CASCON [2008]

Service Component Architecture – A Simpler Architecture for SOAs

Doug Tidwell, IBM
dtidwell@us.ibm.com

Today's quote

- *Anyone who's interested in the future of application development should also be interested in SCA.*
  - David Chappell, *Introducing SCA* **davidchappell.com/articles/Introducing_SCA.pdf**

Agenda

- A brief history of SOAs
- Why SCA makes life simpler
- SCA application diagrams
- Exercise 1: Adding SCA to an application
- Exercise 2: Using an RMI service
- Exercise 3: Using an SCA property
- Exercise 4: Using a Web service
- Resources

A brief history of SOAs

Using a component

- When dealing with a component (in an SOA or not), there are three important pieces of information:
  - The interface of the component
  - The implementation of the component
  - The access method to invoke the component
- We'll consider how we use this information to invoke components.

The bad old days

- Originally, most components were hardwired into an application:
  - The application knew the details of the component's interface at build time.
  - The application accessed the component's implementation at build time.
  - The application knew the details of the component's access method at build time.
- This worked (and still does), but the application is relatively brittle.
  - If the implementation or access method changes, we have to modify our code, rebuild it, retest it and redeploy it.

The early days of Web services

- SOAP introduced a way to invoke a remote service with an XML envelope.
- The SOAP infrastructure built the envelope and sent it to a particular URL; the SOAP service's host invoked a service and sent XML back to us.
  - The application knew the details of the component's interface at build time.
  - *The application did not access the component's implementation at build time; the component is invoked at run time by the SOAP infrastructure.*
  - The application knew the details of the component's access method at build time (usually SOAP/HTTP).

The early days of Web services

- Things have gotten more complicated since then:
  - Protocols other than HTTP
  - Document-style SOAP services instead of RPC
  - Asynchronous invocation with JMS
  - Encryption, conversations, reliable messaging, WS-*
  - Etc.

Better SOA applications with SCA

- An SCA application is even more dynamic:
  - The application knows the details of the component's interface at build time.
  - *The application does not access the component's implementation at build time; the component is invoked by the SCA invocation framework.*
  - *The application does not know the details of the component's access method at build time; this is also handled by the SCA invocation framework.*
- SCA moves the implementation and access method details out of your application and into the middleware.

Better SOA applications with SCA

- We mentioned the three important pieces of information:
  - The interface of the component
  - The implementation of the component
  - The access method to invoke the component
- With SCA, the implementation and the access method are determined at runtime by the infrastructure.
  - Our code isn't involved in this determination, so we don't have to write it or maintain it.

Why SCA makes life simpler

What SCA *is:*

- A simplified programming model for using components
  - SCA components can be BPEL processes, Java POJOs, EJB session beans, PHP scripts, C++ apps….
- An executable model for assembling services
  - You create the definition, then deploy it
- A standard way of expressing dependencies
  - The definition of a component includes all the other components it uses.

What SCA *is not:*

- Tied to a specific programming language, protocol, technology, runtime, etc.
- Web services
  - SCA can access RMI services or local objects, not just SOAP services.
- Workflow
  - Use BPEL for that

Everything's a POJO

- SCA gives your developers a single programming model for using services.
- As your SOA gets more complicated, your developers have to learn more and more interfaces.
  - EJBs, RMI, JCA, JAX-WS, JAX-RPC (and that's just Java!)
- With SCA, you're using a POJO:
  - **`myService.doSomething(x,y);`**
  - We don't know where **`myService`** is, how it's accessed, how it's written, what policies apply to it, and so forth. To us, it's just a POJO.

CASCON [2008]

Everything's a POJO

- Invoking an actual POJO:
  - **`myService.doSomething(x,y);`**
- Invoking a Web service:
  - **`myService.doSomething(x,y);`**
- Invoking an RMI service:
  - **`myService.doSomething(x,y);`**
- Invoking a Web service with digital signatures:
  - **`myService.doSomething(x,y);`**
- And so forth….

Freedom of choice

- Because every SCA component works like a POJO, you have more freedom when you choose a service provider.
- For example, say you discover a great service based on RMI.
  - With SCA, your developers don't have to know anything about RMI to use the service.
  - Without SCA, if your developers don't know RMI, you have to balance the training costs and risk factors against the benefits of the new service.

SCA simplifies your applications

- Another way to look at SCA is that it takes all of the details of access methods, implementations, encryption, authentication, etc. and moves them into the middleware layer.
  - Application developers write business logic, code that actually builds value for your organization.
  - The details of using services are handled by SCA.
  - As the details change, your applications (and the developers who wrote them) aren't affected.

CASCON [2008]

SCA simplifies your applications

- With SCA, you can change the definition of a component without changing the applications that use it.
- The original code:
  - **`myService.doSomething(x,y);`**
- After changing the component so that every request requires a digital signature:
  - **`myService.doSomething(x,y);`**

SCA simplifies adminstration

- SCA gives you a single declarative way to establish policies.
  - "Requests to this service must be authenticated."
  - "Requests to this service must be digitally signed."
- For example, here's how you require digital signatures:
  - **`sca:requires="integrity"`**
- The SCA runtime implements the policy, the application does not.

SCA simplifies governance

- With SCA, you define a component one time, in one place, then point your applications to that definition.
  - If all of the applications use the same definition, you know what components your organization uses.
  - If you need to change the component or how it works, you make that change one time, in one place.
- SCA makes it easier to track what components are being used, and SCA lets you change those components without changing your applications.

The SCA specs

- There are four parts to the specs:
  - The Assembly Model: How to define composite applications
  - The Client and Implementation specifications: How to use SCA in different languages
  - Binding specifications: How to use access methods
  - Policy Framework: How to add security, transactions, conversations, reliable messaging, etc. *declaratively*

Client and Implementation specs

- If you're using Java, there are a couple of specific documents that apply to you:
  - Java Component Implementation Specification
  - Java Common Annotations and APIs
- If you're using Spring, there is a specific document that applies to you:
  - Spring Component Implementation Specification
- If you're using BPEL:
  - Client and Implementation Model Specification for WS-BPEL
- If you're using C++:
  - Client and Implementation Model Specification for C++

Bindings and Policies

- There are three specific binding specifications:
  - Web Service Binding Specification
  - JMS Binding Specification
  - EJB Session Bean Binding
- Policies allow you to define characteristics of services
  - Encryption, authentication, integrity, QoS, conversational, etc.

SCA application diagrams

The SCA development model

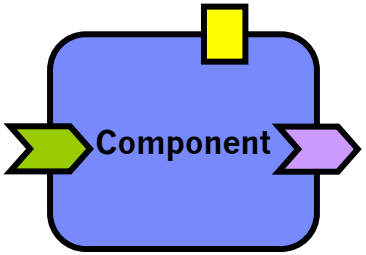
- SCA has a consistent model:
  - An individual piece of code is called a ***component***.
  - Components can be grouped into ***composites***.
    - *Components are atoms, composites are molecues. (David Chappell)*
  - Components and composites are hooked together with ***wires***.
  - Components and composites can have ***properties***.
- We'll use SCA assembly diagrams to illustrate these concepts.

Symbols in SCA assembly diagrams

- Here are the symbols used in SCA assembly diagrams:
  -  A **green chevron** represents a **service**. This is an entry point to the SCA component or composite.
  -  A **purple chevron** represents a **reference**. This points to a service provided by something else.
  -  A **line** represents a **wire**. This is the connection between a service reference and the service itself.
  -  A **yellow rectangle** represents a **property**. This is a value you can set when you invoke the component or composite.

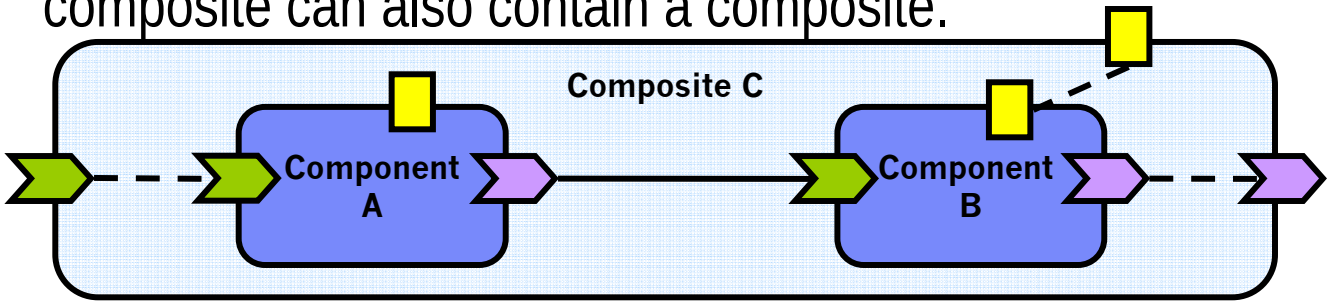
SCA symbols

- More symbols:



A **rounded rectangle** represents a **component**. A component can have services, references and properties.

