



Information and
Communication Systems
Research Group



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

7 WS Coordination WS Transactions

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
<http://www.iks.inf.ethz.ch/>



*Information and
Communication Systems
Research Group*



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

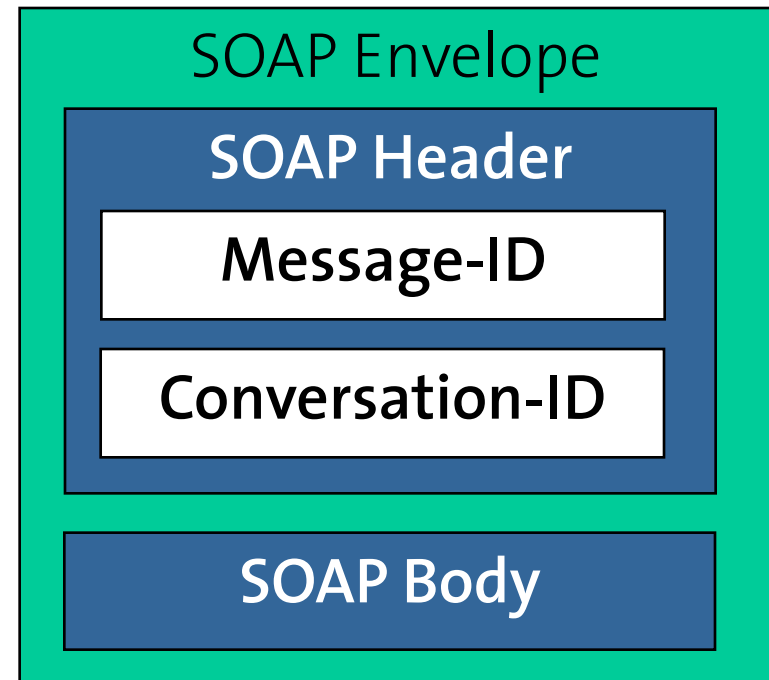
The Problem of Service Coordination

Standardizing Coordination Protocols

- SOAP has been a successful standard fostering the interoperability at the messaging layer. With it, messages can be exchanged regardless of the transport.
- Although SOAP includes the notion of message header, it does not define how such header should be used by the messaging infrastructure.
- To achieve the transparent **conversation routing** of messages, there is the need for a standardized way to generate unique conversation identifiers and store such information in the SOAP message headers.
- WSDL describes the interface of a service by only listing the operations it provides in terms of the structure of the messages that can be sent or received.
- To achieve transparent protocol compliance, a standard is needed to define what are the valid conversations supported by the service interface.
- Additionally, a meta-protocol standard is missing for two parties to agree on the set of supported protocols and how the infrastructure should coordinate them.
- Finally, standards should also cover the handling of concrete protocols (e.g., 2PC, or reliable message delivery)

