


Composing Business Solutions using SCA

Dr Mike Edwards
IBM Hursley
mike_edwards@uk.ibm.com



All statements regarding IBM's future plans, direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Such statements do not represent a commitment of future availability, content, performance or function of any products or features.

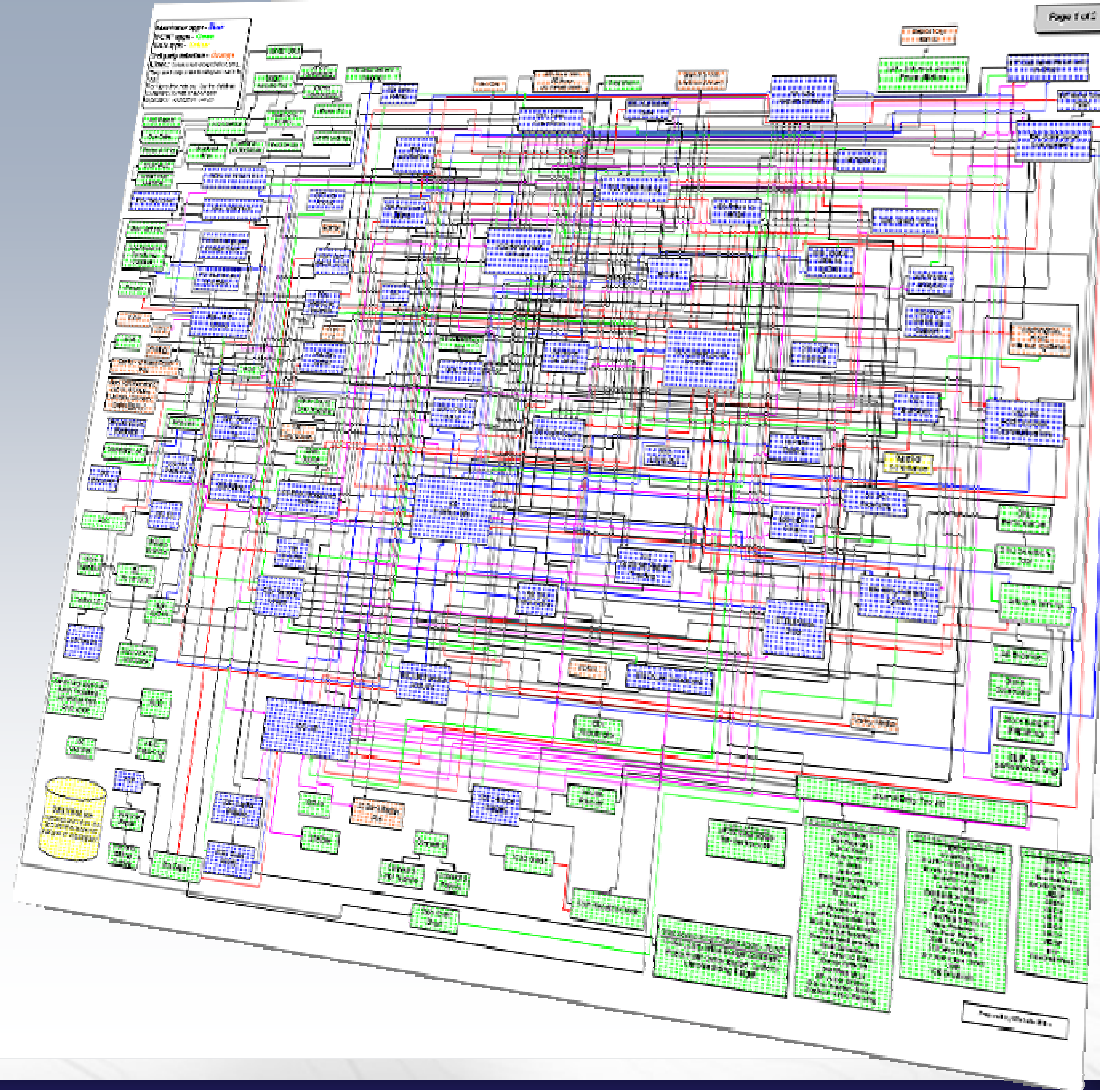
Service Component Architecture (SCA):

A model for the creation of business systems using Service-Oriented Architecture by the composition and deployment of new and existing service components

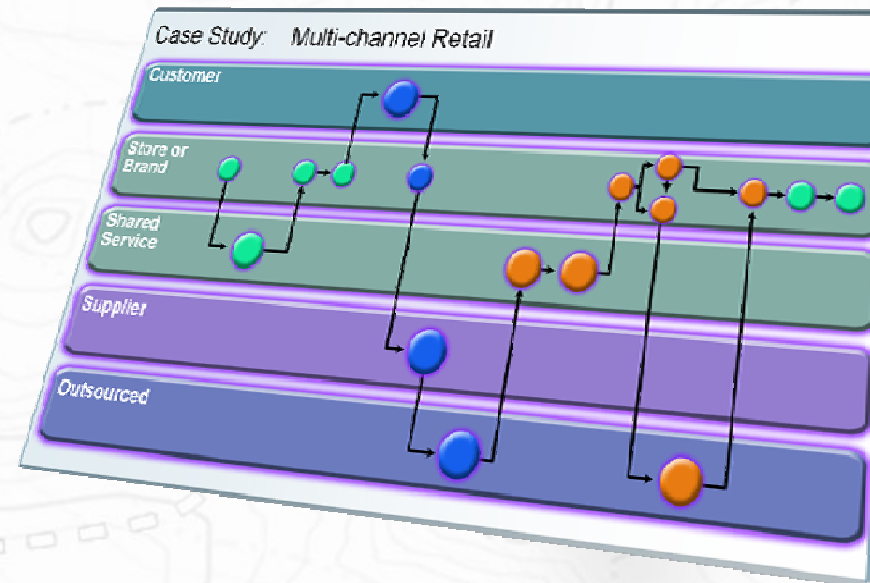
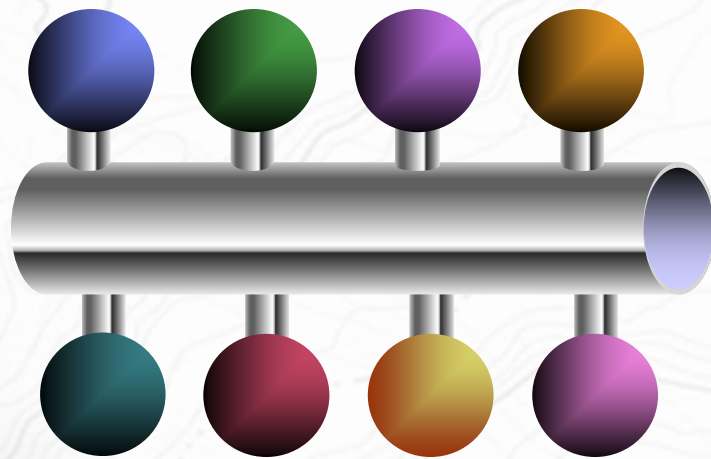
- **SCA: Why**
- **SCA: Scenarios**
- **SCA: Details**
- **SCA: Specifications, Standardization and Industry Support**

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

SOA Programming Model (1)

- 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

SOA Programming Model (2)

- 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 are SCA and SDO?

