The SOA Programming Model

Rob High, Jr.—IBM
Ed Cobb—BEA
Sanjay Patil—SAP
Greg Pavlik—Oracle

SCA Collaboration Team, www.osoa.org

TS-3608

Goal of this Talk

Discuss the role of Service Component Architecture and Service Data Objects in forming the basis of a SOA programming model

Agenda

Properties of Service Orientation

Roles and Components within SOA

Introspection on SCA and SDO

Java™ Language Bindings

Goals of SOA

- Business and IT alignment
  - Software design derived from an intrinsic understanding of business design
  - Separation of concerns and roles driven by business design goals
  - IT systems that enable business agility

Service Oriented Architecture

- In April 2006 the Object Management Group's (OMG) SOA Special Interest Group adopted the following definition for SOA
  - Service Oriented Architecture is an architectural style for a community of providers and consumers of services to achieve mutual value, that
    - Allows participants in the communities to work together with minimal co-dependence or technology dependence
    - Specifies the contracts to which organizations, people and technologies must adhere in order to participate in the community
    - Provides for business value and business processes to be realized by the community
    - Allows for a variety of technologies to be used to facilitate interactions within the community
- In March 2006 the OASIS group SOA Reference Model released its first public review draft. This defines the basic principles of SOA that apply at all levels of a service architecture, from business vision through to technical and infrastructure implementation
  - Service Oriented Architecture; a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.
  - It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations