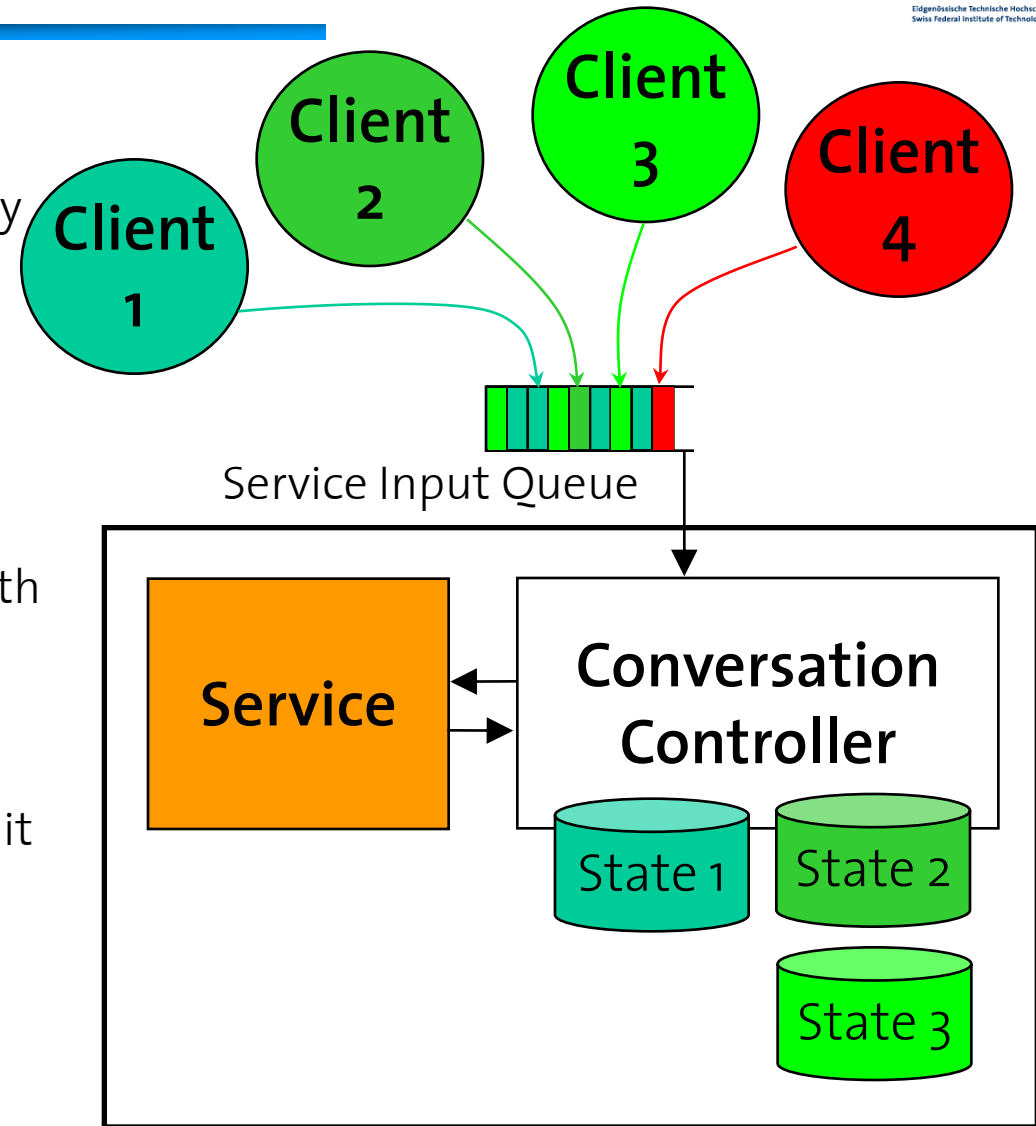
Message Correlation

- In order to support several concurrent conversations, the Web service/messaging infrastructure must define a way to uniquely identify messages and to determine which messages belong to a given conversation.
- The problem consists of reconstructing the session (and related context information) from a stateless, one-time message exchange.
- A similar problem exists with HTTP: how to establish and maintain session information across multiple request-response interactions. The standard solution is to use cookies, bits of information that persist across multiple request-response interactions.
- Reconstructing the context of a message can be done with different assumptions: Identifiers are stored in the header (correlation by the infrastructure) or the content of the message may allow the application to process it correctly (ad-hoc)