- **Service Component Architecture**
 - an executable model for building service-oriented applications as composed networks of service components
 - “how to build composite service applications”
- **Service Data Objects**
 - a unified model for the handling of (service) data irrespective of its source or target
 - “how to handle data in a services environment”

Service Component Architecture (SCA): Simplified Programming Model for SOA

- **executable** model for:
- **building** service components
- **assembling** components into applications
- **deploying** to (distributed) runtime environments
 - Service components built from **new or existing code using SOA principles**
 - **vendor-neutral** – supported across the industry
 - **language-neutral** – components written using any language
 - **technology-neutral** – use any communication protocols and infrastructure to link components

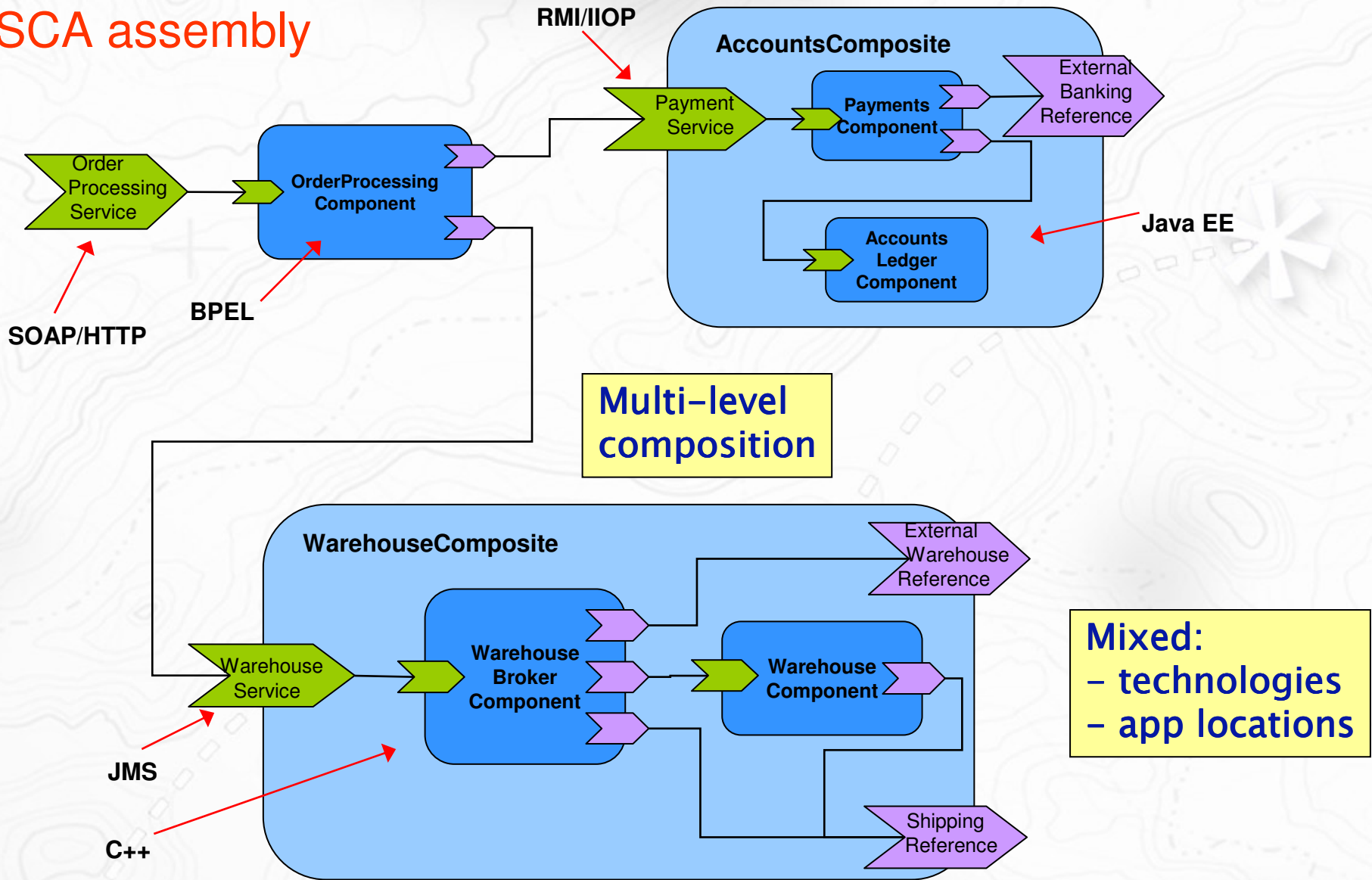
SCA: What is it NOT

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

Key benefits of SCA

- **Loose Coupling**: components integrate without need to know how others are implemented
- **Flexibility**: components can easily be replaced by other components
- **Services** can be *easily* invoked either synchronously or asynchronously
- **Composition** of solutions: clearly described
- **Productivity**: easier to integrate components to form composite application
- **Heterogeneity**: multiple implementation languages, communication mechanisms
- **Declarative** application of infrastructure services
- **Simplification** for **all** developers, integrators and application deployers

SCA assembly



Agenda

- **SCA scenarios**



Bottom-up Composition

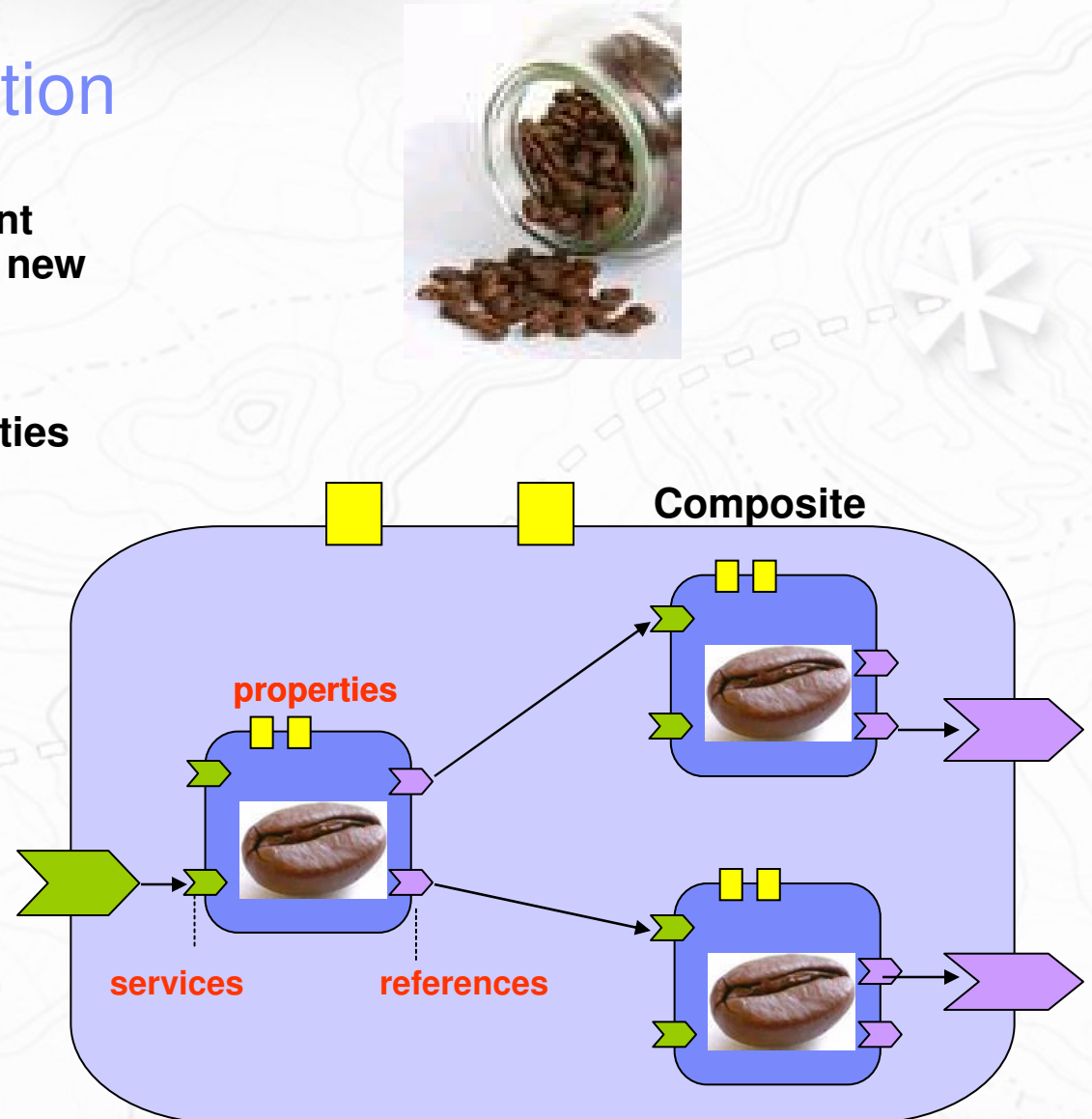
Select a set of existing component implementations for building the new composite