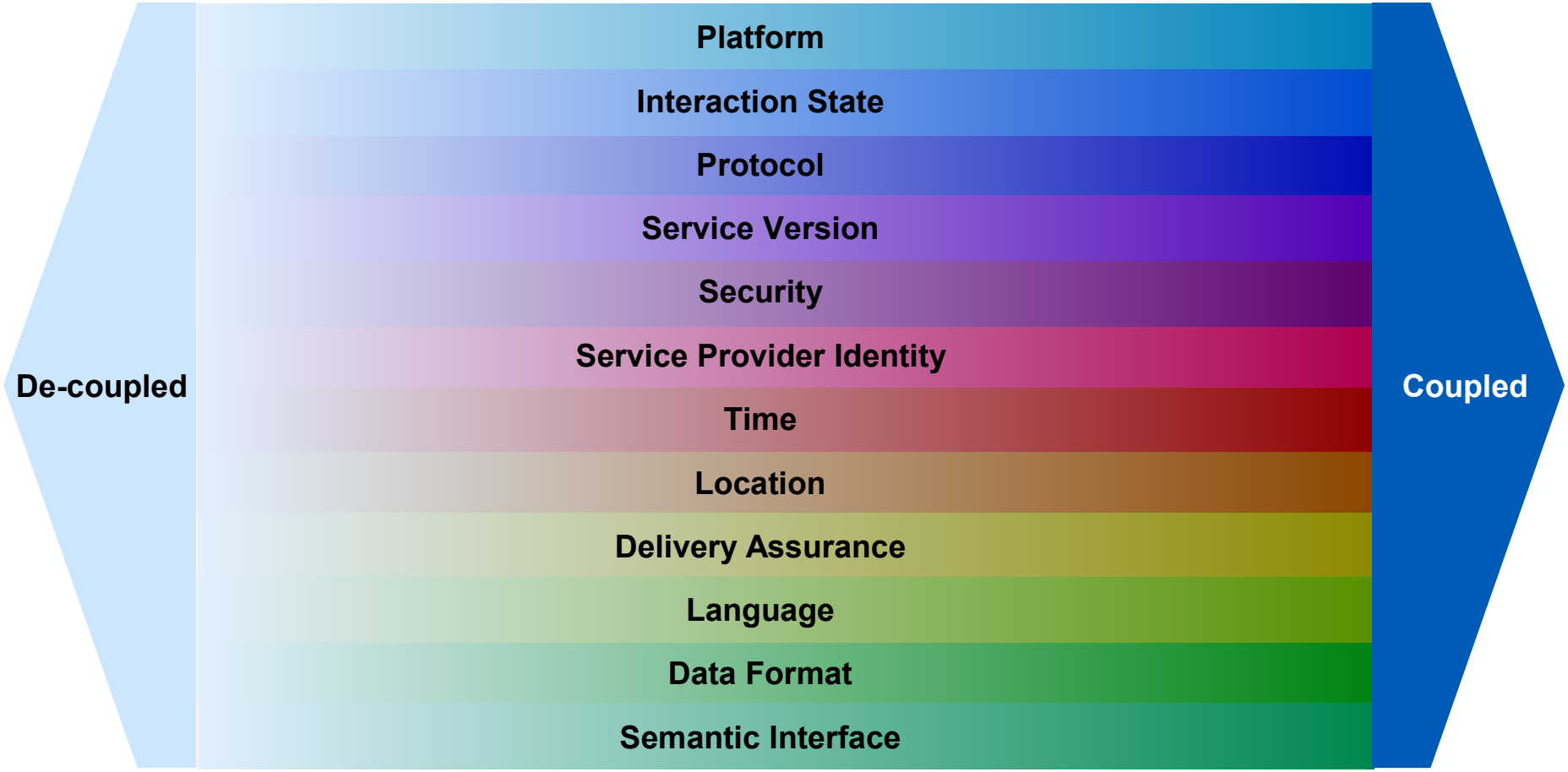
Source: Wikipedia

Principles of SOA

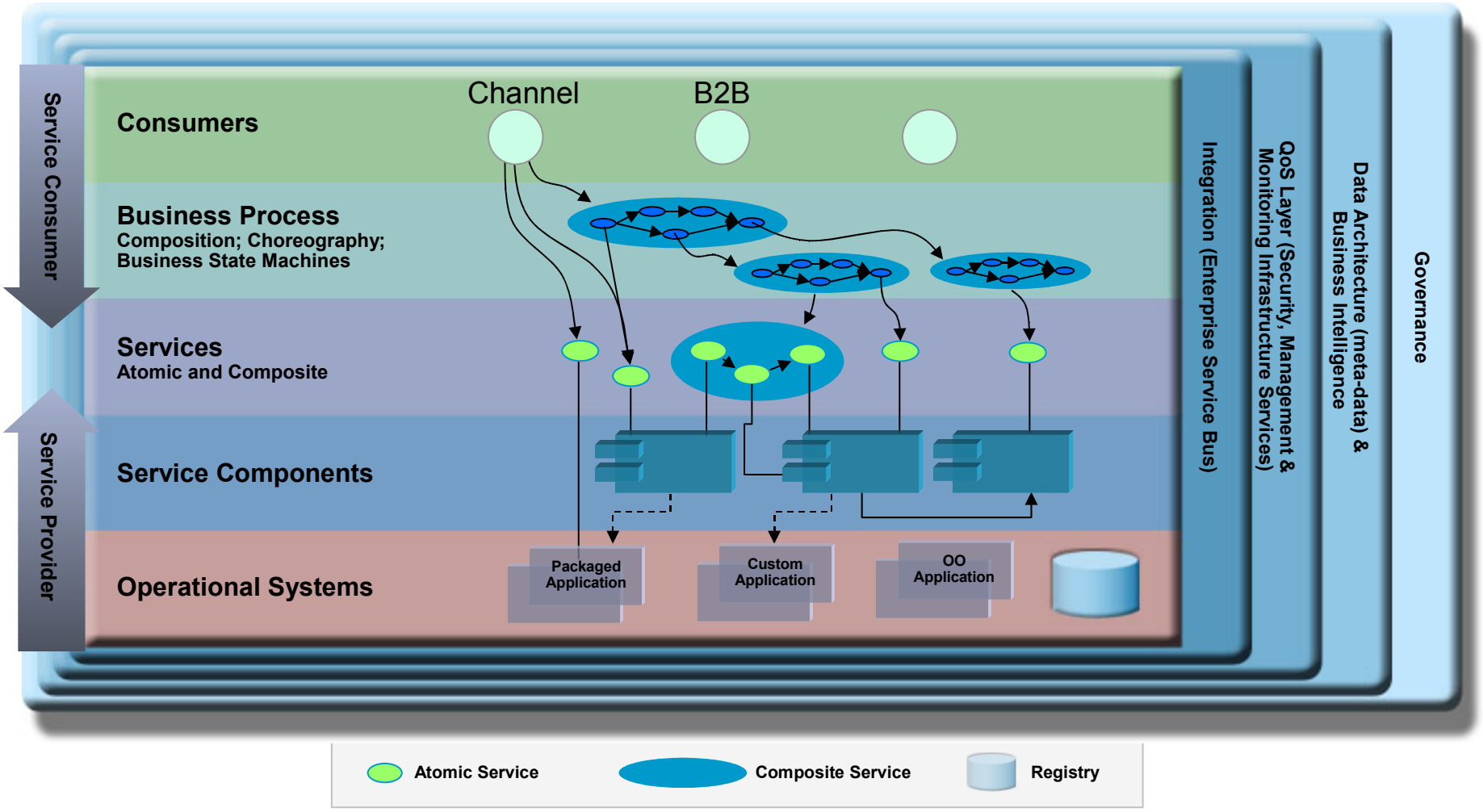
- Services share a formal contract
- Services are loosely coupled
- Services abstract underlying logic
- Services are composable
- Services are reusable
- Services are autonomous
- Services are stateless
- Services are discoverable

Source: Thomas Erl; SearchWebService.com

Loose-Coupling



Composite Applications



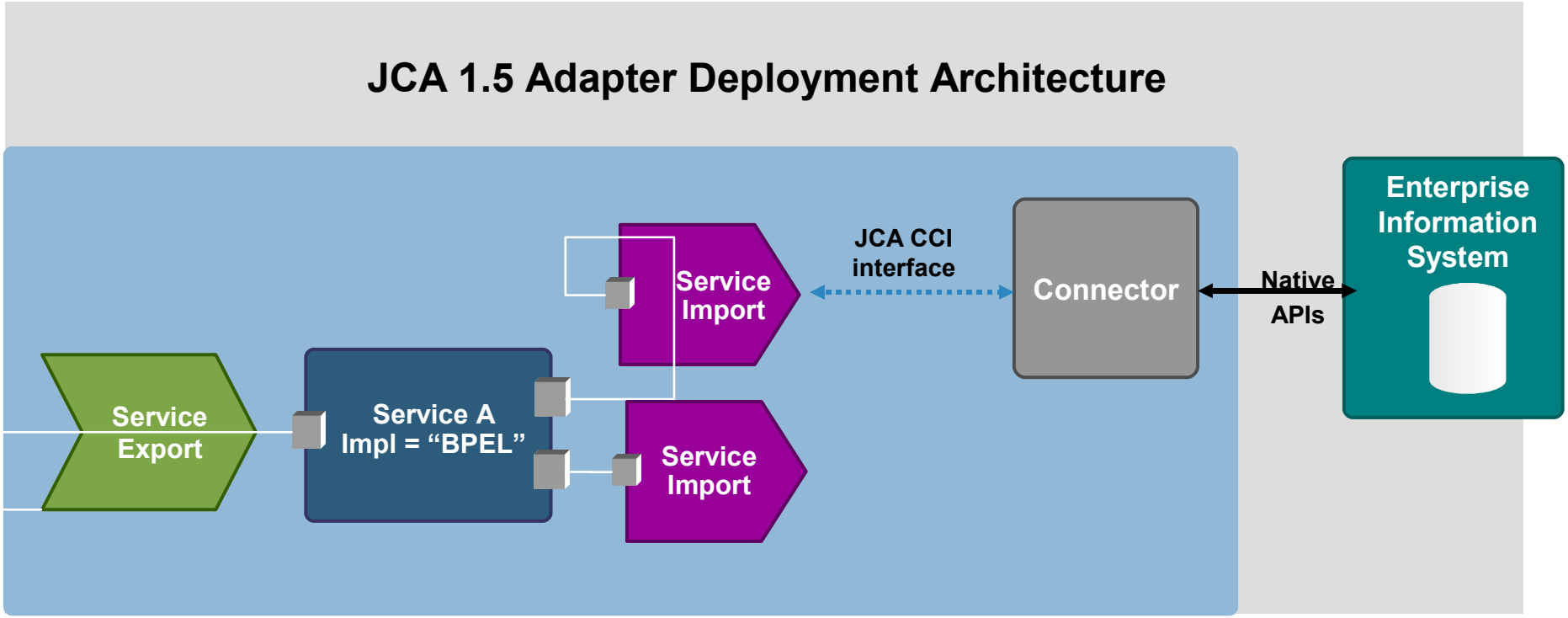
Composite Applications

- Applications are created by composing a set of reusable services
- Applications are Compositions
- Can be created easily, frequently, quickly—respond to the demands of your business in real-time
  - Exploiting the underlying services that you've taken more time to construct, harden, protect from the day-to-day implications of the business
- Separate the static, rigorous aspects of your application from the more dynamic, evolving, and customized aspect of your application
- Apply the appropriate language for the task
  - Java is a good language for service implementations
  - BPEL is a good language for service composition
- (Service Compositions are also Services that can be composed)

