context I

- A key element of next-generation, SOA infrastructure
  - Define a standards base for next-generation service infrastructure
    - SCA:SOA as J2EE:App Server
- Collaboration of companies
  - BEA, IBM, Oracle, SAP, IONA, Sybase, Interface21 (Spring)
  - Launched in November 2005
- Royalty-free specifications
  - SCA Assembly Model (v0.9) Specification
  - SCA Client & Implementation Model for Java (v0.9) Specification
  - SCA Client & Implementation Model for C++ (v0.9) Specification
  - More specifications to come (Binding and Policy Specification, Container Extensibility Specification)

bea Think liquid.

SCA in Context II

- Tuscany open source runtime
  - An Apache umbrella project for new SOA infrastructure based on SCA and SDO
    - Java and C++ SCA runtime
    - SCA tooling
    - Java-based SDO implementation
- Leverage open source as part of the specification process
  - Provides valuable, real-world experience
  - Leverage innovative capability of the open source community
  - Provide a way for individuals and other organizations to contribute to the specification
- SCA will be taken to a standards body when appropriate
- Spec and open-source feedback cycle targeted for Fall 06

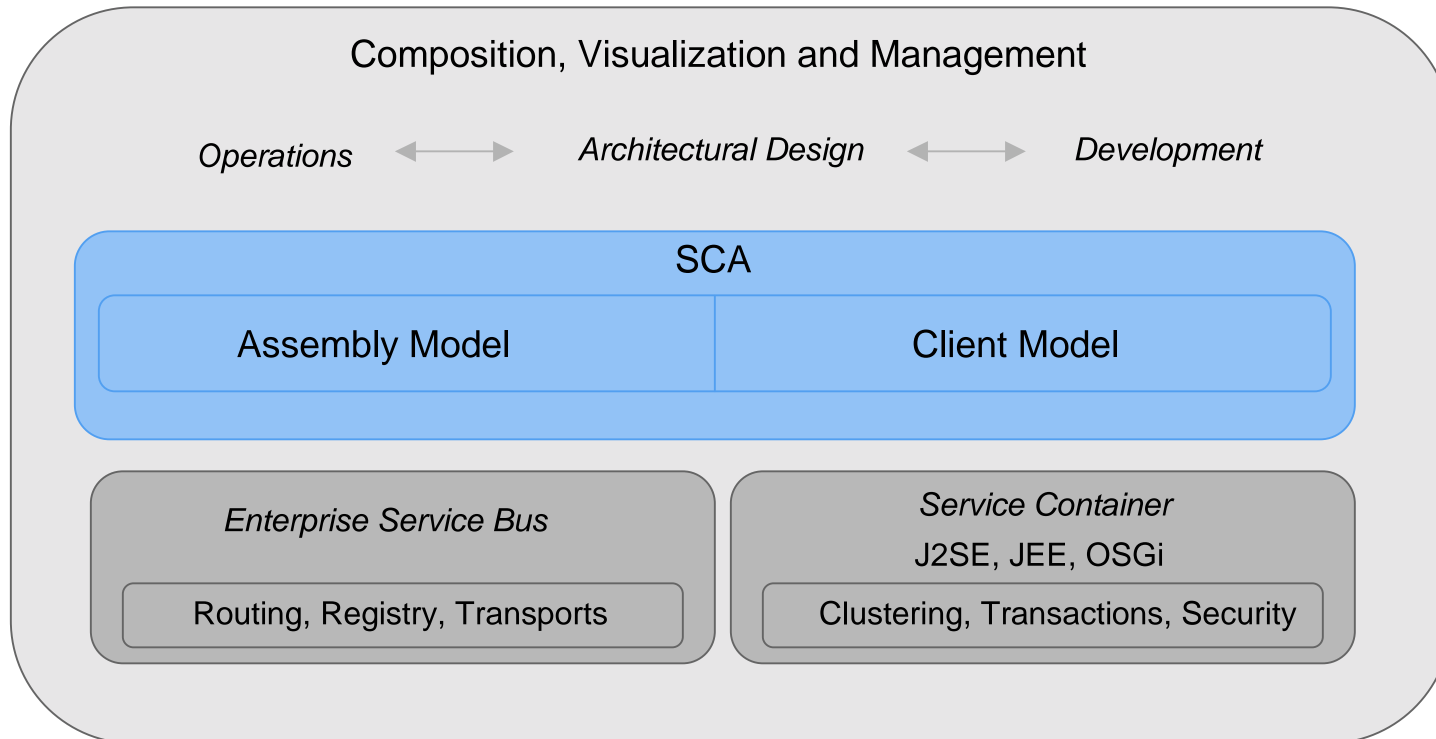
bea Think liquid.