Configure the component properties

Draw internal wires

Wrap the components in a composite and configure external services/references

Hand off the composite to Deployer



Top-down Composition

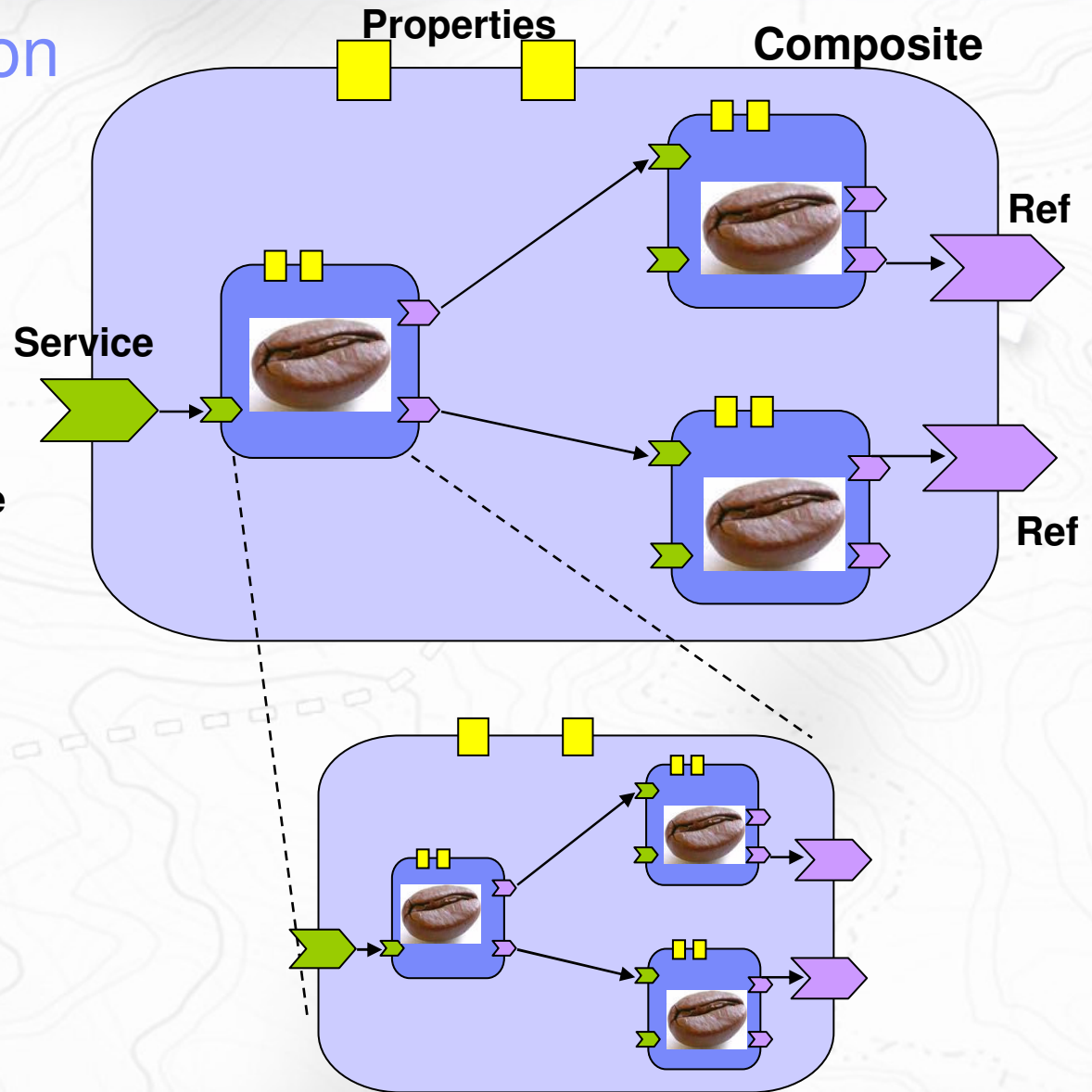
Start with gathering requirements for the top-level composite

Define the services/references and properties for the composite

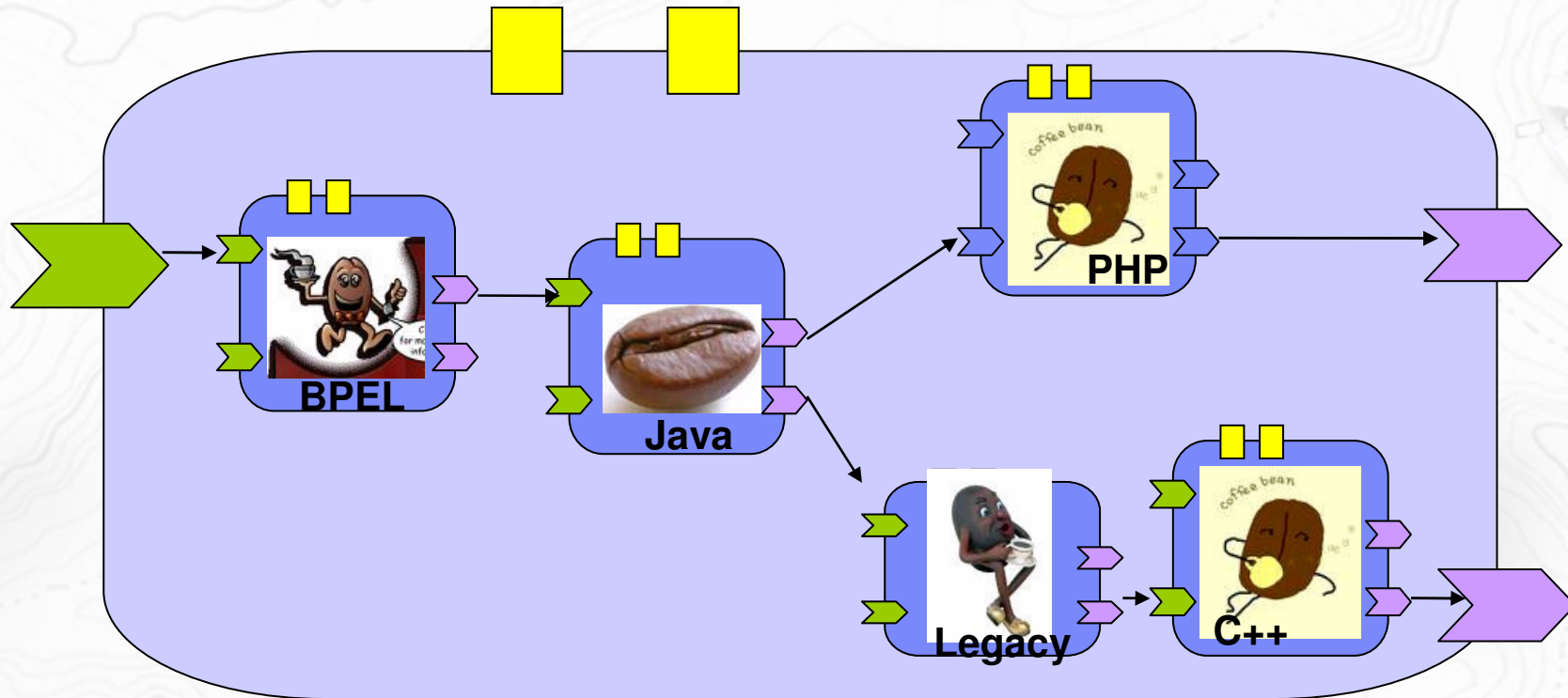
Break down the composite into individual components and wires between them