Legacy Application Components

- Significant amounts of relevant business function already exists in deployed systems
  - Java/J2EE™ platforms
  - CICS/COBOL
  - C++/Tuxedo/TXSeries/Orbix
  - MOM (MQ, Tibco, Sonic, etc.)
  - CORBA/IIOP
  - Packaged Apps
  - IMS
  - …
- Can often be adapted for use in Service Oriented solutions

Legacy Integration



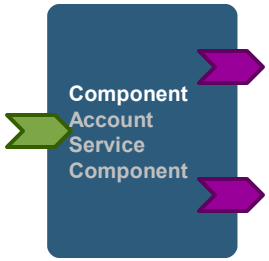
SOA Developer Roles

- Service Developer
  - Creates and publishes service implementations
- Service Consumer
  - Subcase of Service Developer—implements programs that consume services
- Service Composition Developer
  - Creates and publishes a class of service that composes other services
- Assembler
  - Assembles related services for deployment and operations management

Service Component Architecture

- A specification which describes a model for building applications and systems using a Service Oriented Architecture (SOA)
  - Service Component Architecture; building Systems using a Service Oriented Architecture.
  - A joint whitepaper by BEA, IBM, Interface21, IONA, Oracle, SAP, Siebel, Sybase, Version 0.9, November 2005 (http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-sca/SCA_White_Paper1_09.pdf)
- A technology and language-neutral representation of services (and data, when combined with SDO) that can be deployed to a variety of different hosting environments—composing services implemented in a variety of heterogeneous containers and frameworks
  - Includes provisions for mapping to specific languages and technologies
  - Emphasis on a wiring metaphor to enable composition
- Jointly developed and written by IBM, BEA, Oracle, SAP, IONA, Sybase, Interface21, Siebel
  - Currently at 0.9 level draft
  - Will be completed and submitted for formal standardization

Service Component

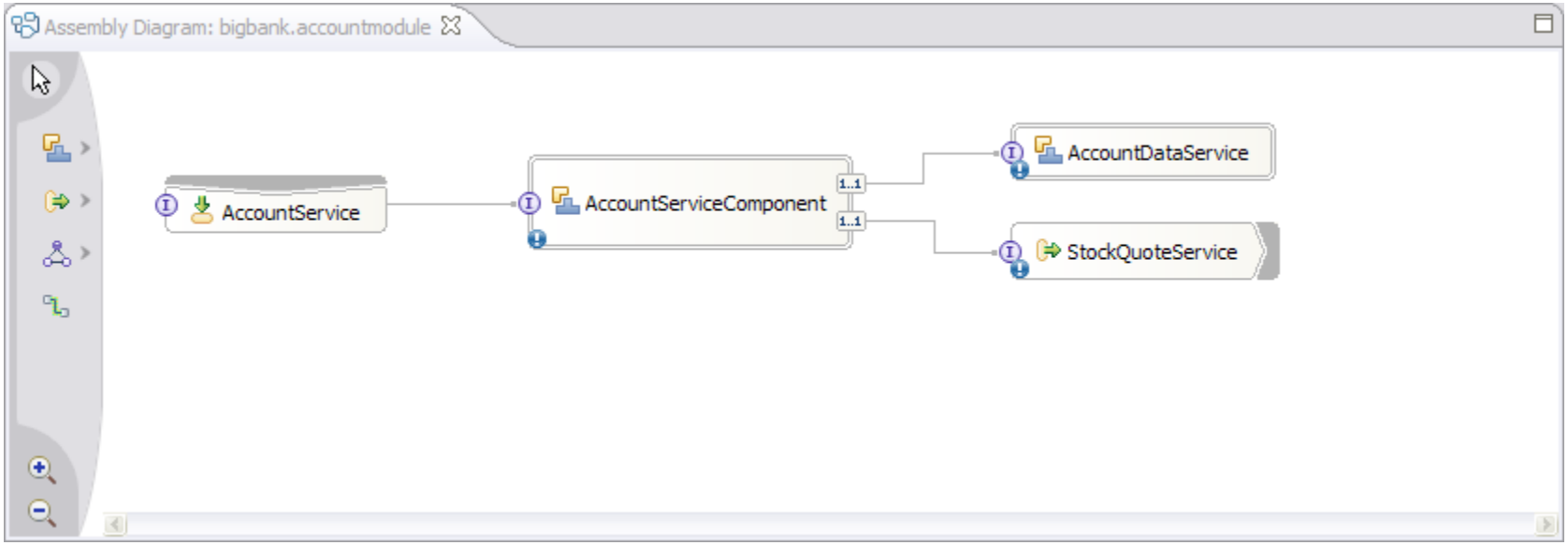


- Configured instance of an implementation
  - There can be more then one component using the same implementation
- Provides and consumes services
- Sets properties; overridable (no, may, must)
- Sets service references by wiring them to services
  - Wiring to services provided by other components or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