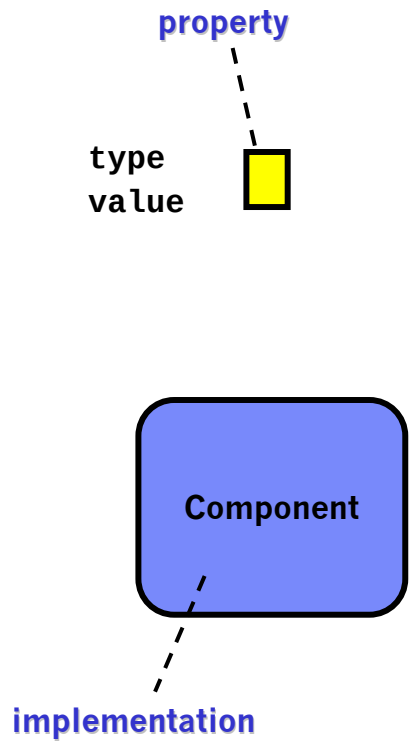
A **large rounded rectangle** represents a **composite**. A composite contains one or more components. Like a component, it can have services, references and properties. A composite can also contain a composite.



Services and references

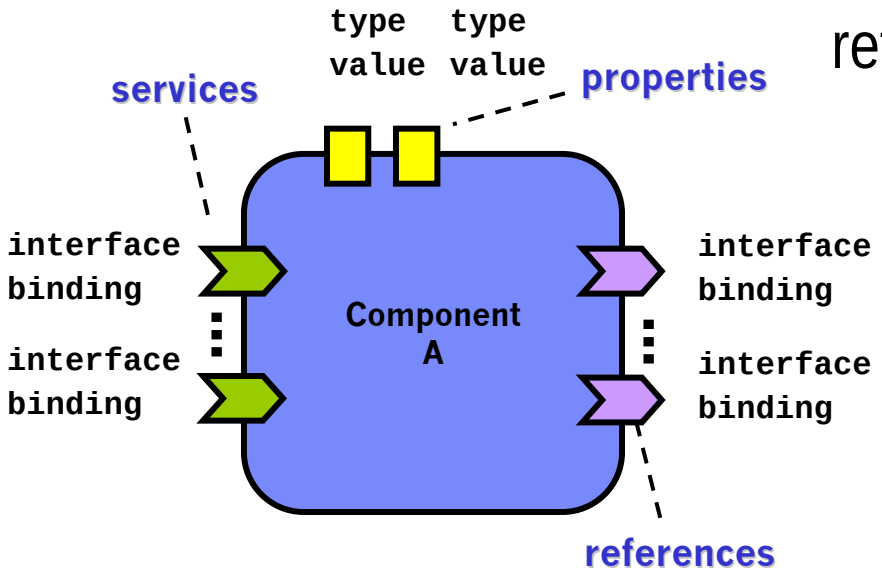
- A service or a reference has an **interface** and a **binding**.
  - The interface might be a Java interface, a WSDL port type, a BPEL partner link, a C++ class, etc.
  - The binding defines the access method. It might be SOAP/HTTP, JMS, ATOM, JSON, RMI-IIOP, SCA, etc.
- A component or composite can provide as many services and use as many references as it needs.

Prooperties and implementations



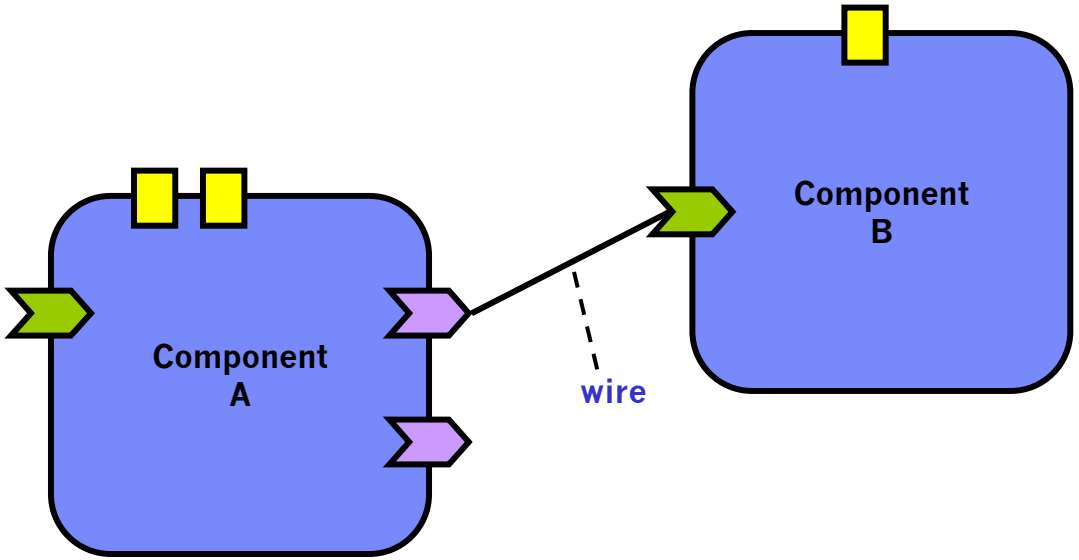
- A **property** has a type and a value.
- A component has an **implementation**; that's the code that actually provides the service.
- The implementation might be PHP, BPEL, Java, C++, Spring, etc.
- We won't show the implementation in most of our diagrams; an application doesn't care what it is.

A component

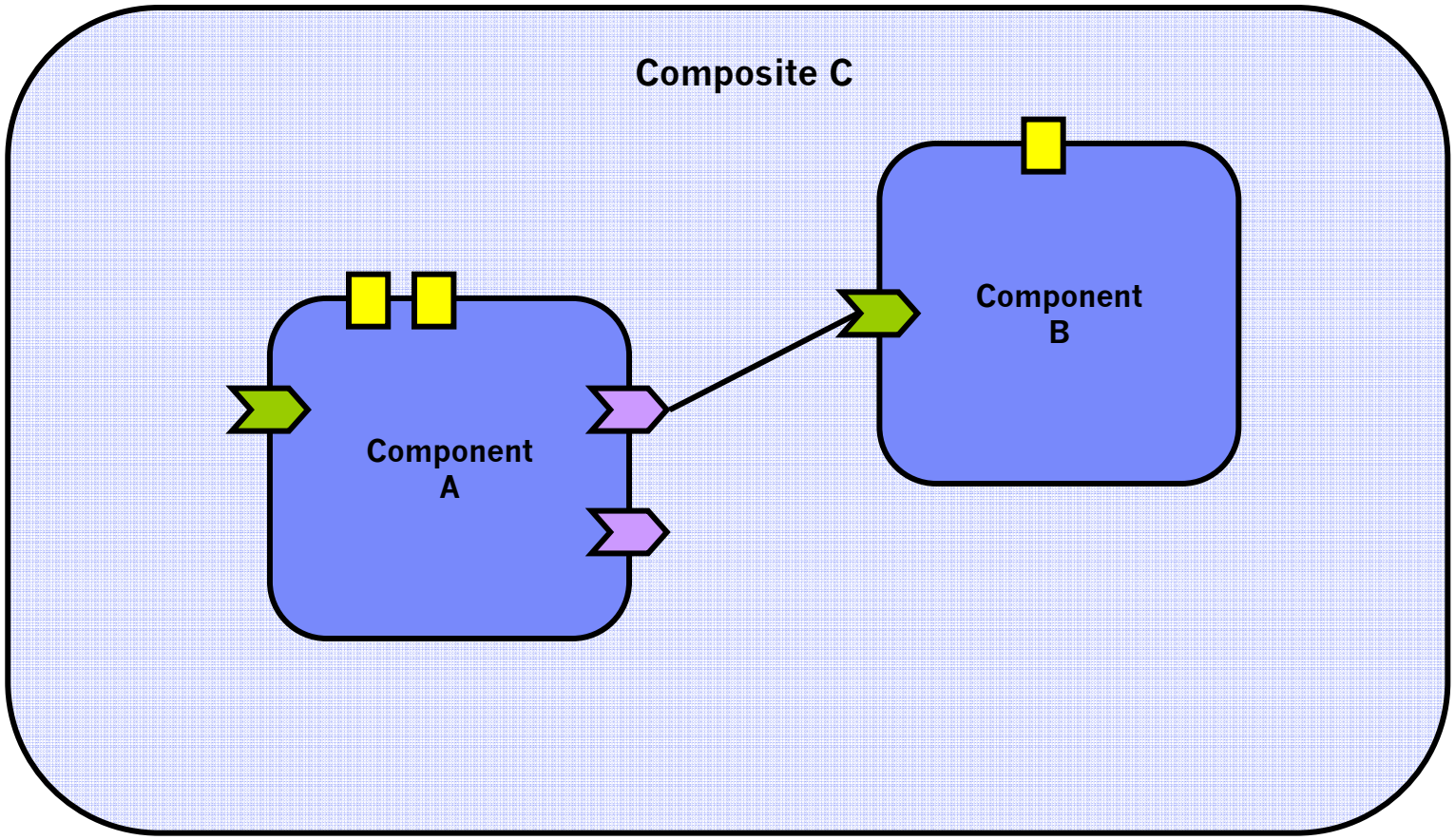


- This diagram is a component with services, references and properties.