Recursively break down each component

Hand off the individual component contracts to developers for implementation



Heterogeneous Assembly



Components in the same composite share a common context for many aspects such as installation, deployment, security settings, logging behavior, etc.

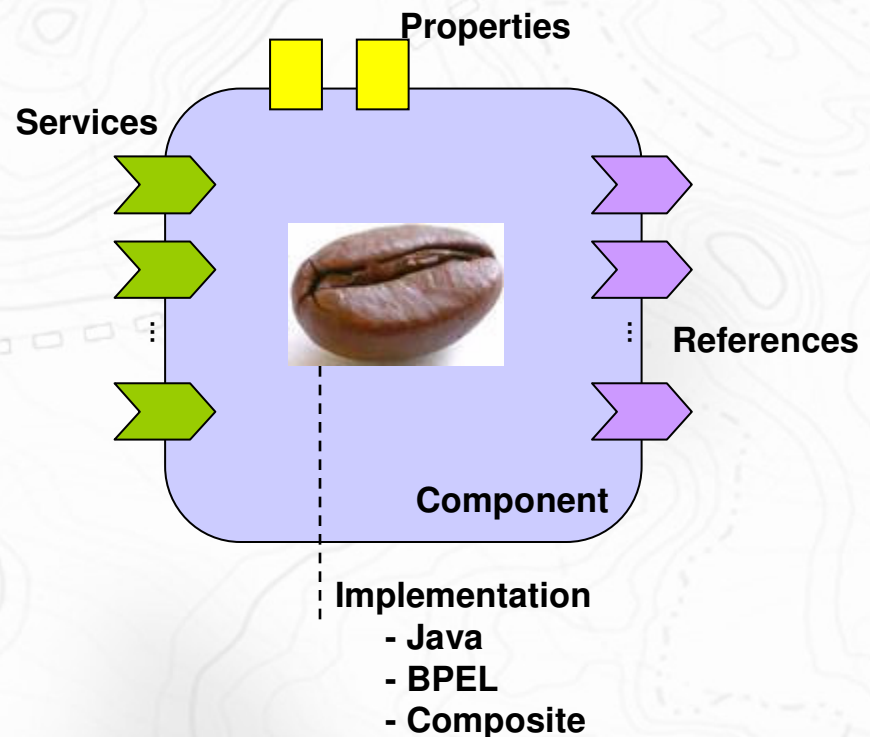
Implementation Reuse – By Configuration

Select an existing component implementation

Configure its behavior (via setting props, refs) to match the current requirements

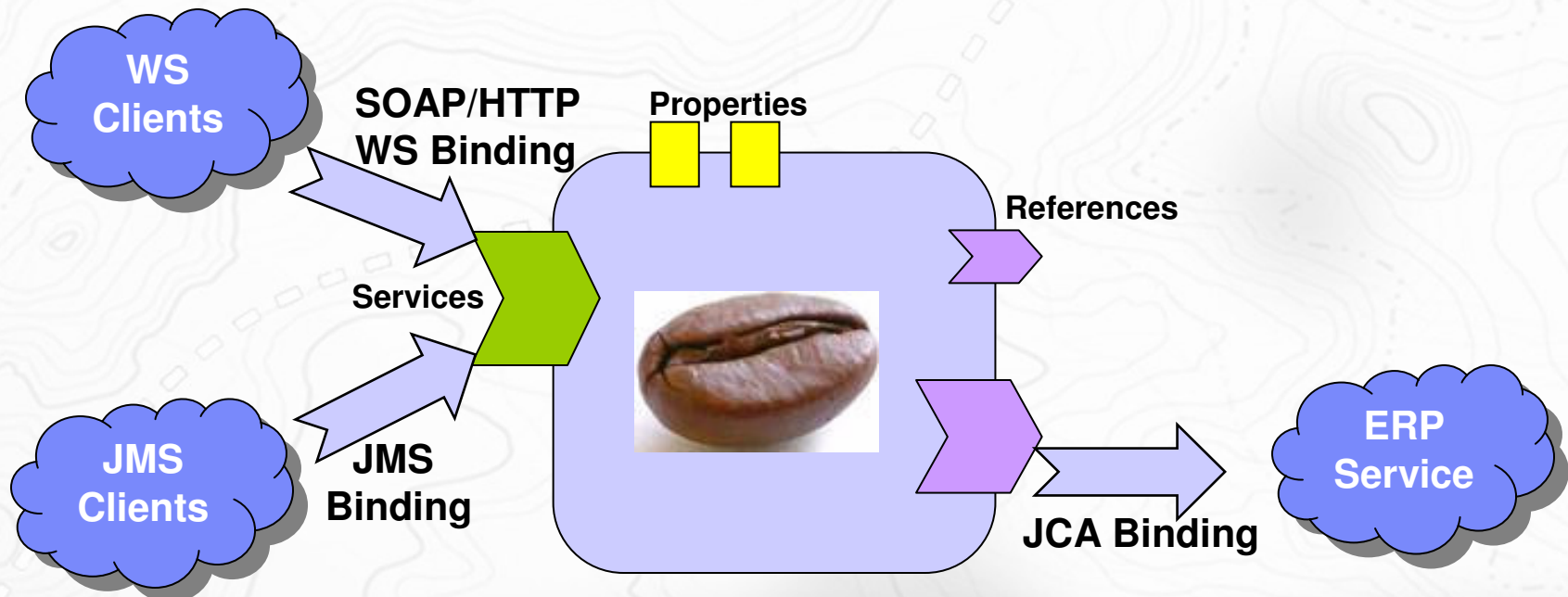
E.g. Configure multiple instances of product pricing component, each with different currency, tax rate, discount charts, etc.