SCA Technology in a Nutshell

- SCA models the “A” in SOA - for systems composed of reusable services
  - A model for service-based system:
    - construction
    - assembly
    - deployment
  - Heterogeneity – supports components from
    - multiple languages
    - multiple container technologies

bea Think liquid.

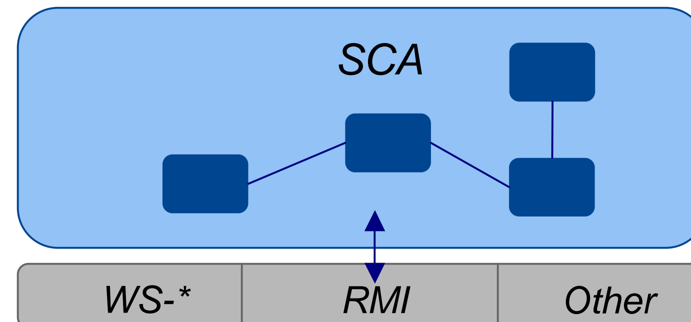
SCA and SOA infrastructure



bea Think liquid.

SCA and Wire Protocols

- SCA provides a unified declarative model for describing service implementations
  - Enables easier composition of service networks
  - Web Services = wire protocols, not implementations
- SCA provides a business-level model for implementing services
  - No technical APIs like JDBC, JCA, JMS, …
- SCA does not require Web Service technologies (e.g. SOAP over HTTP)
  - Also can use Java RMI, JCA…
  - Java interfaces are good, as are WSDL PortTypes
- Service implementations can be customized during assembly



bea
Think liquid.

Service Assembly Model

- Assembly provides
  - A unified, language-independent way to expose implementations as services
    - Java, BPEL, other languages (including .NET)
  - Technology independent modelling and composition of service networks
    - Service dependencies
    - Resolution through wiring
  - Facilities for dynamic service configuration
    - Properties
    - Protocols
    - Qualities of service

Relationship to WCF/Indigo

"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."

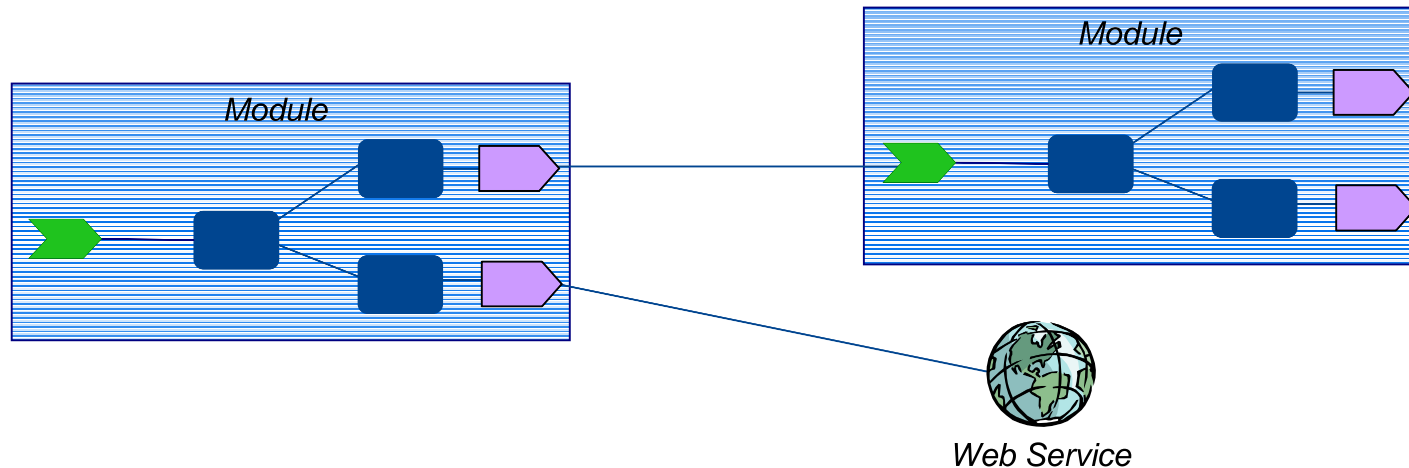
- The '.NET' David Chappell

- Source: http://www.davidchappell.com/HTML_email/Opinari_No15_12_05.html

bea
Think liquid.