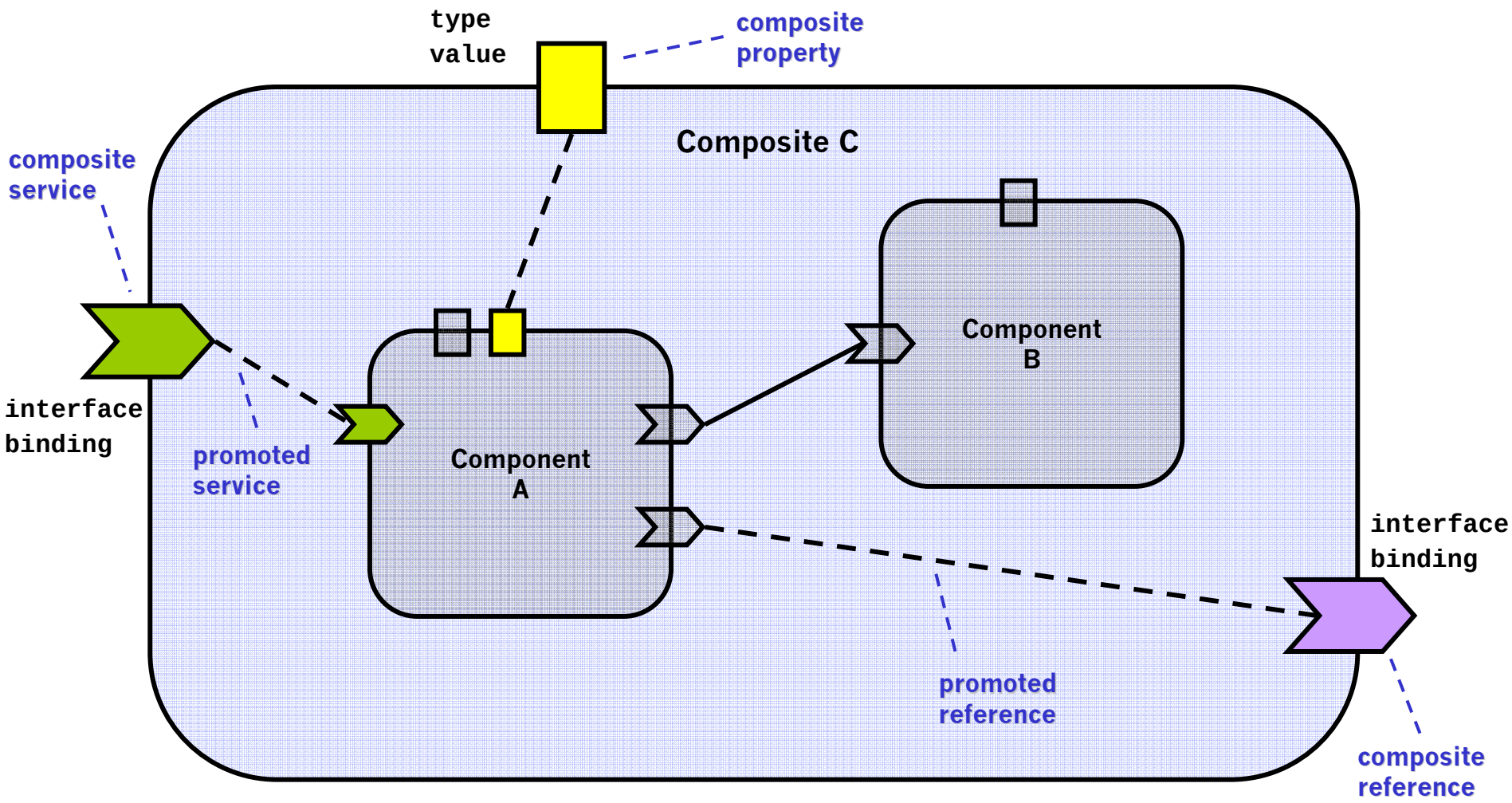
Wiring



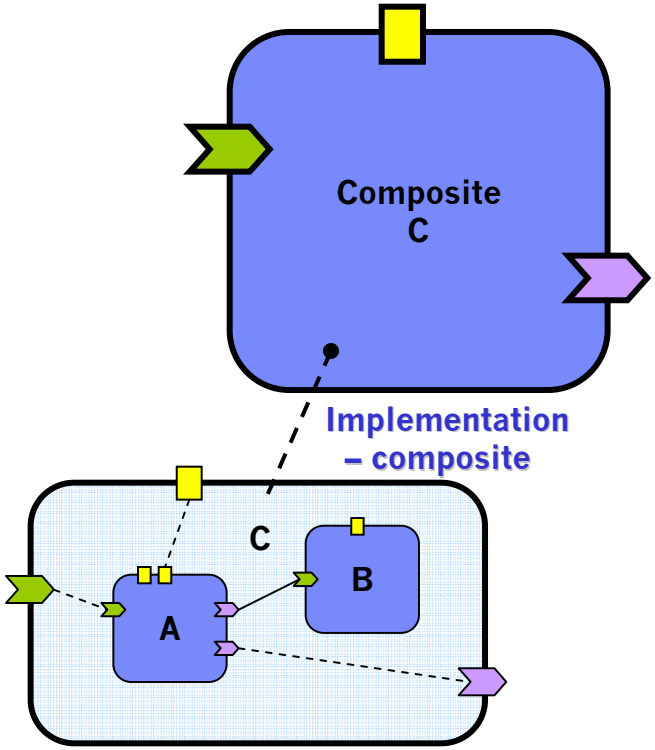
A composite



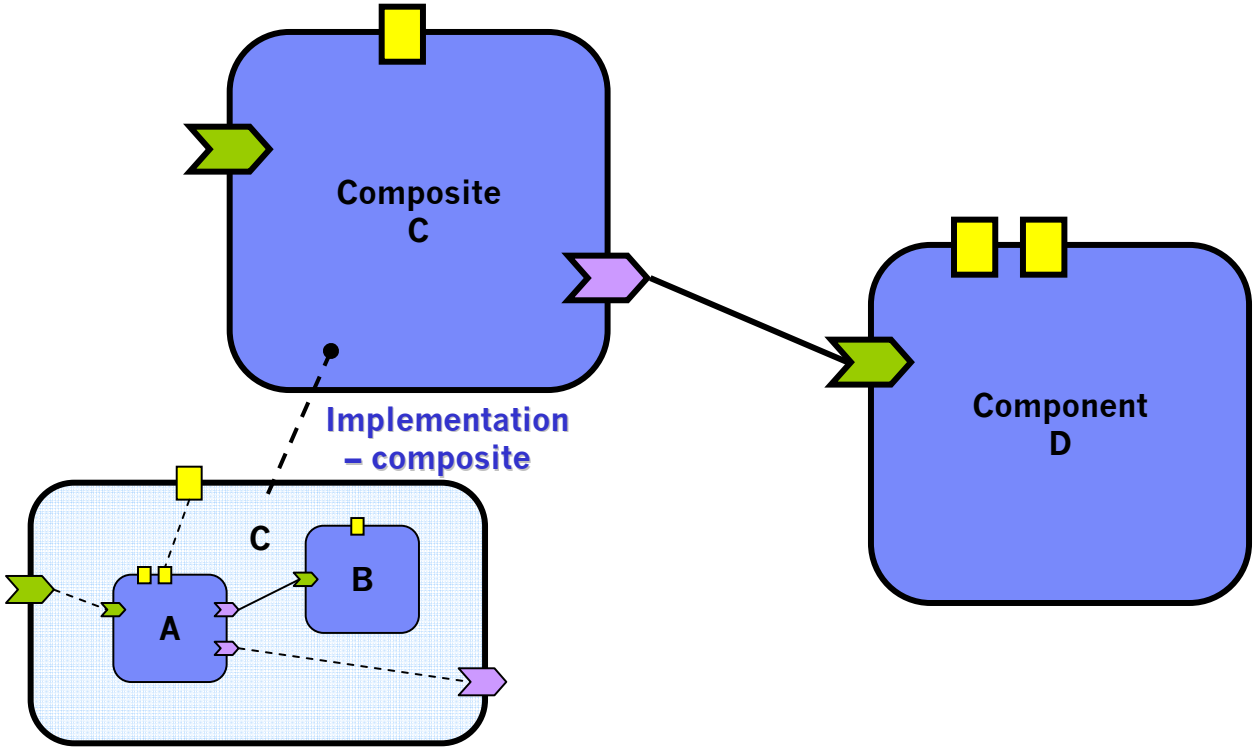
Promotion


type
value
composite
property
Composite C
composite
service
interface
binding
promoted
service
Component
A
Component
B
interface
binding
promoted
reference
composite
reference