Deploy the component implementation
- Multiple instances of the same implementation may be running simultaneously

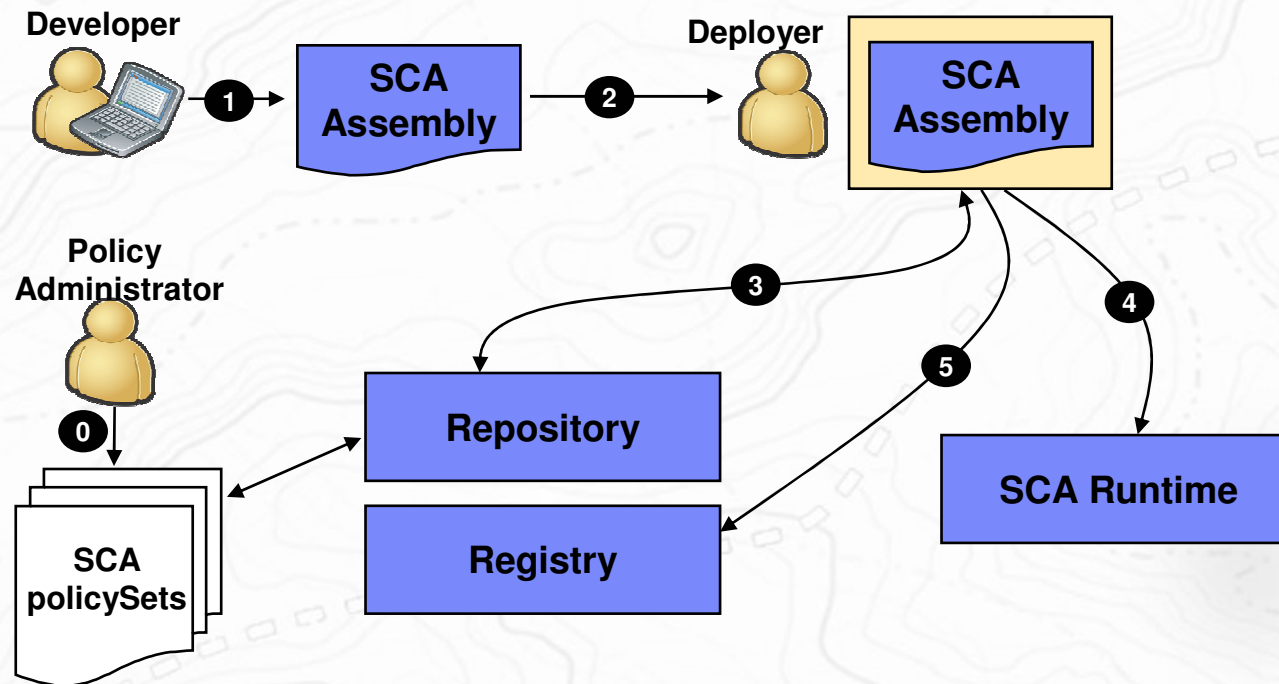


Deployment Flexibility

Deployer chooses and configures communication mechanisms for services/references without having to modify the component implementation



Abstract policy declaration



0. Policy Administrator authors SCA policySets with concrete policies
1. Developer specifies intents on SCA assembly
2. Developer hands over SCA assembly to Deployer
3. Deployer configures SCA assembly by assigning SCA policySets (could be automated)
4. Deployer deploys configured SCA Assembly to SCA Runtime
5. Deployer updates Registry

Agenda

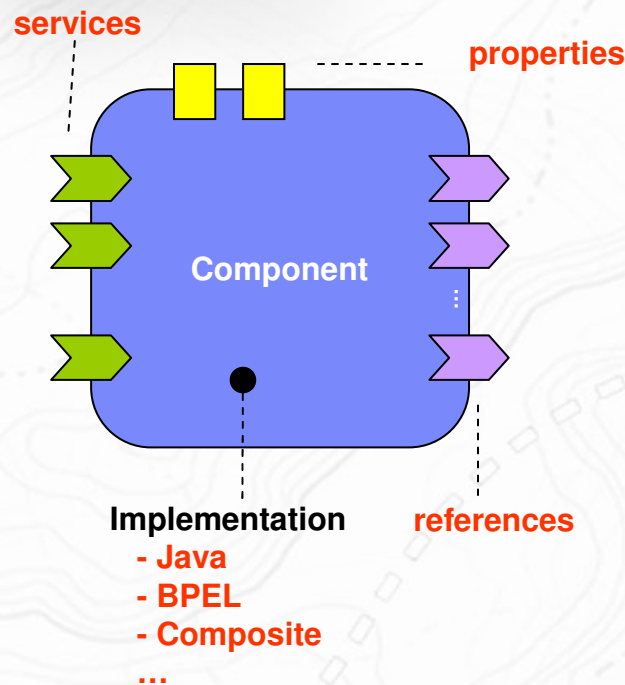
- **SCA details**



SCA Elements

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

Component

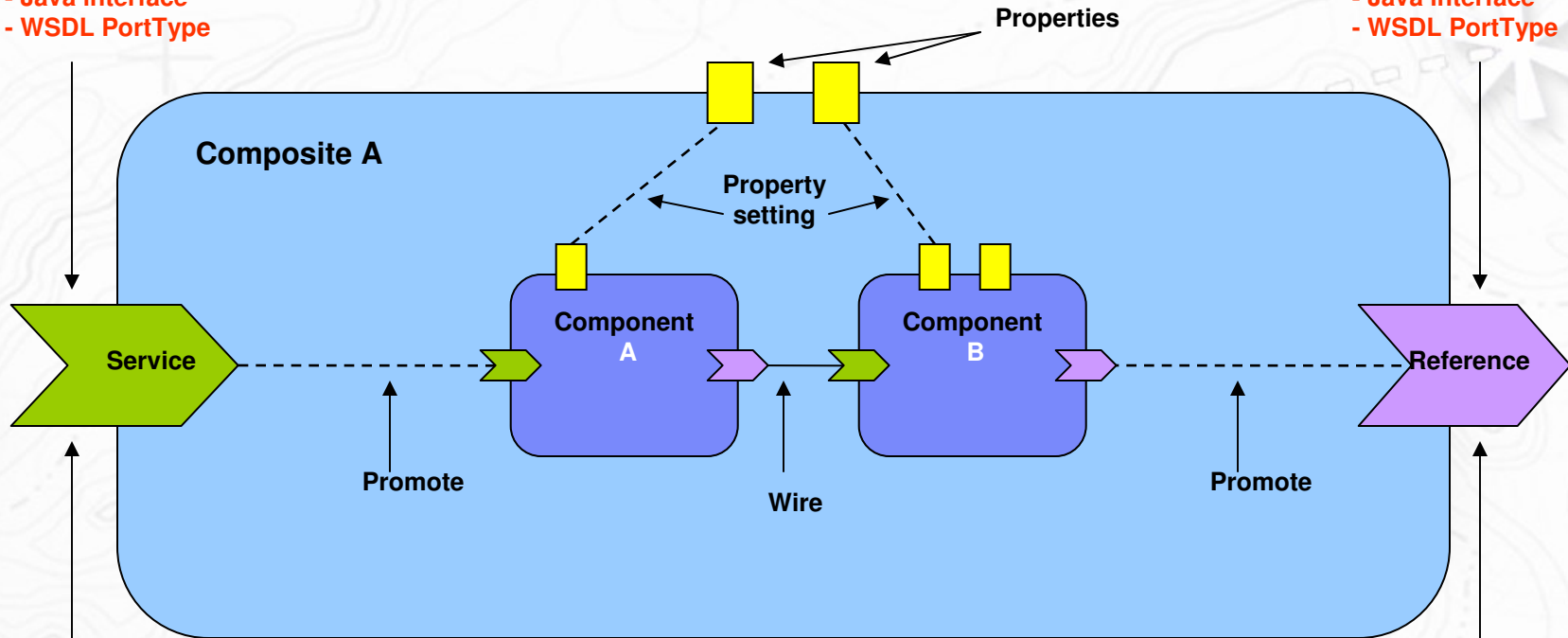


- **configured** instance of **implementation**
 - more than one component can use same implementation
- **provides** and **consumes services**
- Sets **values** for implementation **properties**
- Sets service **references** by **wiring** them to services