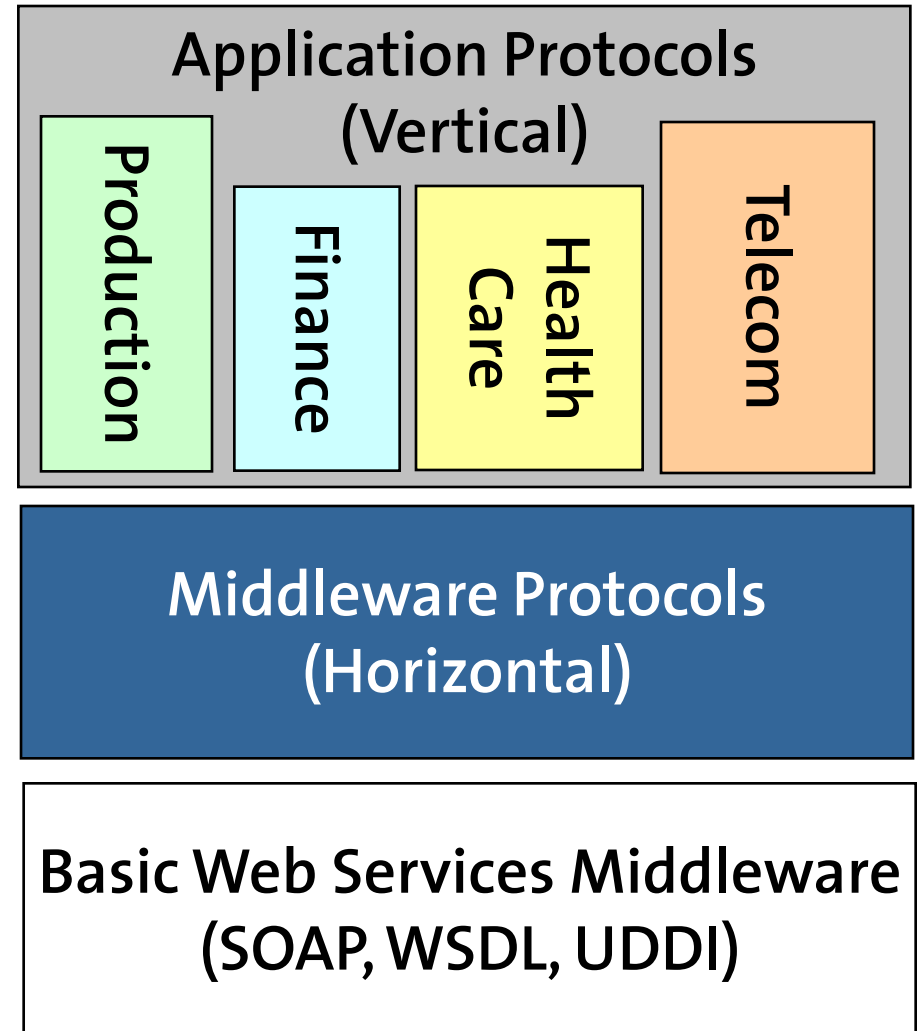
Conversation Controllers

- Conversation controllers are the key component of the coordination infrastructure built on existing reliable messaging middleware. They provide:
 - **Conversation routing.** Considering that there are multiple concurrent executions of a conversation, messages belonging to different conversations must be distinguished and correlated with the current state of the conversation.
 - **Protocol compliance.** Once a message is received, the controller must decide whether it should be accepted as it is expected as part of a conversation or rejected as it does not belong to any ongoing conversation



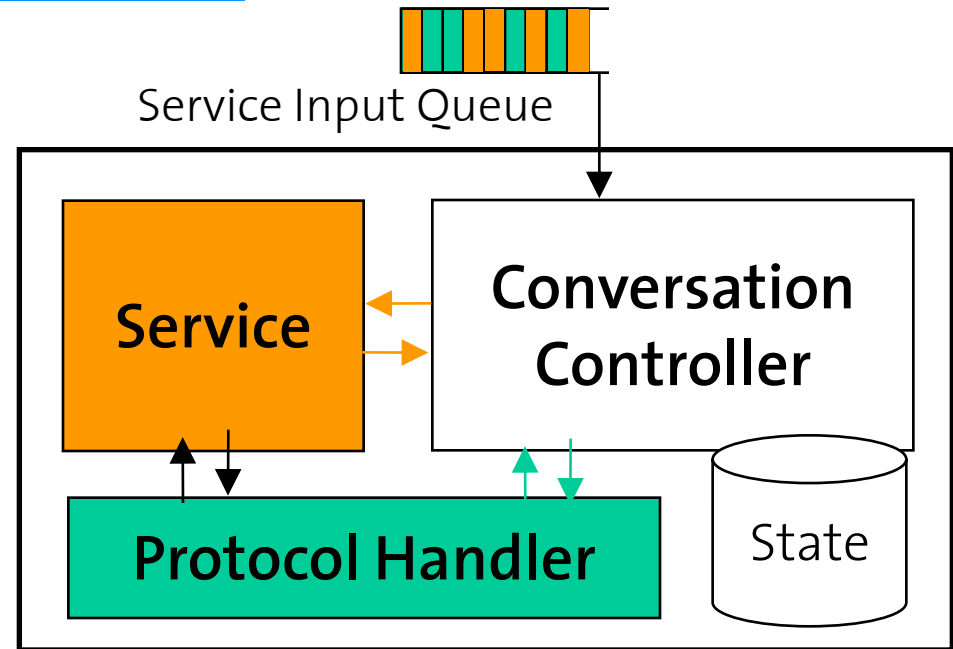
Horizontal and Vertical Protocols

- Horizontal protocols (or middleware protocols) define a common, application-independent infrastructure. Building on top of the basic messaging infrastructure, they extend it with additional properties (e.g., reliability, transactions, security)
- Vertical Protocols are used within a specific business area (e.g., manufacturing, health care, finance, telecommunication). They describe in detail how to conduct concrete business transactions, what are the documents involved and what is their format and semantics.
- Examples of vertical protocols are xCBL or RosettaNet. They can be seen as the XML evolution of previous ones such as EDI.