Service Assembly Model

- Model for assembling tightly coupled code
- Model for assembling loosely coupled services



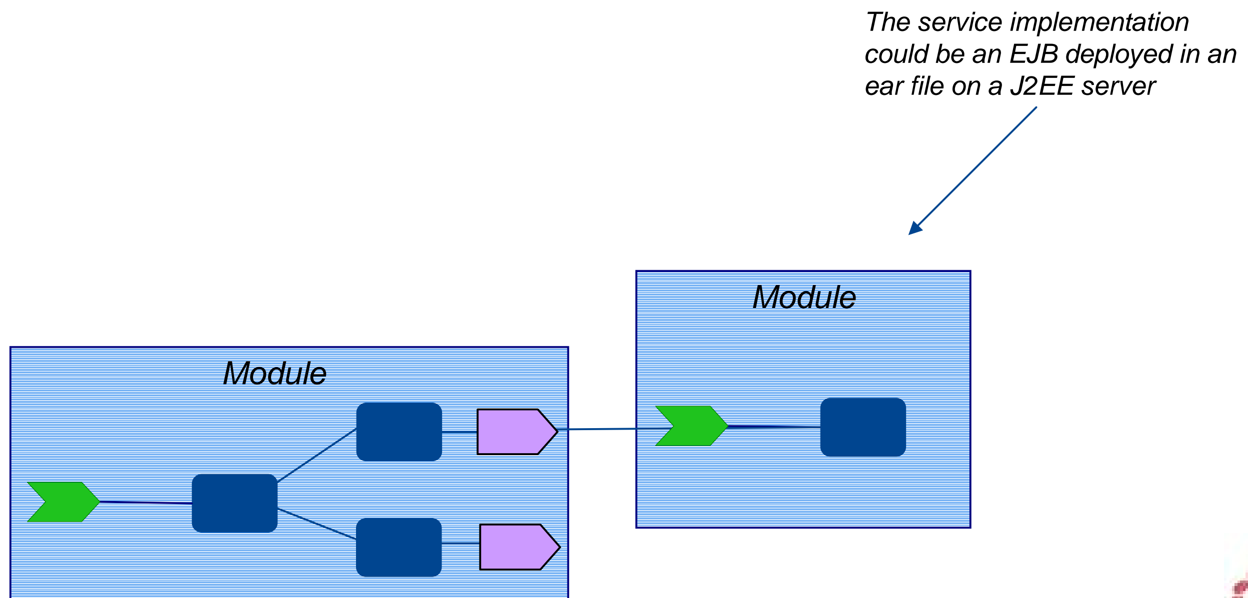
bea Think liquid.

SCA Client and Implementation Model

- Application code too often tied to specific technologies
  - e.g. JMS, JCA, JDBC, MQ, CICS ECI, Tuxedo
- SCA provides a simplified model for authoring and invoking services
  - Uses business level interfaces to access services
  - Removes protocol & middleware concerns from business logic
  - Promotes asynchrony and conversations as first-class elements of the programming model
    - Conversations expressed as part of the service business interface
  - Allows service implementations to access assembly capabilities (e.g. policy)
  - Supports late binding
- Start with a Java specification with other languages to follow

Reuse of existing artifacts

- SCA will also define first-class integration with existing programming models
  - Ability to re-use existing assets
  - Java
    - Java EE technologies such as EJB 2, EJB 3
    - Spring
- Non-Java
  - BPEL
  - COBOL
  - BAPI



bea Think liquid.

Agenda

- SCA Collaboration Overview
- SCA Technical Overview

bea Think liquid.