Composite

Service interface
- Java interface
- WSDL PortType

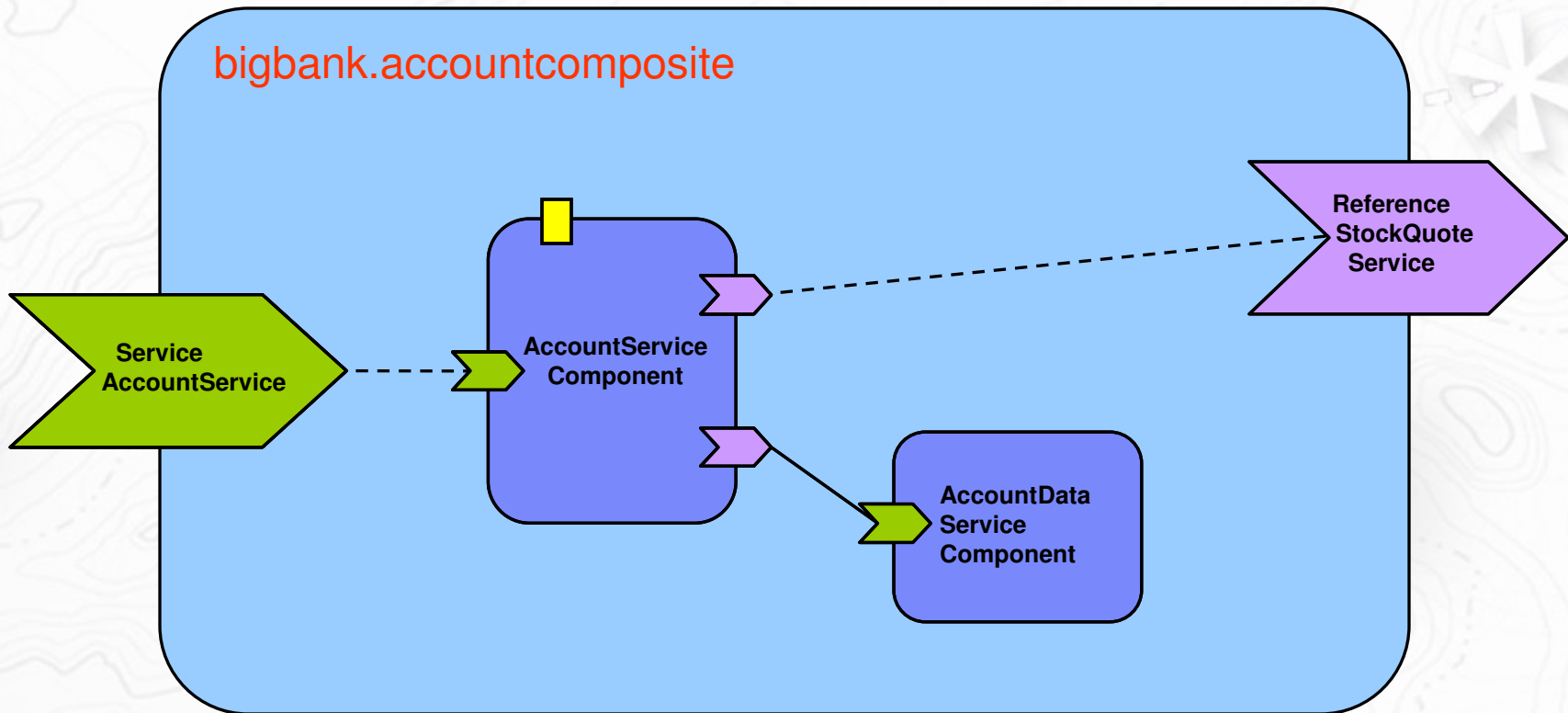
Reference interface
- Java interface
- WSDL PortType



Binding
Web Service
SCA
JCA
JMS
SLSB
...

Binding
Web Service
SCA
JCA
JMS
SLSB
...

Simple Example




```

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

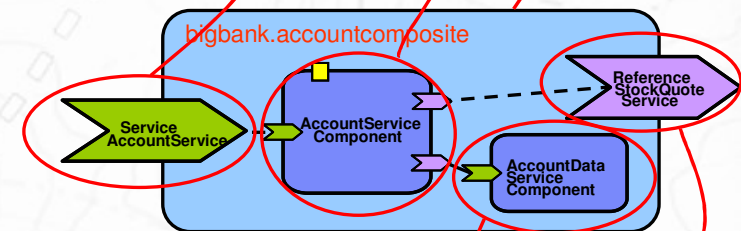
  <service name="AccountService" promote="AccountServiceComponent">
    <interface.java interface="services.account.AccountService"/>
    <binding.ws port="http://www.example.org/AccountService#
      wsdl.endpoint(AccountService/AccountServiceSOAP)"/>
  </service>

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

```



Java Implementation Example: Service Interface

```
package org.example.services.account;  
  
@Remotable  
public interface AccountService {  
    public AccountReport getAccountReport (String customerID);  
}
```

Interface is callable
remotely
eg. as a Web service

Java Implementation Example – Implementation (part 1)

```
package org.example.services.account;

import org.osoa.sca.annotations.*;

@Service(interfaces = AccountService.class)
public class AccountServiceImpl implements AccountService {

    private String currency = "USD";
    private AccountDataService accountDataService;
    private StockQuoteService stockQuoteService;

    public AccountServiceImpl(
        @Property("currency") String currency,
        @Reference("accountDataService") AccountDataService dataService
        @Reference("stockQuoteService") StockQuoteService stockService) {
        this.currency = currency;
        this.accountDataService = dataService;
        this.stockQuoteService = stockService;
    }
}
```

Annotation for the service offered by this class

Constructor with annotations for injected property and references

Java Implementation Example – Implementation (part 2)

```
public AccountReport getAccountReport(int customerID)
    throws AccountDataUnavailableException {

    AccountReport accountReport =
        accountDataService.getAccountReport(customerID);
    List<Stock> stocks = accountReport.getStocks();

    List<StockValues> stockValues =
        stockQuoteService.getValues(stocks, currency);

    accountReport.setStockValues(stockValues);

    return accountReport;
}
} // end class
```