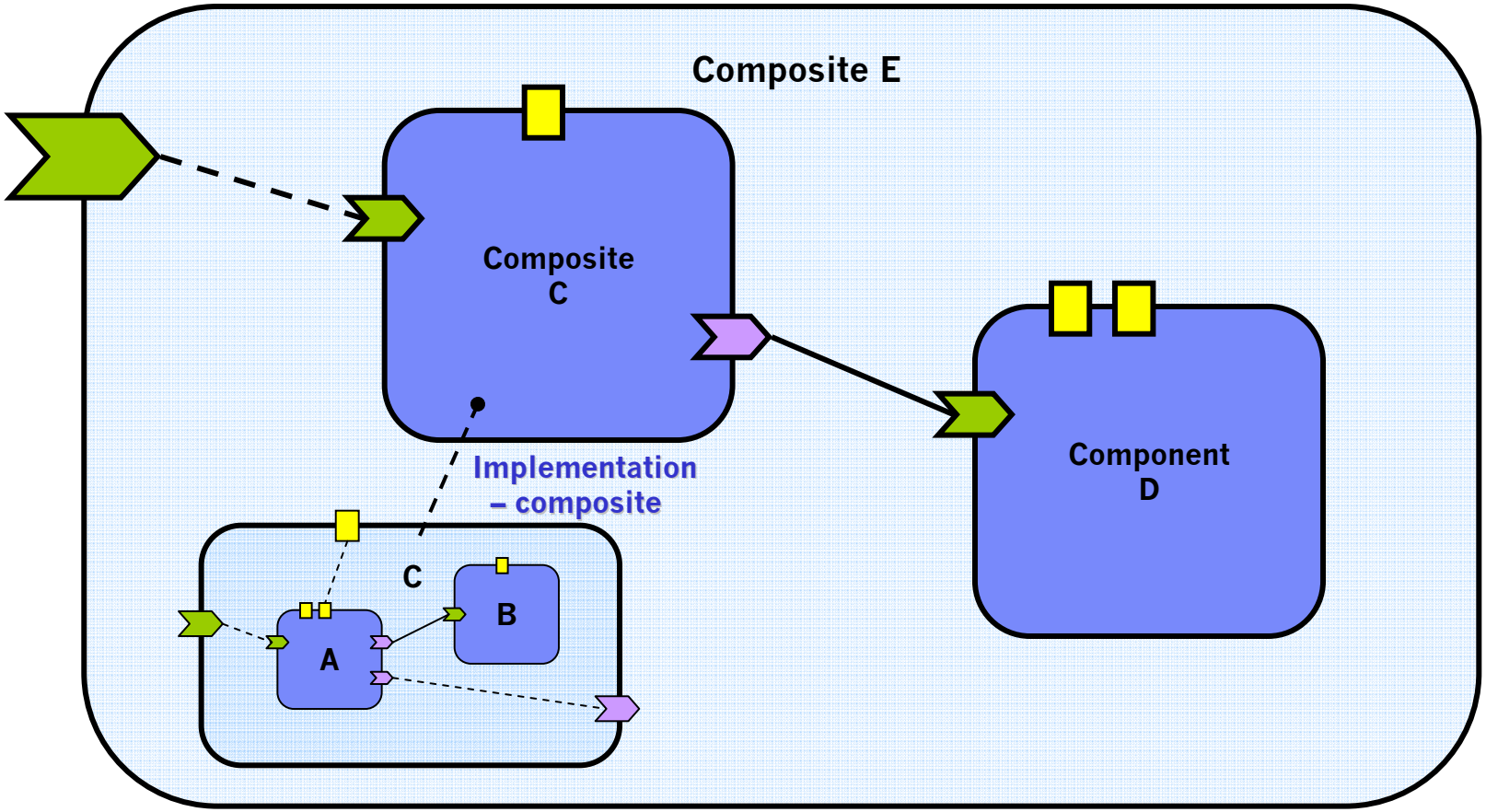
A composite implementation



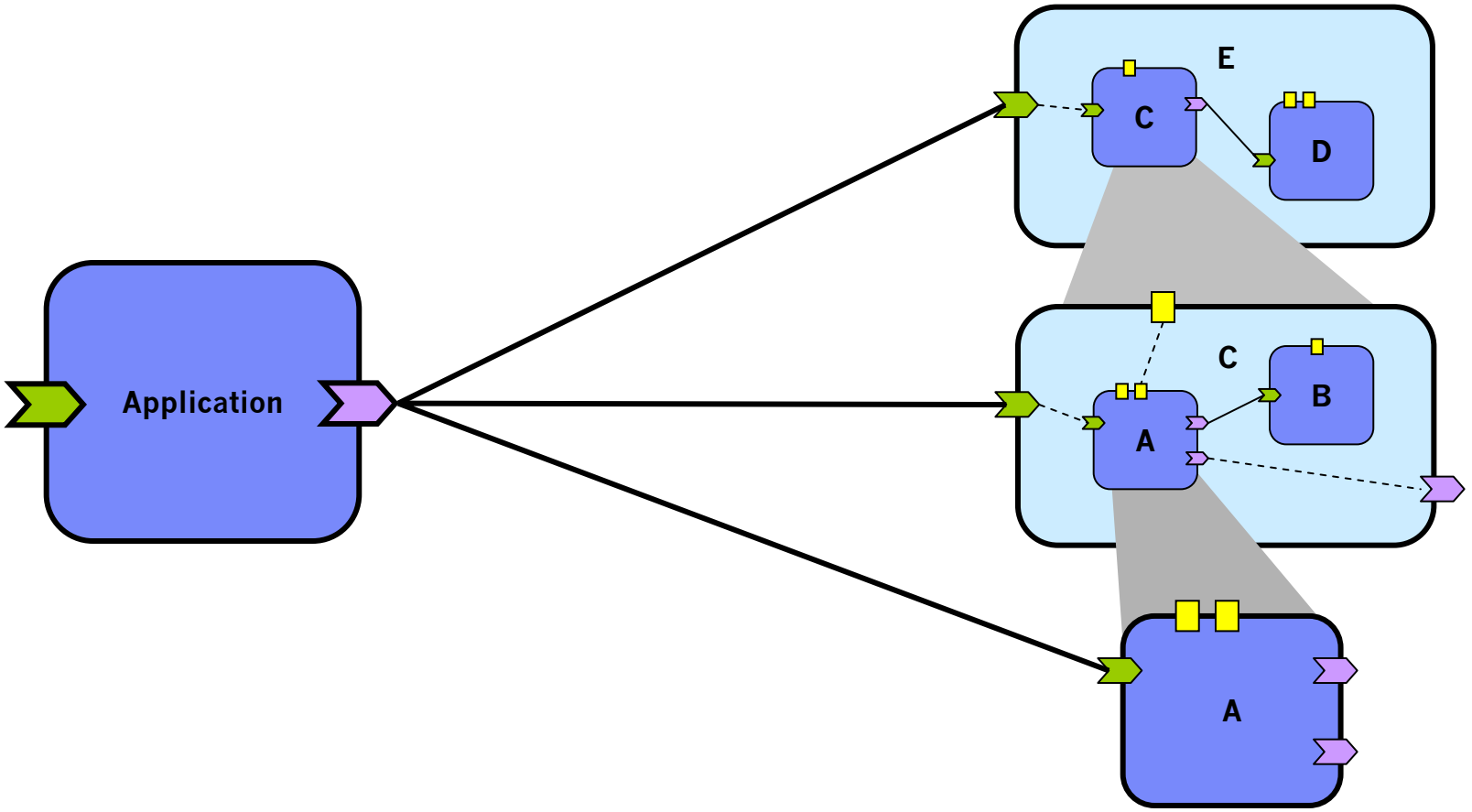
A composite using another component



A composite that includes a composite



SCA's recursive assembly model



Exercises

The Apache Tuscany project

CASCON [2008]

- Our exercises today are based on the Tuscany project.
- The Tuscany project is an open source implementation of the SCA and SDO specs.
  - For SCA, there is support for components written in BPEL and Java; other languages are being added.
  - Tuscany Java also supports the Bean Scripting Framework. That means you can write components in JavaScript, Ruby, Python, Groovy, Haskell, etc.
  - There are also C++ and PHP implementations of SCA and SDO.

The calculator demo

- Our demo application is a calculator. The class we'll run is **`CalculatorClient`**. This class gets a **`CalculatorService`** from the SCA runtime, then it calls the **`add`**, **`subtract`**, **`multiply`** and **`divide`** methods.
  - **`CalculatorService`**, **`AddService`**, **`SubtractService`**, **`MultiplyService`** and **`DivideService`** are all Java *interfaces*.