Assembly Model Concepts

- Design Time Assembly Artifacts
  - Module
  - Implementation
  - Component
  - External Service
  - Entry Point
  - Wire
- Deployment Time Assembly Artifacts
  - Subsystem
  - Module Component
  - External Service
  - Entry Point
  - Wire

Module

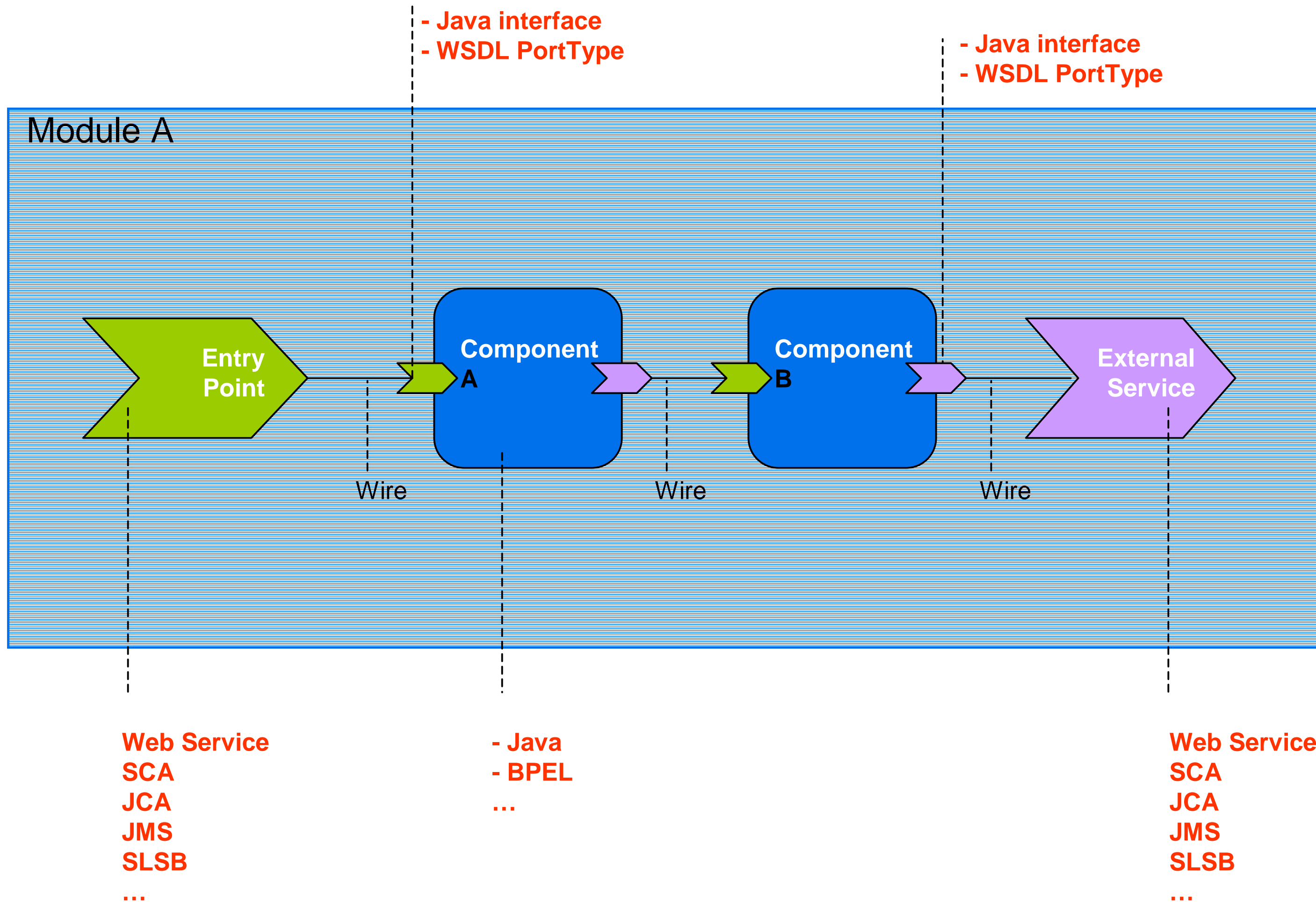
- Is the largest composition of tightly coupled components that are developed and deployed together

- A module contains
  - Business logic organized as ***Components***
  - Remote services termed ***External Service References***
  - Public services termed ***Entry Points***
  - ***Wires*** connect Components, External References, and Entry Points
  - Artifacts defined in ***sca.module*** and 0..n ***<fragmentname>.fragment*** XML files

- It is a basic unit of loosely-coupled composition within an SCA System

bea Think liquid.