Get the basic account report using the account data service

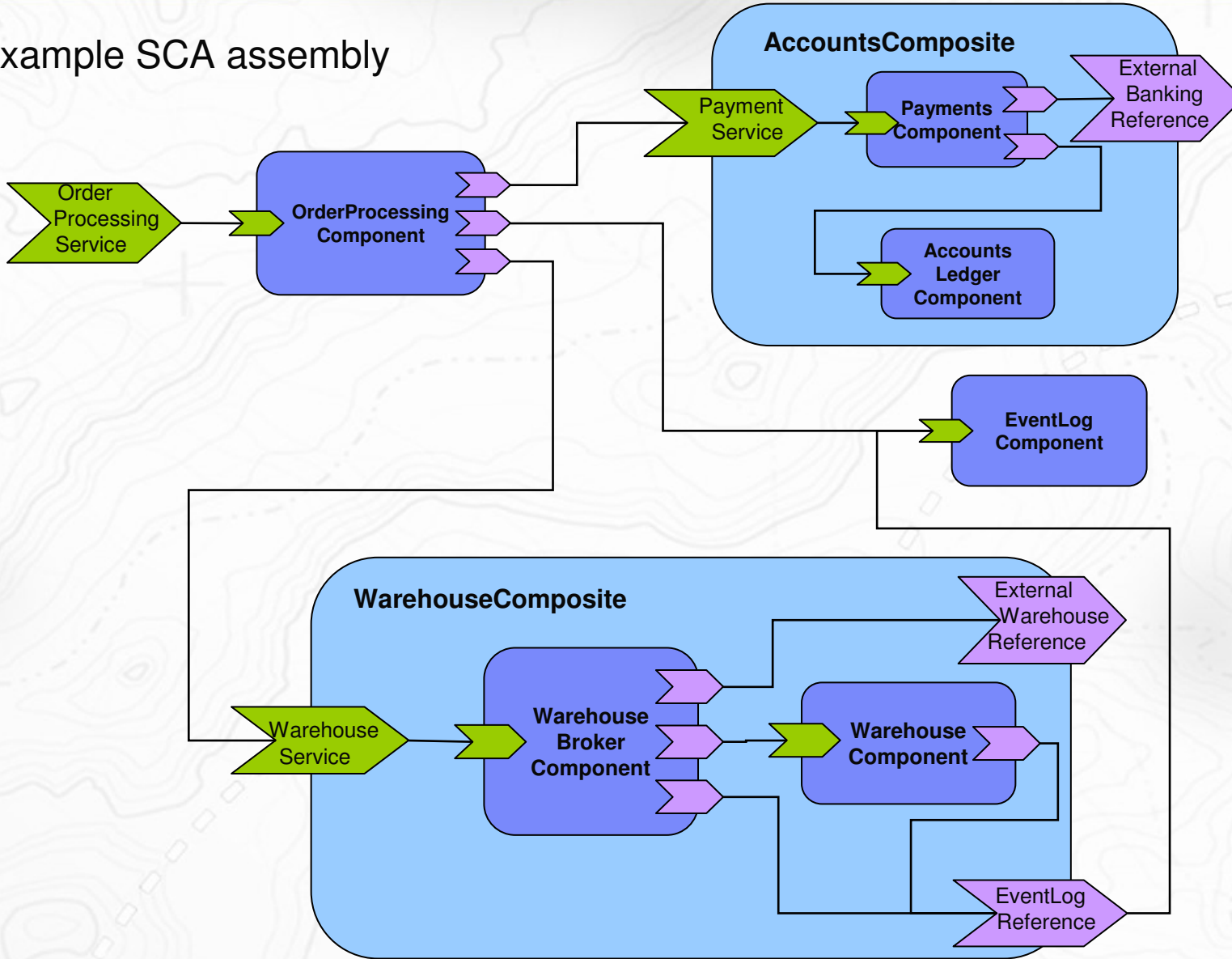
Obtain up to date stock values using the stock quote service

Update the account report with the latest stock values

SCA (Java) Implementation principles

- Code only to business interfaces
 - “Don’t program **to SCA**, just program...”
 - Use Java idioms
 - Minimal middleware APIs used only in special cases
- Components declare both the **services** they offer and **references** to other services they need
- Injection of required service References and Property values
 - via constructor / via setter methods / via direct field injection
- Java annotations for SCA elements
 - services, references, properties
 - + more advanced features such as intents, bindings
- Principles apply to other languages

Example SCA assembly



Significant features of SCA Composites

- **Distributed**
 - each component can exist on a different system/process in a network
- **Heterogeneous**
 - assemblies can contain components of mixed implementation types
- **Recursive or Nested**
 - component in a composite can itself be implemented as a composite
- **Model existing applications/systems**
 - either as components or as composites
- **Declarative application of infrastructure services**
- **“keep APIs out of the business logic”**
 - philosophy for component implementation

SCA Implementation Types: Client & Implementation Specifications

- Specify how service components and service clients are built
- Specific to particular language, framework or language/framework-specific APIs
- Extensible – more languages/frameworks can be added
- Currently defined C&I specifications:
 - BPEL
 - Java POJO
 - Spring Framework
 - C++
 - EJB (in preparation)

Example Implementation Types

```
<component name="ComponentA">  
  <implementation.bpel process="foo:Process/Example/processA"/>  
  ...  
</component>
```

← BPEL

```
<component name="ComponentB">  
  <implementation.java name="com.foo.ImplementationB"/>  
  ...  
</component>
```

← Java POJO

```
<component name="ComponentC">  
  <implementation.spring location="SpringApplicationC.jar"/>  
  ...  
</component>
```

← Spring

```
<component name="ComponentD">  
  <implementation.cpp library="libraryD" header="ServiceDImpl.h"/>  
  ...  
</component>
```

← C++

SCA Interaction Model

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

Bidirectional Interfaces (Callbacks)

- Used for asynchronous messaging
- Unifies the provider (service) interface with callback interface
- 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