The calculator demo

- The **`CalculatorClient`** class creates an **`SCADomain`** based on the definitions in a **`.composite`** file.
  - The name of the **`.composite`** file is passed in as a parameter.
- The **`CalculatorClient`** class and the interfaces it calls never change.

The calculator demo

- We'll use **`CalculatorClient`** to invoke:
  - A POJO
  - An RMI service
  - A service with an incompatible interface
  - A Web service
  - A Web service with authentication tokens in the SOAP header
  - A Web service with a digitally-signed request

A generic SCA client

- Here's a generic SCA client application:

```
// Load a .composite file
SCADomain scaDomain = SCADomain.
  newInstance("x.composite");
CalculatorService calcServ =
  scaDomain.getService(CalculatorService.class,
    "CalculatorServiceComponent");
. . .
// Once calcServ is set, we just call it:
System.out.println(calcServ.add(3,2));
```

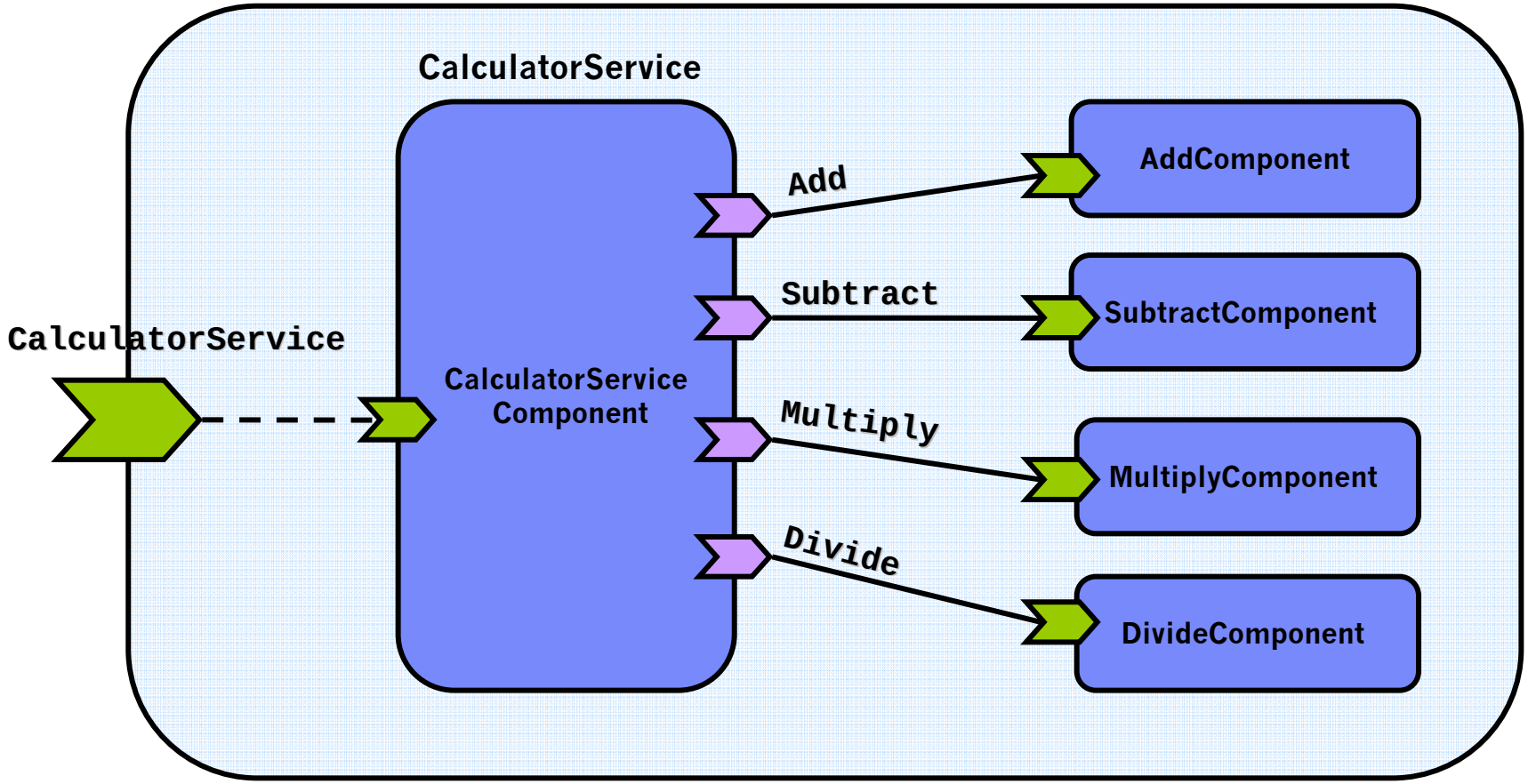
Exercise 1

Adding SCA to an application

Exercise 1

- In this exercise, you'll add SCA support to an application.
  - Edit **`CalculatorClient.java`** in the **`calculator`** package in the Exercises project.
  - Look for the text **`TODO:`**
  - You need to create the **`SCADomain`** and the **`CalculatorService`** object.
- Use the **`PojoCalculator.composite`** file.

The POJO composite



The POJO component definition

- In the **PojoCalculator.composite** file:

```
. . .
<component
  name="CalculatorServiceComponent">
  <implementation.java class="..."/>
  <reference name="addService"
    target="AddServiceComponent"/>
. . .
</component>
. . .
<component name="AddServiceComponent">
  <implementation.java class="..."/>
</component>
. . .
```

Finding the service definition

- We asked for two things in our code:
  - A new **`SCADomain`** created from the file ***`x`*`.composite`**
  - A service named **`CalculatorServiceComponent`**
- To simplify the example, both **`.composite`** files define components that have the same name.

Finding the `.composite` file

- There are different ways to find the file:
  - Use the **`.composite`** file name as a parameter.
  - Use the same name for every XML configuration file that you might want to use.
    - Change the Java **`CLASSPATH`** so that the SCA infrastructure finds one **`.composite`** file instead of another.
  - Put the name of the **`.composite`** file into a Java **`.properties`** file. Load the string at runtime.
  - Change the **`.composite`** file to point to a different service.

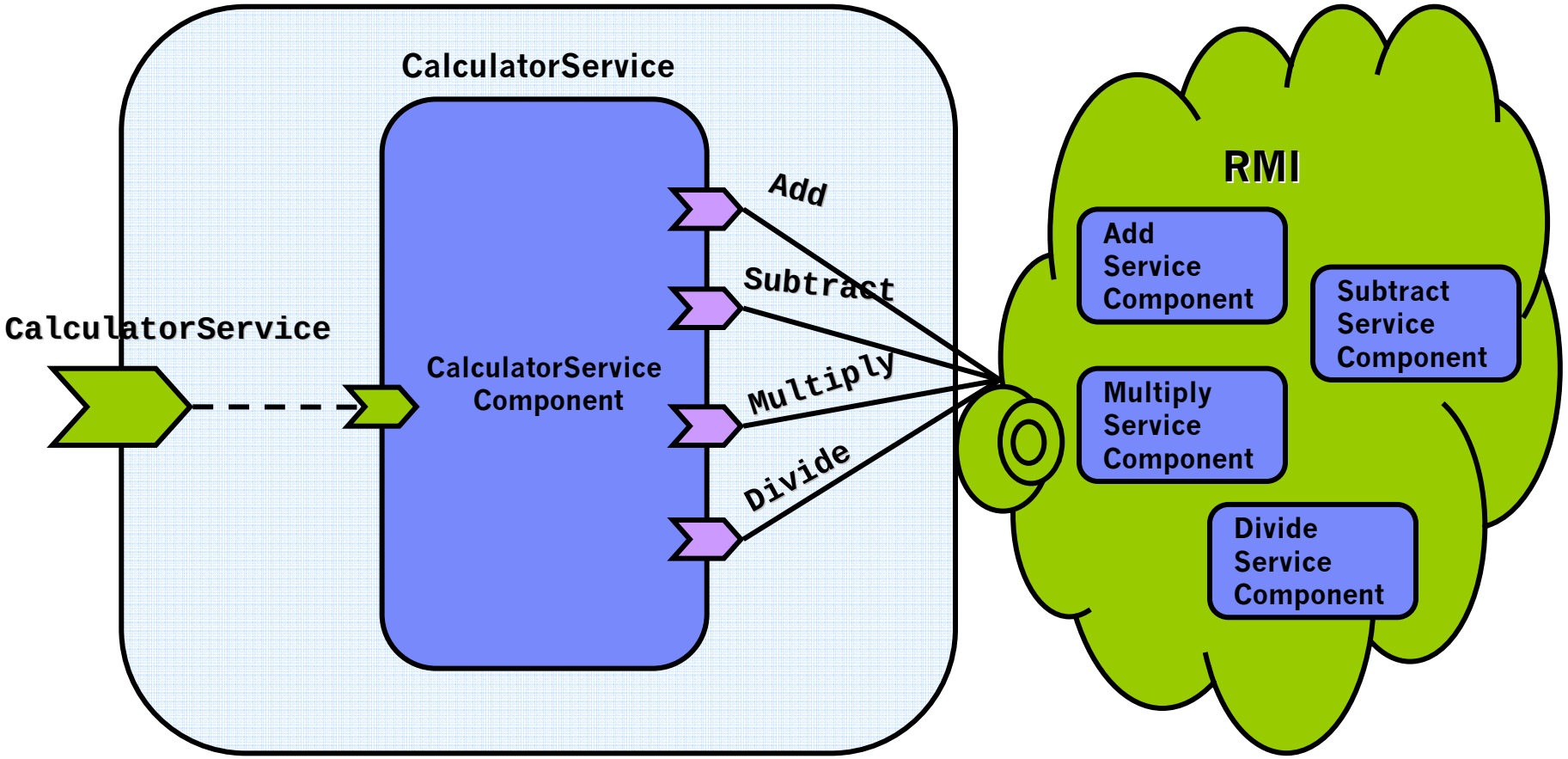
Exercise 2

Using an RMI service

Exercise 2

- In this exercise, you'll run the **`CalculatorClient`** application again, but you'll use an RMI service instead of a POJO.
  - Simply invoke the application with a different **`.composite`** file.
  - Be sure the RMI server is started before you invoke **`CalculatorClient`**.