Module



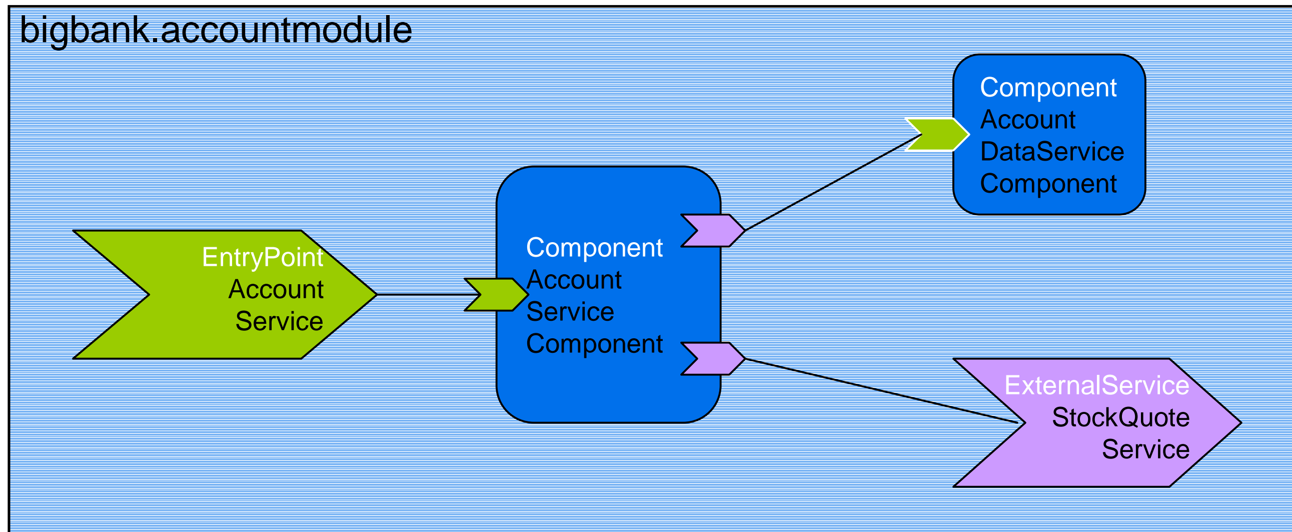
bea Think liquid.

Module

- Visibility
  - A module defines the boundary for component visibility
  - Components are not directly referenced by external clients
  - Components do not directly reference module external services
- Locality
  - Components within a module run within a single address space
  - *Local Services* may be defined that
    - May only be access from within a module
    - Depend on by-reference semantics
- Deployment
  - A module defines a unit of deployment for business logic

bea Think liquid.

Module Representation



sca.module

```
<module   xmlns="http://www.osoa.org/xmlns/sca/0.9"
          xmlns:v=http://www.osoa.org/xmlns/sca/values/0.9 name=”my
  module" >

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

          <v:stockQuoteService>StockQuoteService</v:stockQuoteServi
  >
     </references>
  </component>
```

Component Implementation

- Provides the implementation of a component
- May be one of several different *implementation technologies*
  - e.g. Java, BPEL, C++, …
  - implementation type *extensibility*
- Provides the business function of one or more *services*
- Uses other services through service *references*
- Defines configurable *properties*
- Service and references are typed by *interfaces*
  - e.g. Java interfaces, WSDL portTypes
  - interface type *extensibility*
  - can be *remotable* or *local-only*
  - can be *bi-directional*, companion callback interface
  - can be *scoped*
    - stateless, request, session, module, … *(extensible)*
- Services, references and properties define the *configurable aspects of an implementation*. SCA refers to them as the

Local & Remotable Services

- Local Service Characteristics
  - typed with local interface, e.g. Java interface with no @Remotable annotation
  - not addressable outside the boundaries of a module
  - fine-grained, tightly coupled interfaces
  - parameters & return values are by-reference

- Remotable Service Characteristics
  - typed with remotable interface, e.g. Java interface that has an @Remotable annotation, or WSDL portType
  - may be addressed outside the scope of a module if configured
  - coarse-grained, loosely coupled
  - parameters & return values are by-value

bea Think liquid.

Implementation - Java

```java
@Remotable
public interface AccountService{
        public AccountReport getAccountReport(String customerID);
}


public class AccountServiceImpl implements AccountService {
  @Property
  private String currency = "USD";
  @Reference
  private AccountDataService accountDataService;
  @Reference
  private StockQuoteService stockQuoteService;
  public AccountReport getAccountReport(String customerID) {
        // implementation
  }
}
```


bea Think liquid.

Component Type

Component Type generated from an implementation

```
<?xml version="1.0" encoding="ASCII"?>
<componentType xmlns="http://www.osoa.org/xmlns/sca/0.9"
              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>
```


bea Think liquid.