- Model stateful service interactions
- Frees application programmer from conversation/correlation management
- Imposes requirements on bindings
- 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>
```

Agenda

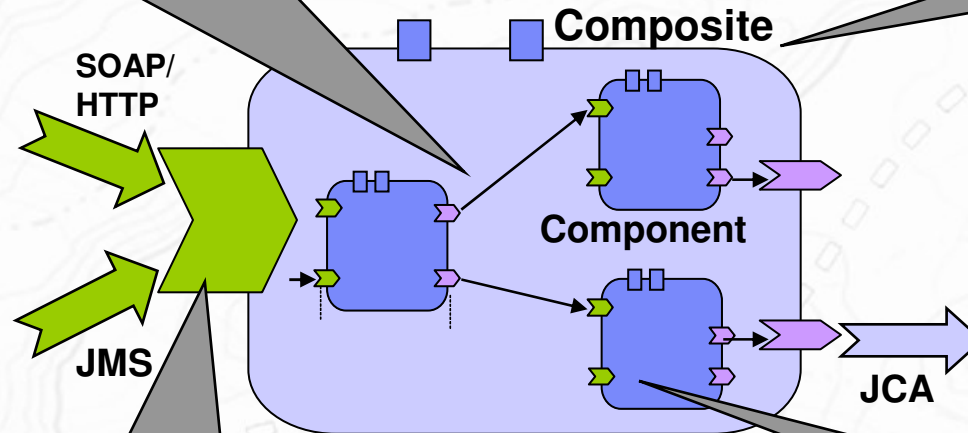
- ***SCA Specifications, Standardization and Industry Support***



SCA Technology

How do I define, use and administer policies for non-functional aspects (QoS, etc)?
 → SCA Policy Framework Spec

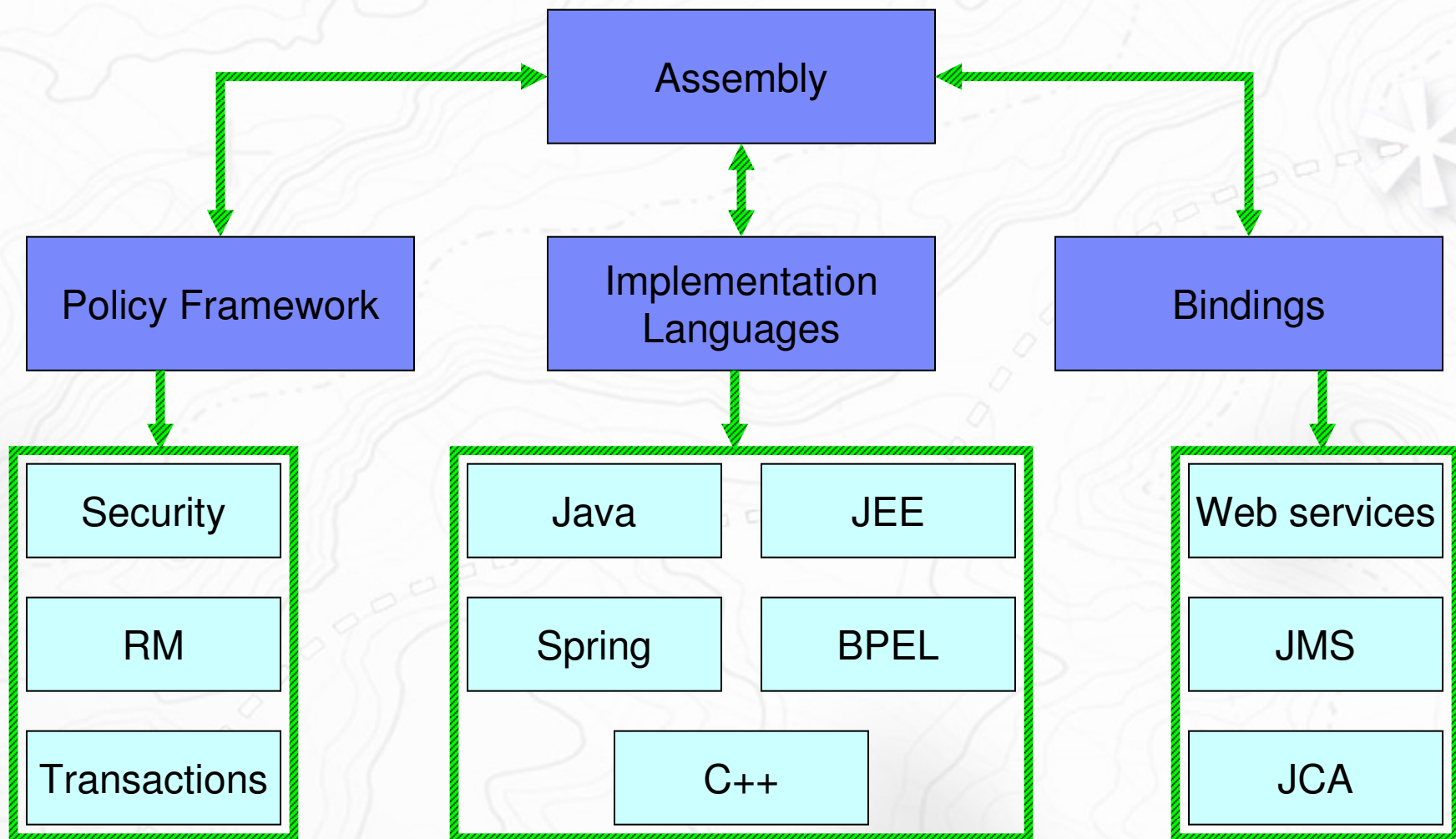
How do I define, configure and assemble components to create composites?
 → SCA Assembly Spec



How do I configure SCA services/references to use SOAP/HTTP or JMS or JCA, ...
 → SCA WS Binding Spec, ...

How do I develop SCA components in BPEL? Or in Java? Or C++, PHP, ...
 → SCA BPEL Client & Impl Spec, ...

The SCA Specifications



4Q04

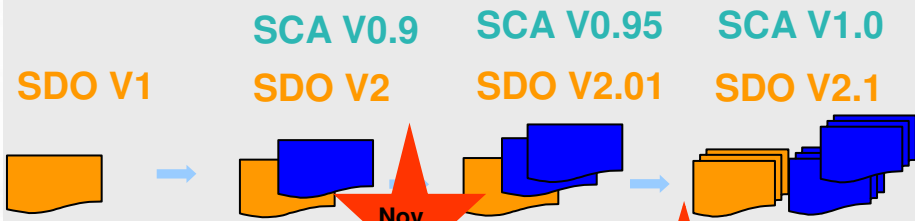
3Q05

2Q06

1Q07

2Q07

2007 +



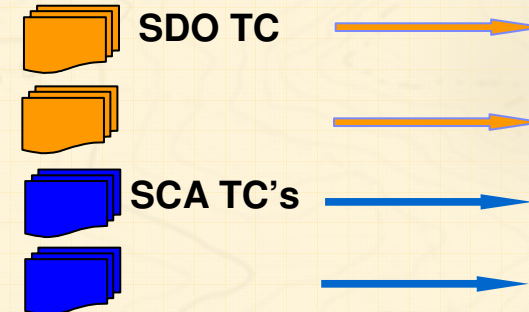
Finalization of further SCA Specs

Further complementary incubation

Nov 2005
 Press Announcement of Project Launch

July 2006
 New Partners Announced

March 2007
 Specs 1.0 Submission for Standardization



Early Adopters

ISVs

System Vendors

Customer Value

Adoption

Time Line Summary

Open Source Projects and Implementations

- **Apache Tuscany Incubator Project**
 - Provides SOA programming runtime based on SCA, SDO
 - Java™ & C++ implementations today
 - Aim to support several runtimes (eg Tomcat) and protocols
 - Associated PHP implementation on PECL site
 - <http://incubator.apache.org/tuscany>

- **Eclipse SOA Tools Project**
 - Eclipse-based tooling for SOA applications and systems
 - Based on SCA as model for solutions built using SOA
 - Target range of systems including SCA runtimes such as Tuscany
 - <http://www.eclipse.org/stp/>

- **Several vendor implementations**
 - IBM WebSphere, Oracle Fabric, BEA, RogueWave, TIBCO

Summary

- SCA is an agile approach to developing systems using a service-oriented architecture
 - wide industry support
 - standardization taking place at OASIS
- SCA is being implemented in Open Source at Apache and at Eclipse
- SCA is implemented in WebSphere
 - WebSphere Application Server 6.1 SOA Feature Pack
 - WebSphere Process Server, WebSphere ESB

Useful links...

- OASIS Open CSA
<http://www.oasis-openca.org/>
- OASIS SCA Technical Committees
<http://www.oasis-openca.org/committees>
- Open SOA Collaboration
<http://osoa.org/display/Main/Home>
- V1 level of SCA specifications
<http://osoa.org/display/Main/Service+Component+Architecture+Specifications>
- Useful papers and interesting SCA information:
<http://osoa.org/display/Main/SCA+Resources>
- OASIS Webinar downloads:
<http://www.oasis-openca.org/resources>

A light purple background featuring a topographic map with contour lines, a dashed line, and a star symbol.

Questions and Answers

© IBM Corporation 2008. All Rights Reserved.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries. For a complete list of IBM trademarks, see www.ibm.com/legal/copytrade.shtml

AIX, CICS, CICSplex, DB2, DB2 Universal Database, i5/OS, IBM, the IBM logo, IMS, iSeries, Lotus, OMEGAMON, OS/390, Parallel Sysplex, pureXML, Rational, RCAF, Redbooks, Sametime, System i, System i5, System z, Tivoli, WebSphere, and z/OS.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.