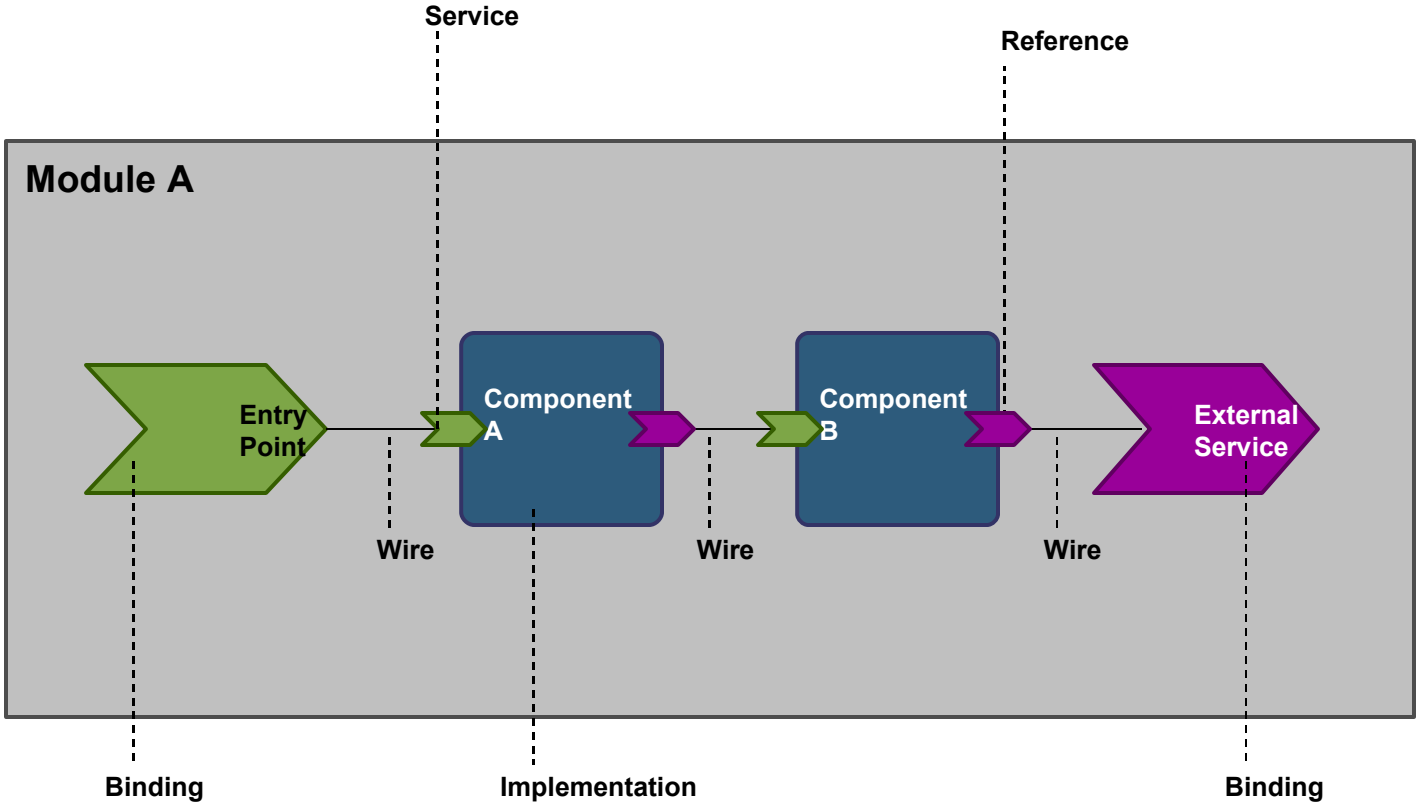
```
<?xml version="1.0" encoding="ASCII"?>
<module      xmlns="http://www.osoa.org/xmlns/sca/0.9"
       xmlns:v="http://www.osoa.org/xmlns/sca/values/0.9"

  name="bigbank.accountmodule" >

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

</module>
```

Assembly

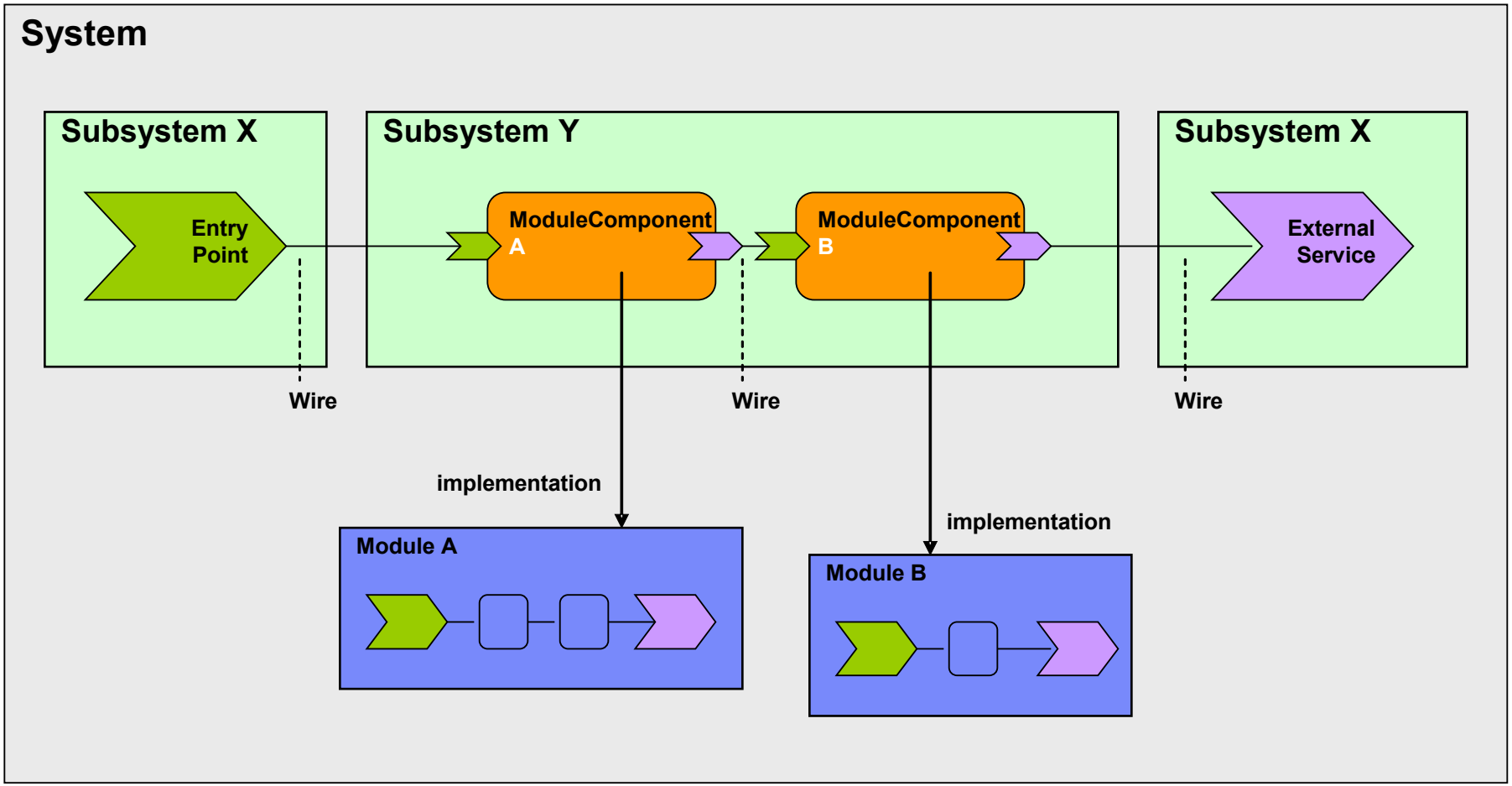
- Composes one or more services and their relationships, dependencies, policies and declared visibility in a Module