The RMI composite



The RMI component definition

- In the **`RMICalculator.composite`** file:

```
. . .
<component
  name="CalculatorServiceComponent">
  <implementation.java class="..."/>
  <reference name="addService">
    <binding.rmi host="localhost"
     port="8099"
     serviceName="CalculatorRMIService"/>
  </reference>
. . .
```

Exercise 3

Using an SCA property

Properties

- Another feature of SCA is the ability to define **properties** for components.
- We've found a new calculator service that lets us define the number of decimal places in the result. The methods look like this:

```
double add(double n1, double n2,
  int precision);
```

- The **`precision`** parameter to these methods doesn't exist in our **`CalculatorClient`** application or the **`CalculatorService`** interface.

Properties

- We can define a property to handle the difference between the interfaces:
  - We'll set up a component that has the same interface as the original calculator class.
  - That component will use a property to define the **`precision`** parameter.
  - The component will call the methods of the new calculator class, passing in the property as the third parameter.

Properties

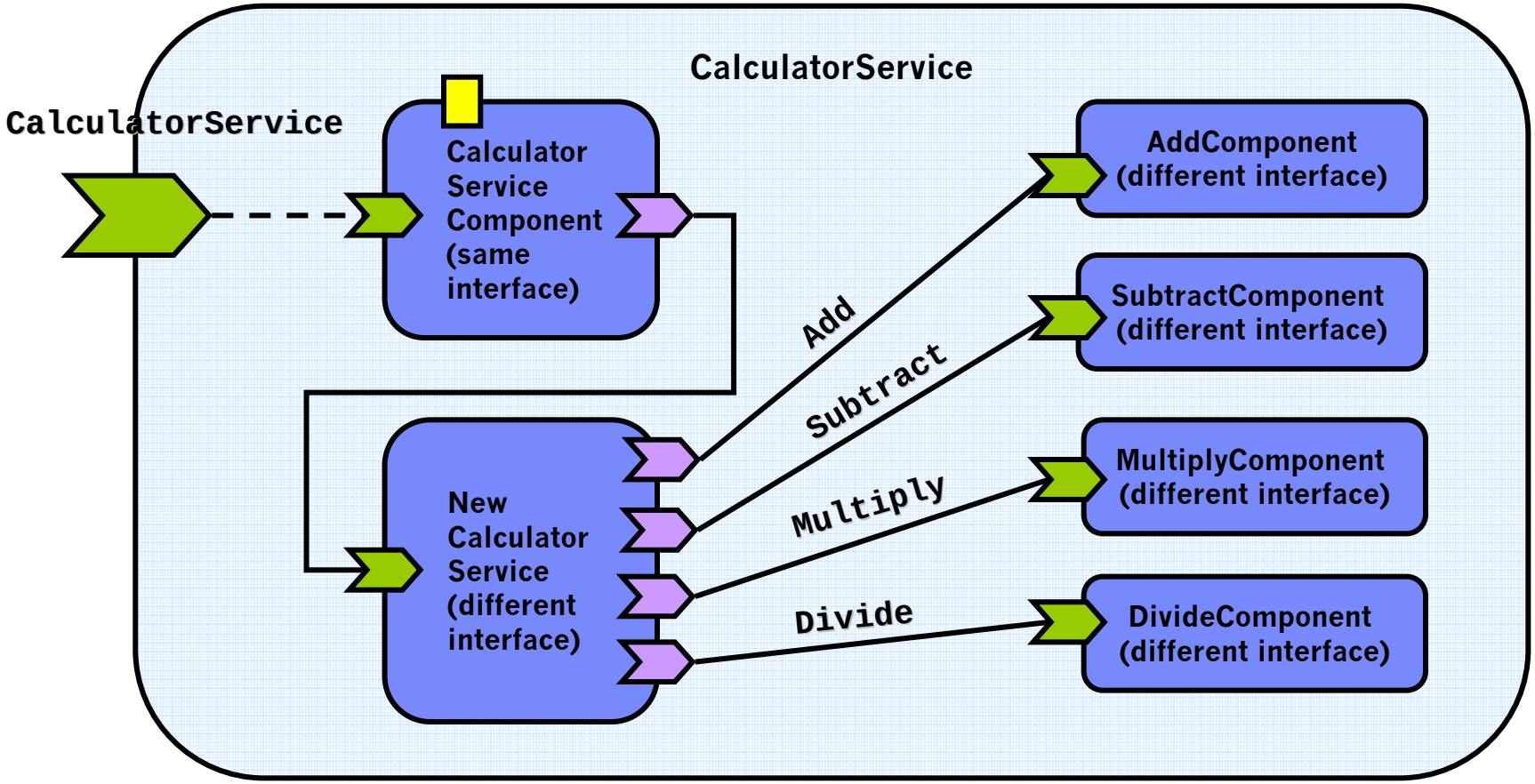
- Here's how the property looks in the service code:

```
private int precision = 2;
@Property(name="precision")
public void setPrecision(int
precision)
{
  if (precision >= 0)
    this.precision = precision;
}
```

Properties

- When the SCA domain loads the service, the property from the **`.composite`** file is passed to the **`setPrecision`** method.
- If the property isn't defined in the **`.composite`** file, the code uses the default value (**`2`**) that we coded in the service class.
  - If there's no default value and no property in the **`.composite`** file, the SCA runtime initializes the value to zero.

The composite with a property



Exercise 3

- In this exercise, you'll run the **`CalculatorClient`** application again, but you'll use the new precision calculator service.
  - Simply invoke the application with the **`PrecisionCalculator.composite`** file.
  - Change the value of the property in the **`.composite`** file and running the application again.