Component

- *Configured* instance of an implementation
  - there can be more then one component using the same implementation
- *Provides* and *consumes services*
- Sets *properties*; overridable (no, may, must)
- Sets service *references* by *wiring* them to services
  - wiring to services provided by other components or by external services

```
<component name="AccountServiceComponent">
   <implementation.java
class="services.account.AccountServiceImpl"/>
   <properties>
      <v:currency override="may">EURO</v:currency>
   </properties>
   <references>
      <v:stockQuoteService>StockQuoteService</v:stockQ
```


bea Think liquid.

External Service

- Represent *remote services* that are external to the module
  - accessed by clients within a module like any other component service
  - valid reference values
- Use *bindings* to describe the access mechanism to the external service
  - e.g. Web service, stateless session EJB, JMS, JCA, …
  - binding type *extensibility*
  - overridable (no, may, must)

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


bea Think liquid.

Entry Point

- Used to *publish remotable services* provided by a module, for use by module external clients
  - can be service provided by component or external service
- Use *bindings* to describe the access mechanism and endpoint address
  - e.g. Web service, stateless session EJB, JMS, JCA, …
  - binding type extensibility
  - always overridable

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


bea
Think liquid.

Wire

- Resolves a service reference
- Valid wire sources are
  - service references
  - entry points
- Valid wire targets are
  - services
  - external services
- Source and target have to be in the same module
- Reference and service interface have to be compatible
- Typically represented by *references* elements

Conversational Services

- SCA allows *Session-Scoped* services to provide *conversational services*

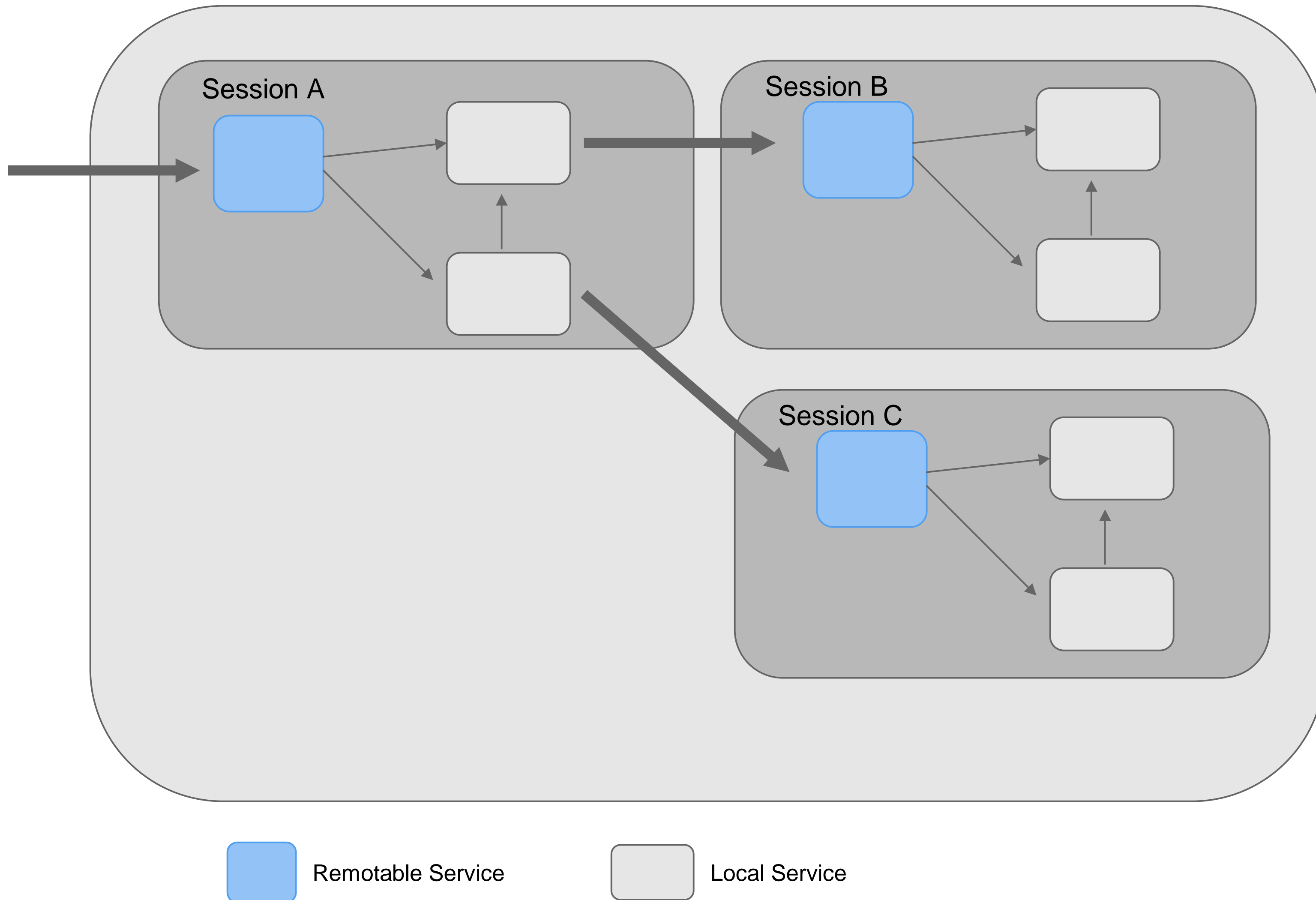
```
package com.bigbank;

@Scope("Session")
@Remotable
public interface LoanService {
  public void apply(LoanApplication application);
  public void lockCurrentRate(int termInYears);
  public void cancelApplication();
  public String getLoanStatus();
}
```