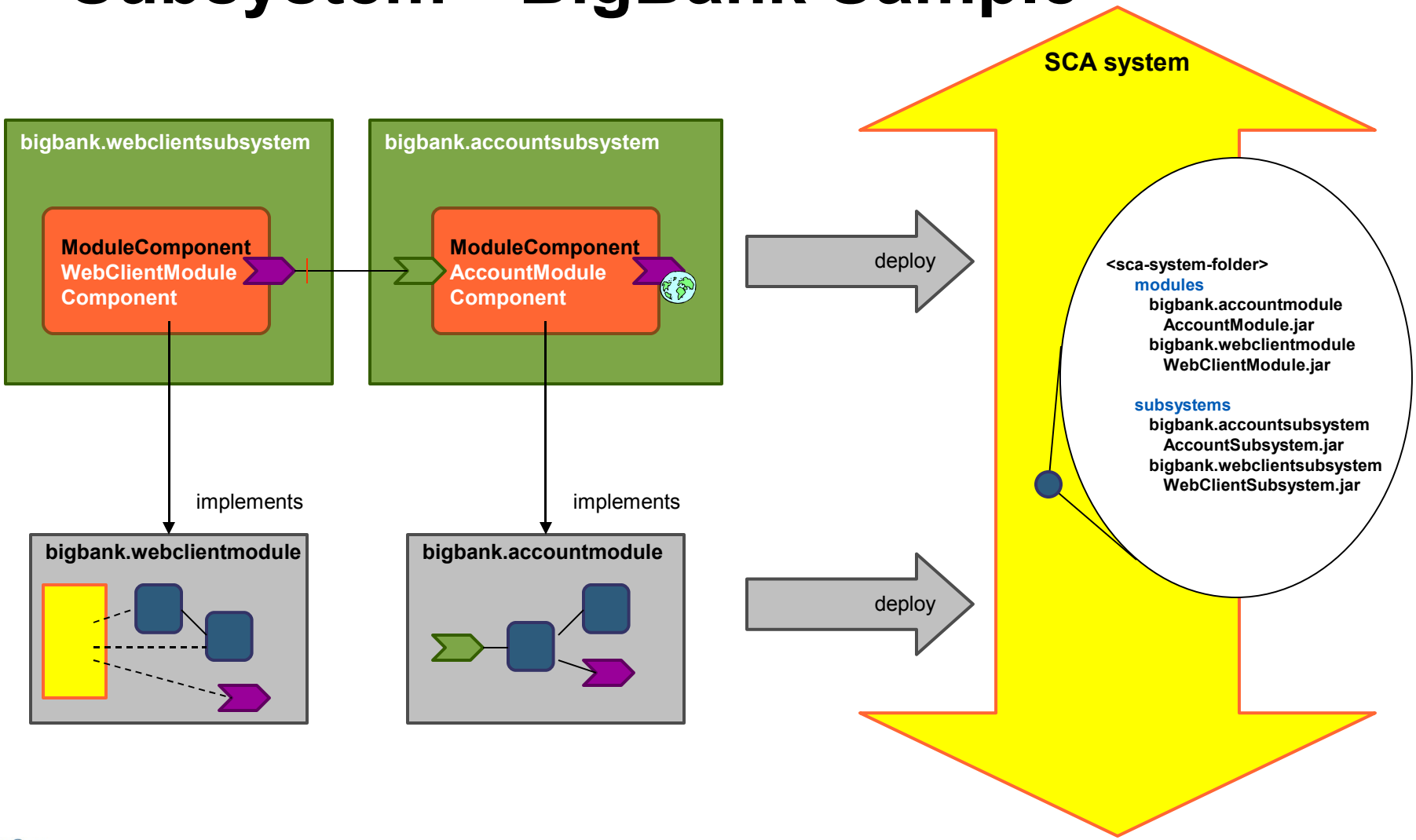
Modularity


Service
Reference
Module A
Entry Point
Component A
Component B
External Service
Wire
Wire
Wire
Binding
Implementation
Binding