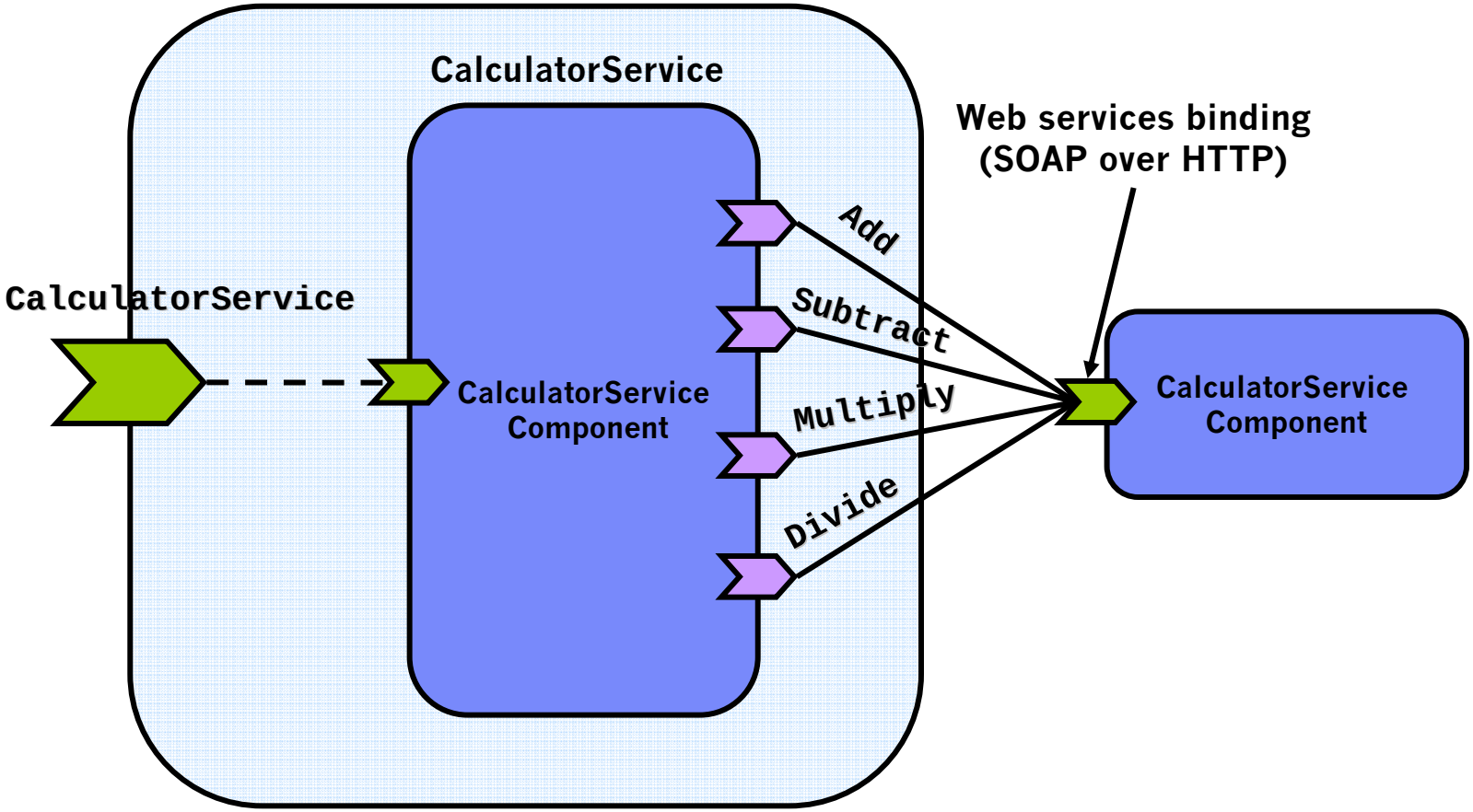
Exercise 4

Using a Web service

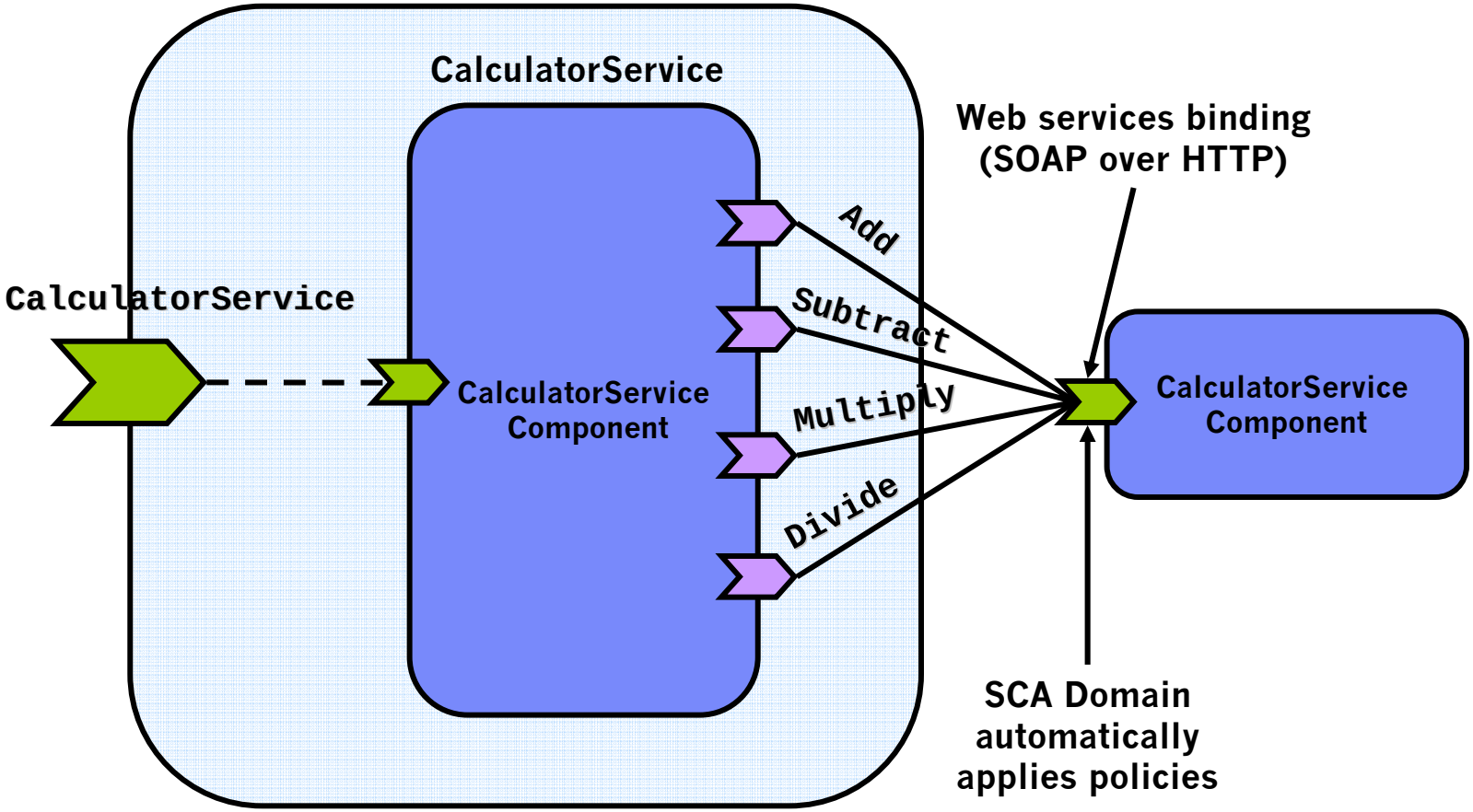
Exercise 4

- In this exercise, you'll run the **`CalculatorClient`** application to invoke a Web service.
- You'll invoke the Web service with three different **`.composite`** files:
  - One simply uses the Web service
  - One uses authentication
  - One uses digital signatures

The Web service composite



The Web service composite with policies



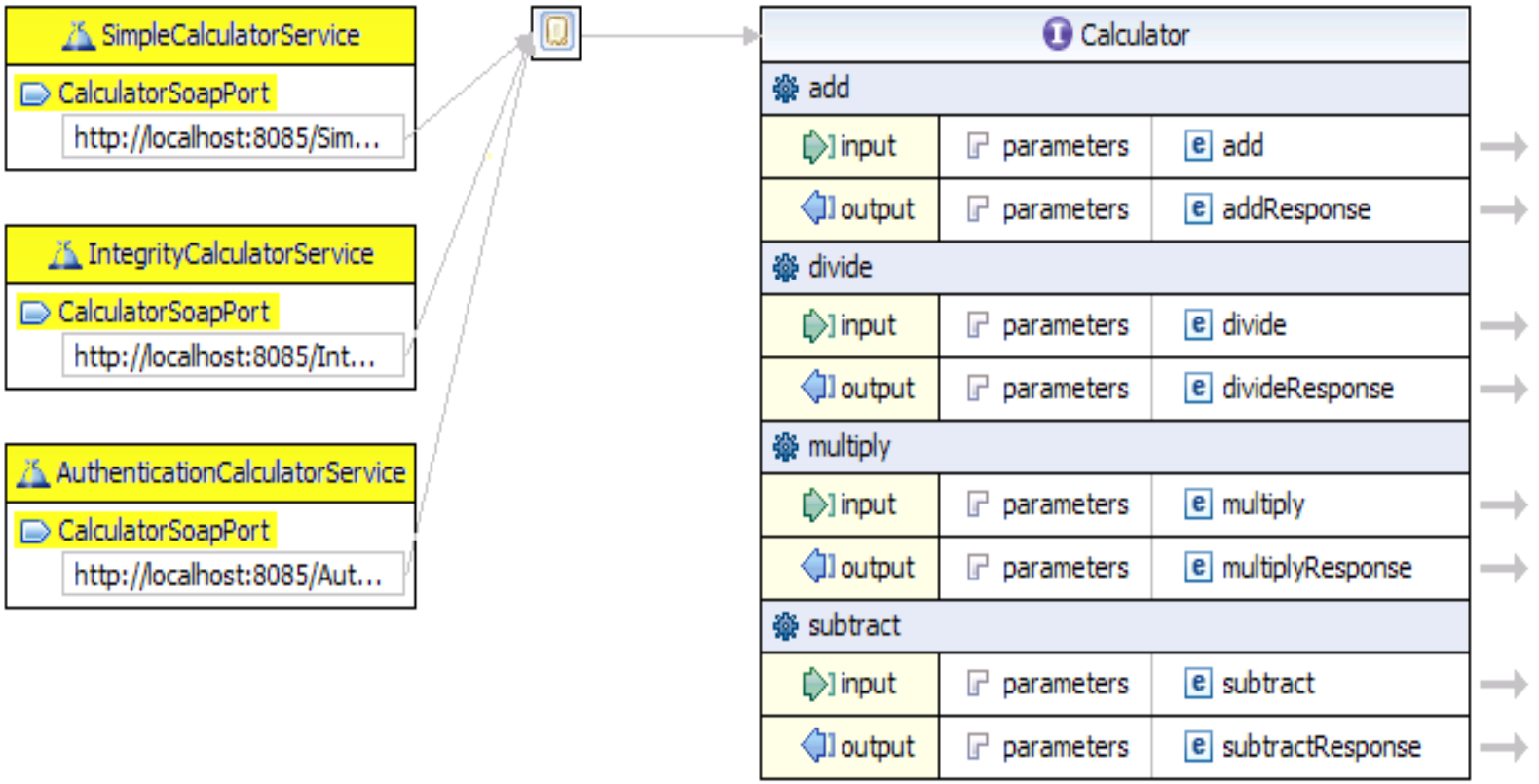
The Web service component definition

```
<component name="CalculatorServiceComponent">
  <implementation.java
    class="wsCalculatorClient.
           WSCalculatorServiceComponent"/>
</component>
<reference name="WsCalculatorService"
  promote="CalculatorServiceComponent/
  calculatorService">
    <interface.java
      interface="calculator.
                 CalculatorService"/>
    <binding.ws
      wsdlElement="http://calculator#
              wsdl.port(CalculatorService/
              CalculatorSoapPort"/>
</reference>
```