Session Scoped Service Use

```
package com.independent;

import com.bigbank.LoanService;

@Remotable

public class BrokerImpl implements MortgageBroker {


  @Reference

  LoanService loanService;  // LoanService is Session Scoped


  public void applyCustomer customer, HouseInfo houseInfo, int term) {

     LoanApplication loanApp;

     loanApp = createApplication(customer, houseInfo);

     loanService.apply(loanApp);

     loanService.lockCurrentRate(term);

  }
```


bea Think liquid.

Session Scoped Services



bea
Think liquid.

Conversation Lifetime

Starting conversations

- A @reference to a conversational service is injected.
- A call is made to ModuleContext.newSession.

Continuing conversations

- **Holding** the service reference that was created when the conversation started.
- **Getting** the service reference object passed as a parameter from another service
- **Loading** a service reference that had been written to some form of persistent storage.
- **Looking up** the service reference based on the session ID:

  ModuleContext.lookupServiceReference(serviceName, id)

Ending conversations

- A server operation that has been annotated @EndSession has been called

bea Think liquid.

Callbacks for Bidirectional Services

```
@Remotable
@Callback(MyServiceCallback.class)
public interface MyService {
    public void someMethod(String arg);
}


@Remotable
public interface MyServiceCallback {
    public void receiveResult(String result);
}
```

Using Callbacks from Service Implementations

```
public class MyServiceImpl implements MyService,
   MyServiceCallback {
    @Callback
    private MyServiceCallback callback;

    @Reference
    private MyService backendService;

    public void someMethod(String arg) {
        backendService.someMethod(arg);
    }

    public void receiveResult(String result) {
        callback.receiveResult(result);
    }
```


bea Think liquid.