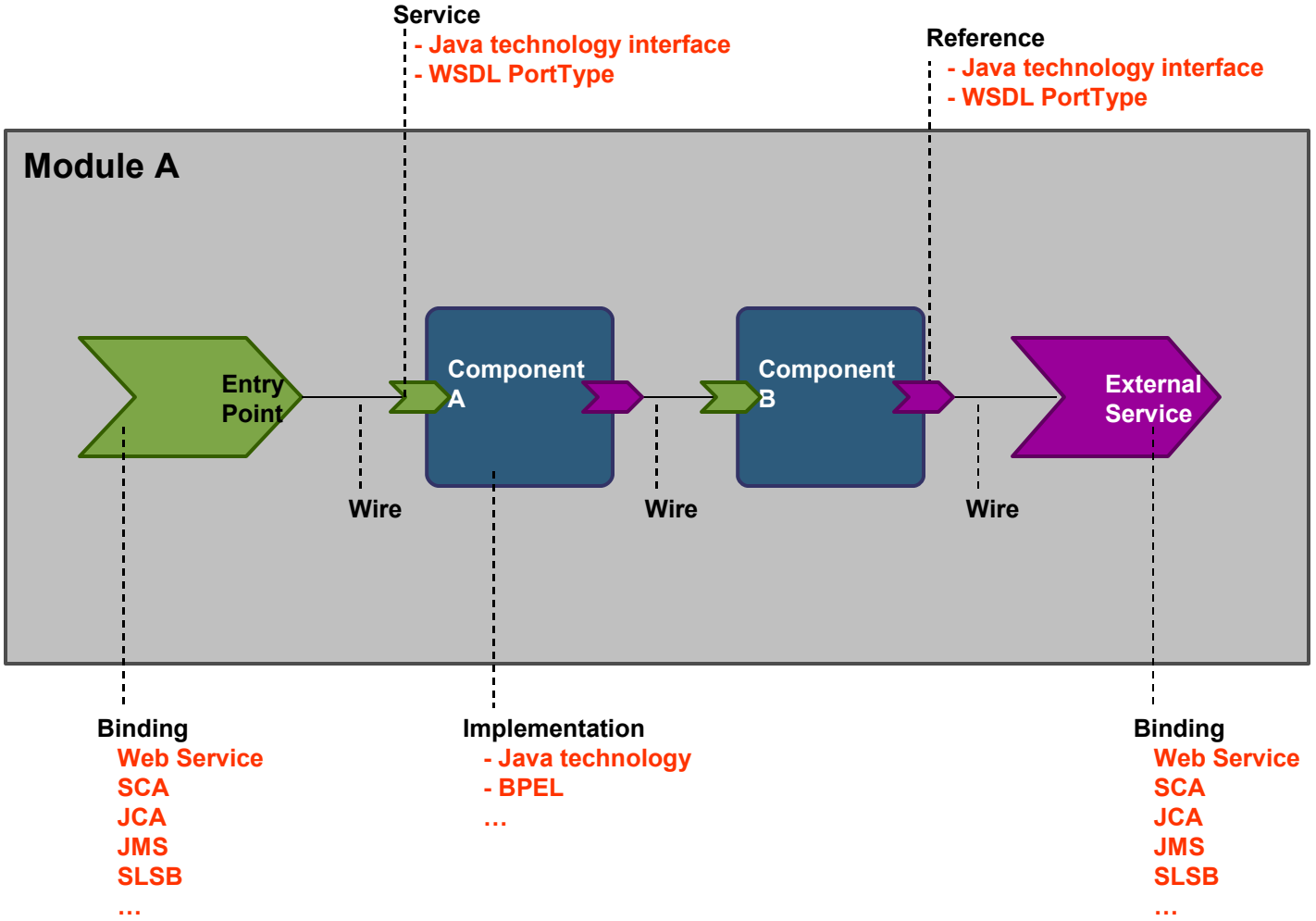
System and Subsystem



Subsystem—BigBank Sample

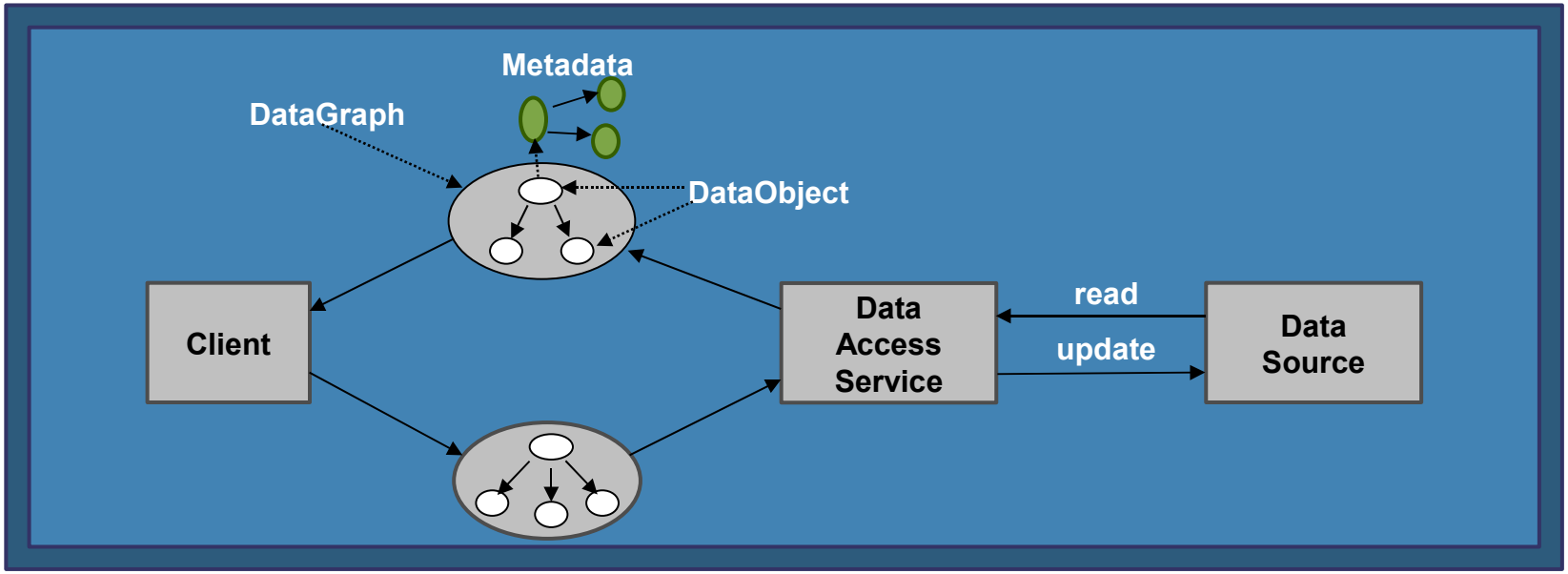


Implementation

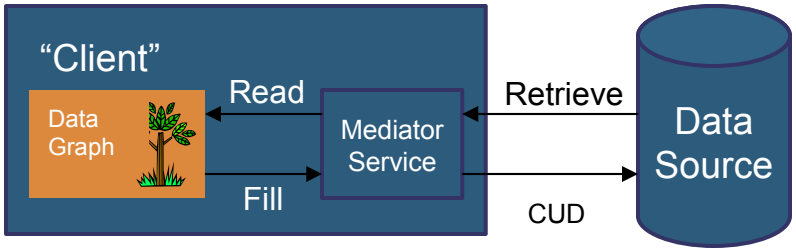


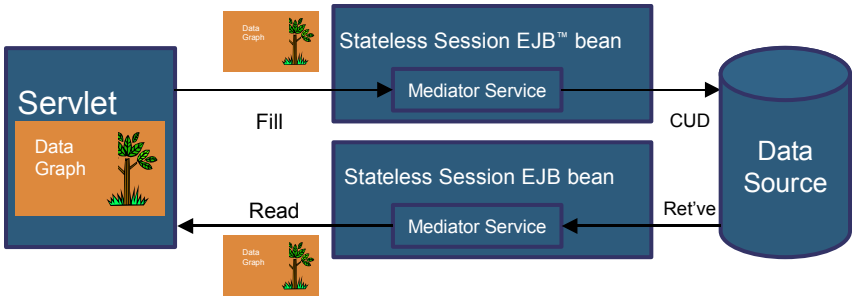
Service Data Objects

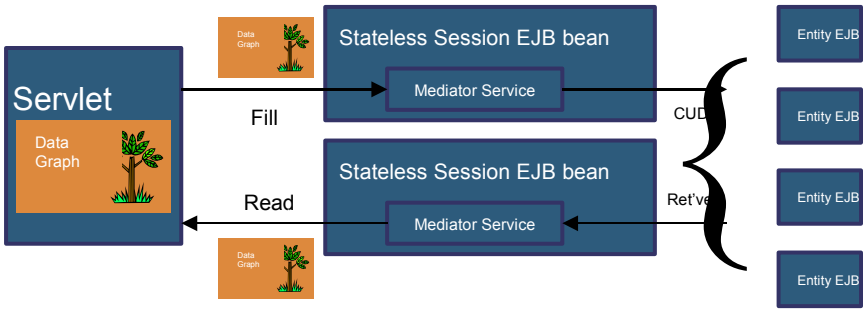
- Data is stored in a disconnected, source-independent format defined by the DataObject
  - Stored in a graph
  - Provides both dynamic loosely-typed and static strongly-typed interfaces to the data
    - getAddress()
    - getString("Address")
- DataGraph holds the root data object
  - Remembers change history
  - Provides Access to metadata about the DataObjects
- Data Access Service is responsible for filling graph of DataObjects from data source, updating data source from DataObject changes