The Web service component definition

- The SCA runtime handles uses the WSDL file and the Java interface generated from it.
- The **`<interface.java>`** element refers to the same Java interface used for all the other examples.
- The **`wsdlElement`** attribute of the **`<implementation.wsdl>`** element consists of:
  - The **`targetNamespace`** from the WSDL file
  - A hash mark (#)
  - **`wsdlPort()`**, and in the parentheses:
  - The **`name`** of the **`<wsdl:service>`**
  - A slash (/)
  - The **`name`** of the **`<wsdl:port>`**

The WSDL file



The promoted reference

- The SCA domain uses the promoted reference in the **<component>** definition for dependency injection.
- In the Java code:
  - **public class WsCalculatorServiceComponent**
    . . .
    // This has public get and set methods
    CalculatorService calculatorService;
    . . .
- In the **.composite** file:
  - **<reference promote="CalculatorServiceComponent/calculatorService">**

Matches exactly; this is the field name, not the class name

Name of the component as defined in the **.composite** file

Dependency injection

- The SCA runtime uses dependency injection to resolve the reference.
- The class that implements the actual service (effectively a proxy that handles the SOAP calls) is generated.
- The SCA runtime creates an instance of that class and calls **`setCalculatorService()`** to set the class object.
- Once the object is defined, the service looks like a call to a POJO.

Bindings

- In SCA, a binding specifies how to access a service.
  - Current bindings include WSDL, RMI, JMS, JCA and EJBs.
  - More bindings are coming all the time at **`osoa.org`**.
  - Like all of SCA, the binding specification is open, so you can create your own.

Policies

- Previous standards efforts, WSDL in particular, didn't include how to define policies for services.
- SCA gives you a single declarative way to establish policies.
  - "This component must provide this level of QoS."
  - "All traffic on this wire must be digitally signed."
- To add authentication declaratively:
  - **`sca:requires="authentication"`**
- The SCA runtime implements the policy, the application does not.

Without SCA

- **It's more difficult to reuse your code.** The code is tied to a particular service implementation, so anyone who wants to reuse your code has to change that.
- **It's more difficult to maintain your code.** If you need to use a different service provider, you have to change the code.
- There's another drawback: Without SCA, the programmer has to know the details about the service endpoint and the access method.

SCA resources

The SCA & SDO portal

- **devx.com/IBMSCA/** has dozens of articles and resources for SCA.
- Registration for Webcasts, briefings and other events will be announced here.
  - There's a Webcast on the new SCA Feature Pack for WAS on 24 September.

The Apache Tuscany project

CASCON [2008]

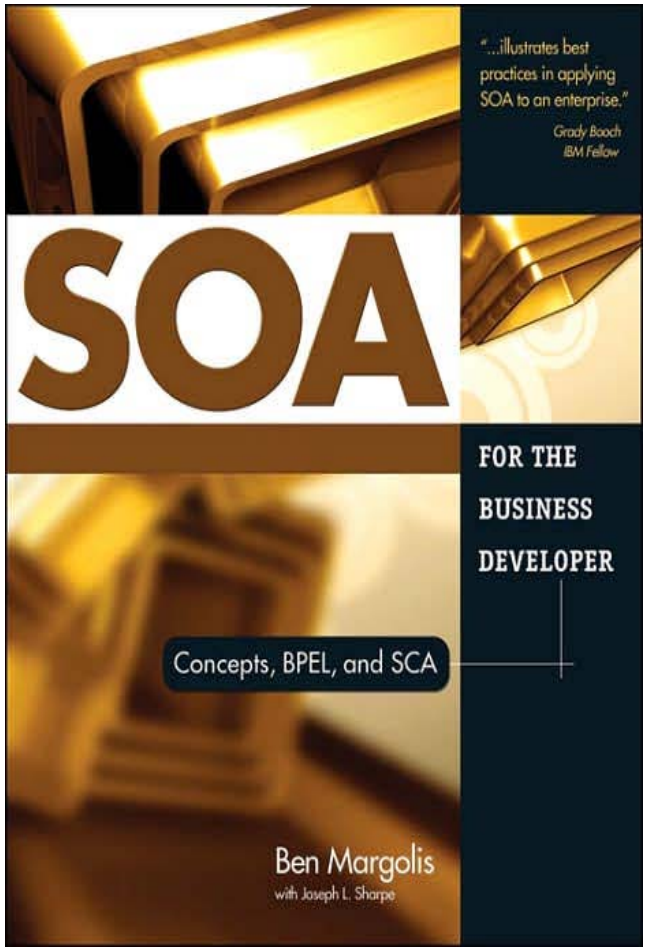
- The Tuscany project is hosted at **tuscany.apache.org**.
- The Tuscany dashboard at **tuscany.apache.org/tuscany-dashboard.html** is a great place to start.
- A C++ version is available as well.

osoa and OASIS Open CSA

- **osoa.org** is the original home of SCA and SDO. The specs are there, along with many helpful articles.
- The SCA and SDO standards work is being done at OASIS. See **oasis-opencsa.org**.

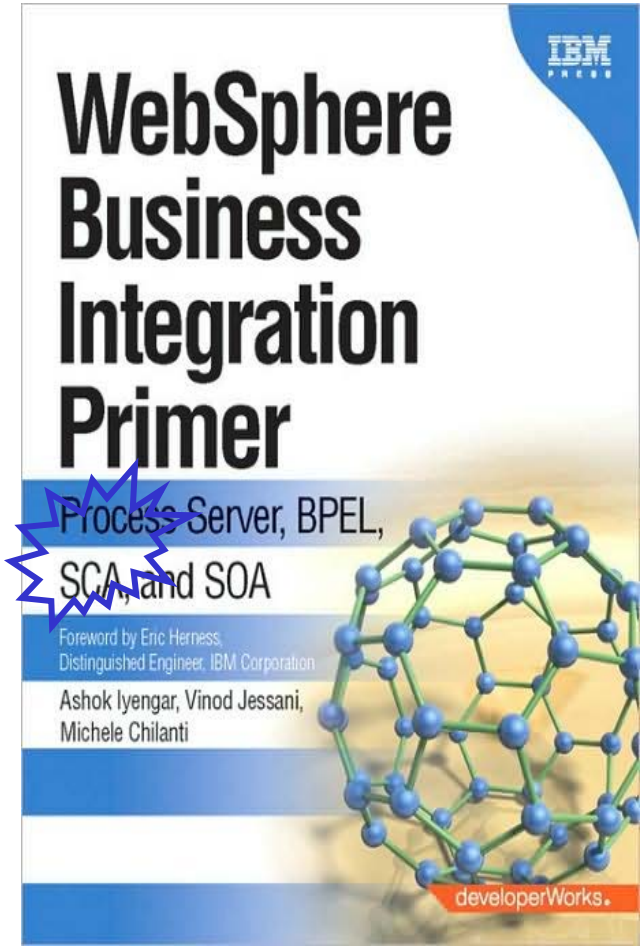
A great book on SCA

- Get Ben Margolis' book SOA for the Business Developer: Concepts, BPEL and SCA.
- *This is a great book on SCA*.
- Many of the best minds in the SCA world reviewed this book.
- ISBN 1-58347-065-4.
- mc-store.com/5079.html



A great book on *WebSphere* & SCA

- This book guides you through the complete business integration landscape, including SCA.
- It's full of samples that use real products (WPS, WID, Business Modeler, Business Monitor, WSRR).
- From IBM Press – ISBN 0-13-713672-2.
- You can buy the PDF online at **ibmpressbooks.com/bookstore/product.asp?isbn=0137136722**.



Thanks!

Doug Tidwell, IBM
dtidwell@us.ibm.com