Assembly Model Concepts

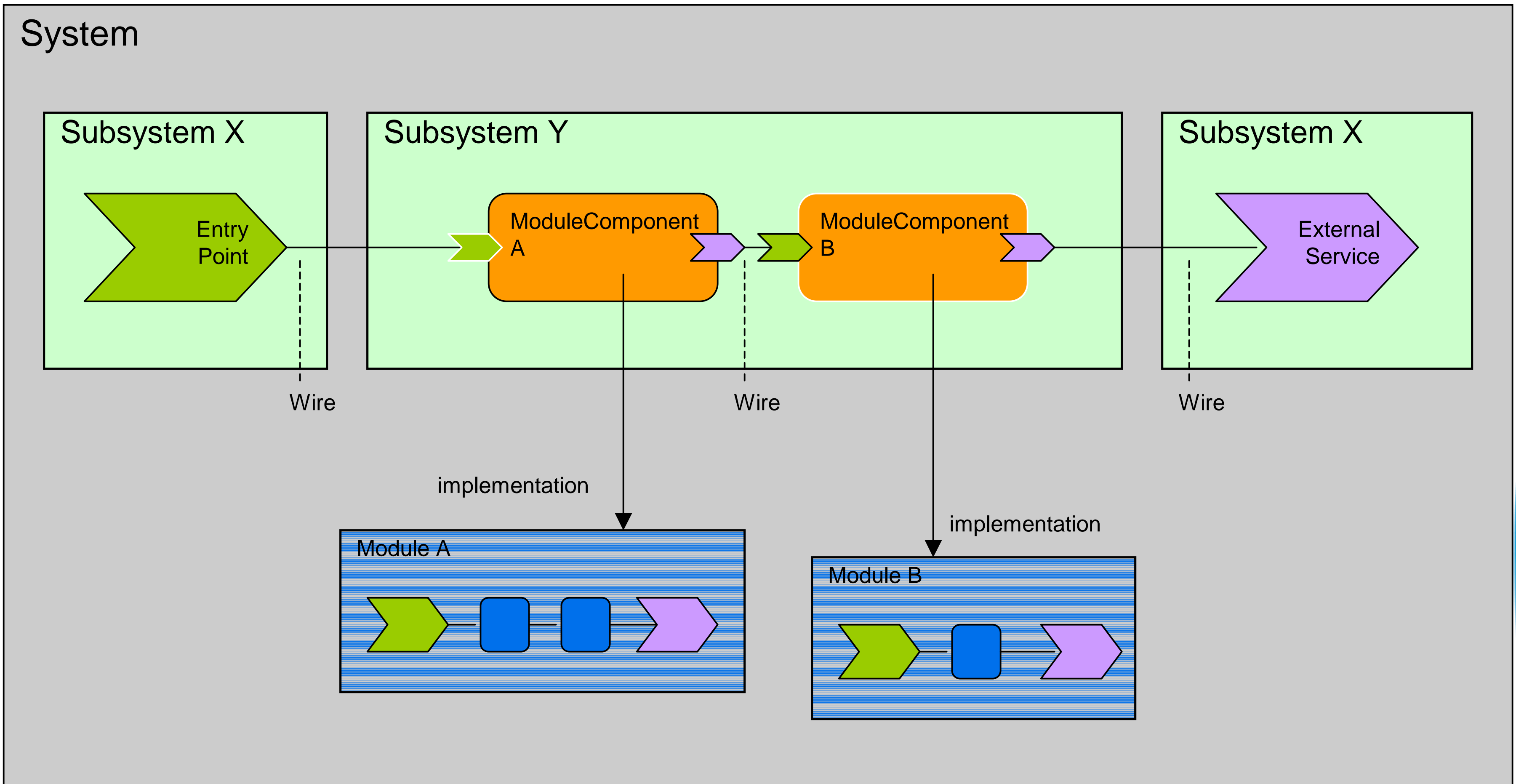
- Design Time Assembly Artifacts
  - Module
  - Implementation
  - Component
  - External Service
  - Entry Point
  - Wire
- Deployment Time Assembly Artifacts
  - Subsystem
  - Module Component
  - External Service
  - Entry Point
  - Wire

Subsystem

- Modules get deployed and configured into an *SCA system*
- An SCA system contains *module components*, *external services*, *entry points*, *wires*
- They get configured into the system using *sca.subsystem file*
- An SCA system is defined by all its subsystems
- Subsystems make deployment simpler
  - individual subsystems can be created and deployed independently
  - can contain only wires, or module components, or external services, or entry points
  - can control naming of elements in the system
    - e.g. no requirement for globally unique names

bea
Think liquid.

Subsystem



bea Think liquid.

Module Component

- *Configured* instance of a module
  - there can be more then one module component using the same module
- Modules are the *implementations* of module components
  - entry points of the module define their *services*
  - external services (with overridable may or must) of the module define their *references*
  - properties of components contained in the module that are marked as overridable define their settable *properties*
- *Recursive model*
  - system level components (i.e. module components) are implemented by modules
  - components within modules are implemented by standard or custom implementation technologies
- Sets *properties*
- Sets service *references* by *wiring* them to services
  - wiring to services provided by other module components or by external services

bea Think liquid.

Subsystem and Module Component

```
<?xml version="1.0" encoding="ASCII"?>

<subsystem xmlns=http://www.osoa.org/xmlns/sca/0.9 name="bigbank.accountsubsytem">

  <moduleComponent name="AcountModuleComponent" module="bigbank.accountmodule"/>
        <properties>
            <v:currency>US</v:currency>
        </properties>
        <references>
            <v:StockQuoteService>
                   http://www.betterstockquote.com/services/StockQuoteService
            </v:StockQuoteService>
        </references>
  </moduleComponent>

</subsystem>
```


bea Think liquid.