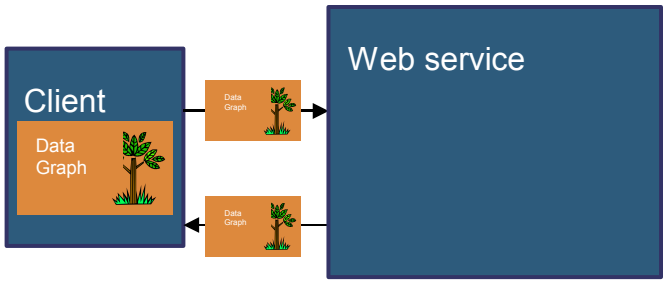


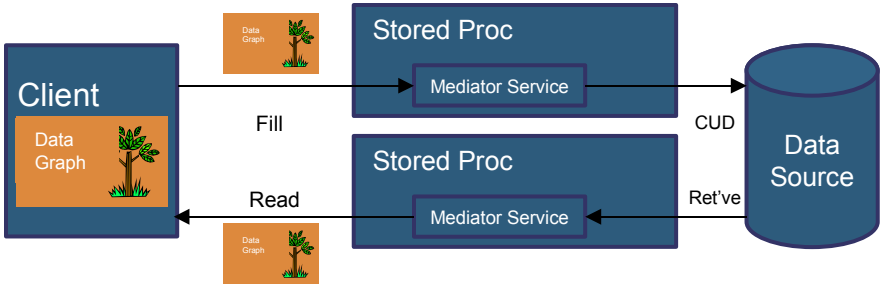
SDO Topologies



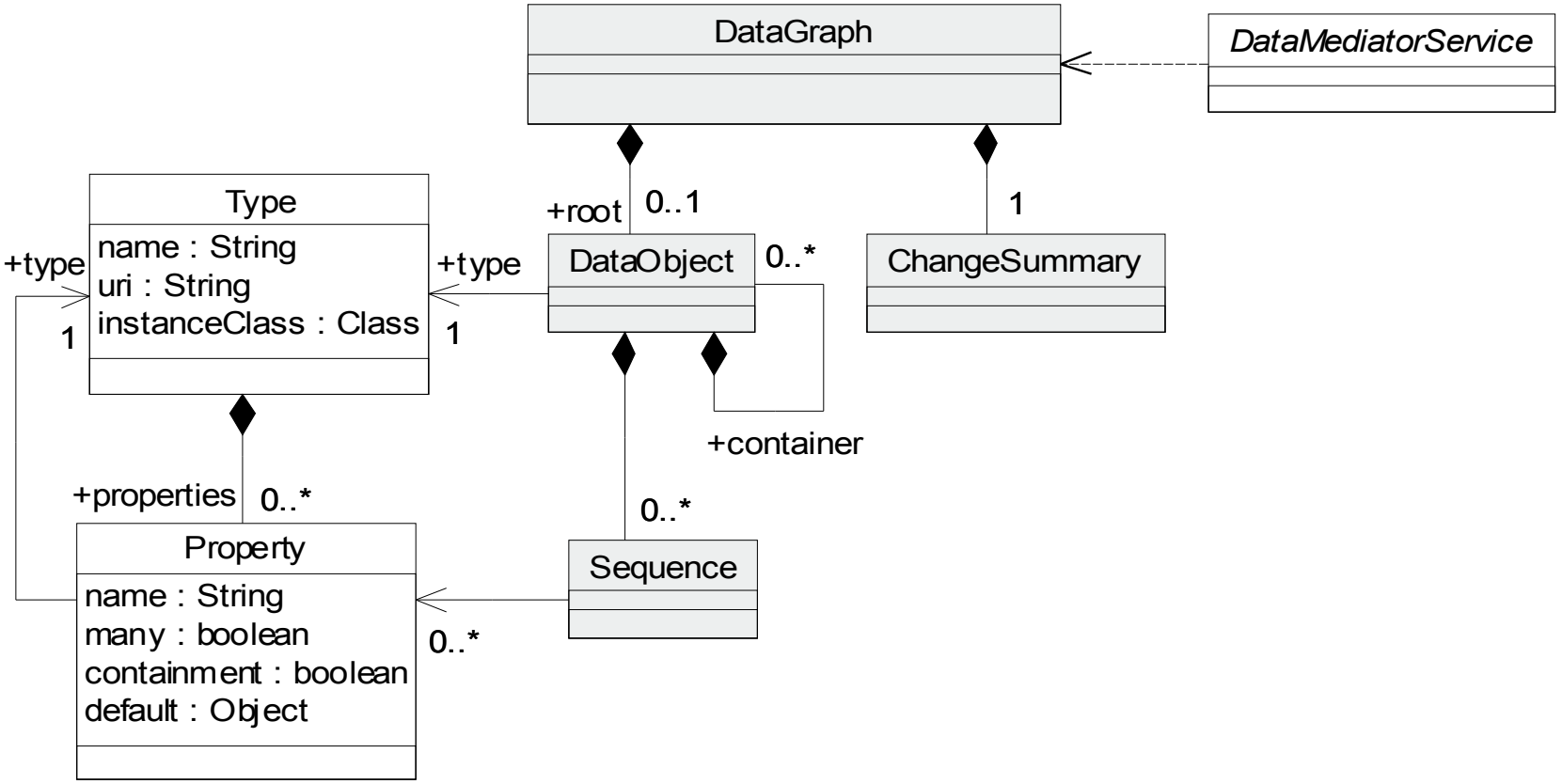








UML of SDO Classes



Service Development Languages

- Services will be derived from many different sources
- SOA explicitly presumes to embrace business application function where it already exists
  - Businesses want to compose business services, they don't want to be constrained by the technology choices made in the past
  - Java language, COBOL, C++, WSBPEL, XML, …, are all legitimate sources of business function
- The programming model for SOA needs to allow the use of an extensible set of implementation and composition languages and technologies

Implementation—Java Technology


```
public class AccountServiceImpl implements AccountService {

    @Property
    private String currency = "USD";

    @Reference
    private AccountDataService accountDataService;

    @Reference
    private StockQuoteService stockQuoteService;

    public AccountReport getAccountReport(String customerID) {

            DataFactory dataFactory = DataFactory.INSTANCE;
            AccountReport accountReport = (AccountReport)dataFactory.create(AccountReport.class);
            List accountSummaries = accountReport.getAccountSummaries();

            CheckingAccount checkingAccount = accountDataService.getCheckingAccount(customerID);
            AccountSummary checkingAccountSummary = (AccountSummary)dataFactory.create(AccountSummary.class);
            checkingAccountSummary.setAccountNumber(checkingAccount.getAccountNumber());
            checkingAccountSummary.setAccountType("checking");
            checkingAccountSummary.setBalance(fromUSDollarToCurrency(checkingAccount.getBalance()));
            accountSummaries.add(checkingAccountSummary);

            …

            return accountReport;
    }

    private float fromUSDollarToCurrency(float value){
            if (currency.equals("USD")) return value; else
            if (currency.equals("EURO")) return value * 0.8f; else
            return 0.0f;
    }
}
```

Implementation—C++

Service interface

```
// LoanService interface
class LoanService {
public:
        virtual bool approveLoan(unsigned long customerNumber,
                                 unsigned long loanAmount)
= 0;
};
```

Implementation declaration header file

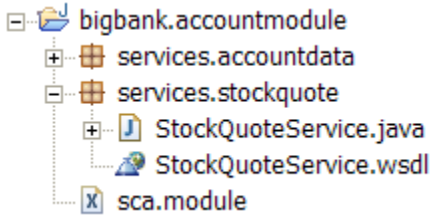
```
class LoanServiceImpl : public LoanService
{
public:
        LoanServiceImpl();
        virtual ~LoanServiceImpl();
        virtual bool approveLoan(unsigned long customerNumber,
        unsigned long loanAmount);
};
```

Implementation

```
#include "LoanServiceImpl.h"
LoanServiceImpl::LoanServiceImpl()
{
      …
}

LoanServiceImpl::~LoanServiceImpl()
{
      …
}

bool LoanServiceImpl::approveLoan(
                        unsigned long customerNumber,
                        unsigned long loanAmount)
{
      …
}
```

Client—C++

```
#include "ComponentContext.h"
#include "CustomerService.h"

using namespace osoa::sca;

void clientMethod()
{
    unsigned long customerNumber = 1234;
    …
    ComponentContext context = ComponentContext::getCurrent();
    CustomerService* service =
    (CustomerService* )context.getService("customerService");
    short rating = service->getCreditRating(customerNumber);
}
```



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™ bean, Java Message Service (JMS), J2EE Connector Architecture
  - Binding type extensibility
  - Overridable (no, may, must)

```
<?xml version="1.0" encoding="ASCII"?>
<module    xmlns="http://www.osoa.org/xmlns/sca/0.9"
           xmlns:v="http://www.osoa.org/xmlns/sca/values/0.9"
   name="bigbank.accountmodule" >

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

ExternalService StockQuote Service

SOA and Web 2.0

- SOA enables a systematic approach to dynamic composition of services as a response to business design requirements
- Web 2.0 expresses a desire to enable the truly ad hoc—the ability to do what I want and need to do without the constraints of a system of conformance
  - Web usage is increasingly oriented toward interaction and rudimentary social networks, which can serve content that exploits network effects with or without creating a visual, interactive web page*
- Are these competitive or complimentary?
- Consider: Every business, every commercial endeavor, has some aspects which are strategically important for which stability is critical to economic value, and has some aspects which are fundamentally dynamic and inherently dependent on being able to customize to the situation and leveraging the moment of potential arbitrage

* Source: Wikipedia

Useful Information

- Contacts
  - sharpc@uk.ibm.com
  - mike_edwards@uk.ibm.com
  - mrowley@bea.com
- SCA, SDO specs and related material
  - http://www.ibm.com/developerworks/webservices/library/specification/ws-sca/
  - http://www.ibm.com/developerworks/webservices/library/specification/ws-sdo/
- Apache “Tuscany” project
  - http://incubator.apache.org/tuscany
- Eclipse STP project
  - http://www.eclipse.org/stp/

Summary

- SOA is an architectural style designed specifically to better align IT and Business
- Loose-coupling is an inherent property of service oriented systems
- Composite applications are formed from heterogeneous services, derived from a variety of language and technologies, including Java
- SCA and SDO enable technology and language-neutral composition of services

Q&A

The SOA Programming Model

Rob High, Jr.—IBM
Ed Cobb—BEA
Sanjay Patil—SAP
Greg Pavlik—Oracle

SCA Collaboration Team, www.osoa.org

TS-3608