Transparent protocol handling

- In addition to stateful conversations, assuming the appropriate level of standardization, the coordination infrastructure can handle coordination protocols providing properties such as security, transactions and reliability.
- To do so, a protocol handling layer is added between the low-level messaging infrastructure and the implementation of a service.
- The protocol handler may be transparent and exchange messages automatically, without the intervention of the Web service implementation. For example, reliable message delivery can be implemented this way.



- Alternatively, the protocol handler provides a framework which is reused by the service implementation. For example, a generic 2PC transactional protocol handler may rely on the service to decide whether to commit or abort the transaction, while coordinating the delivery of the corresponding messages.



*Information and
Communication Systems
Research Group*



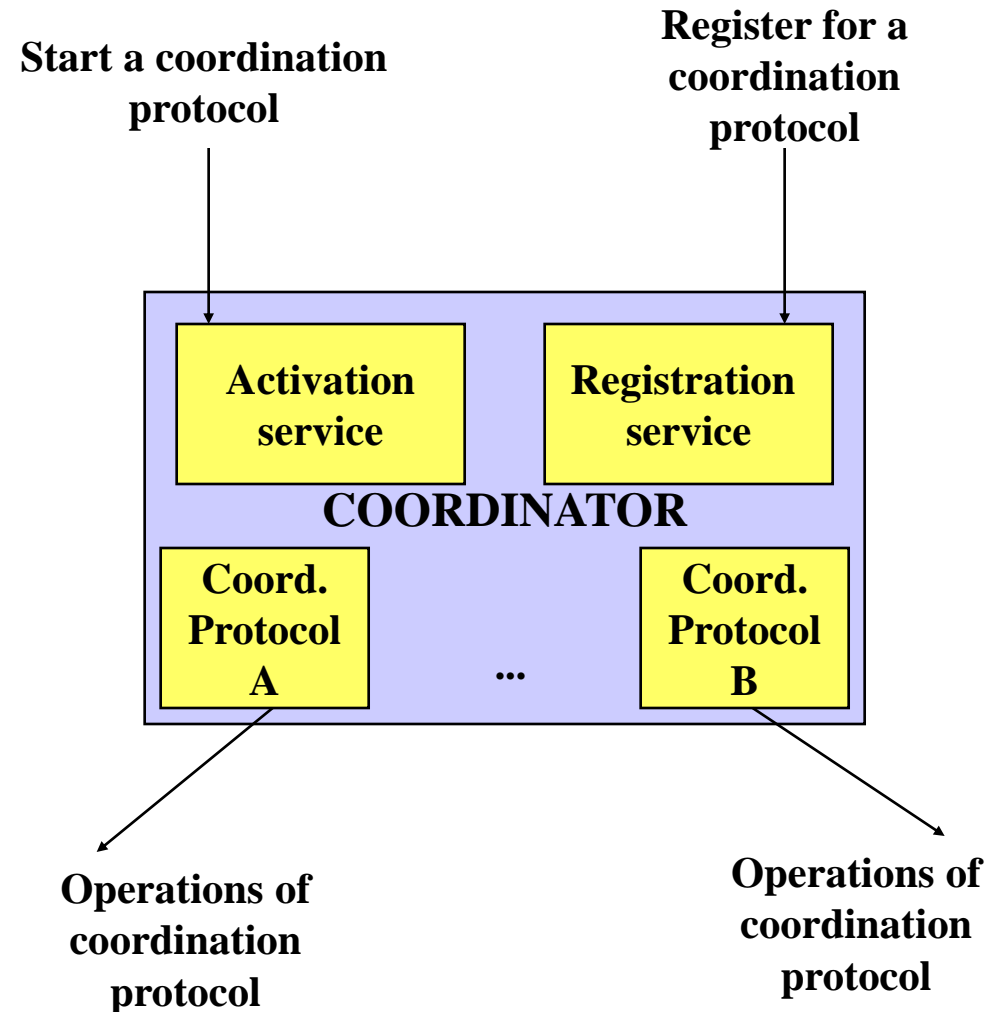
ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

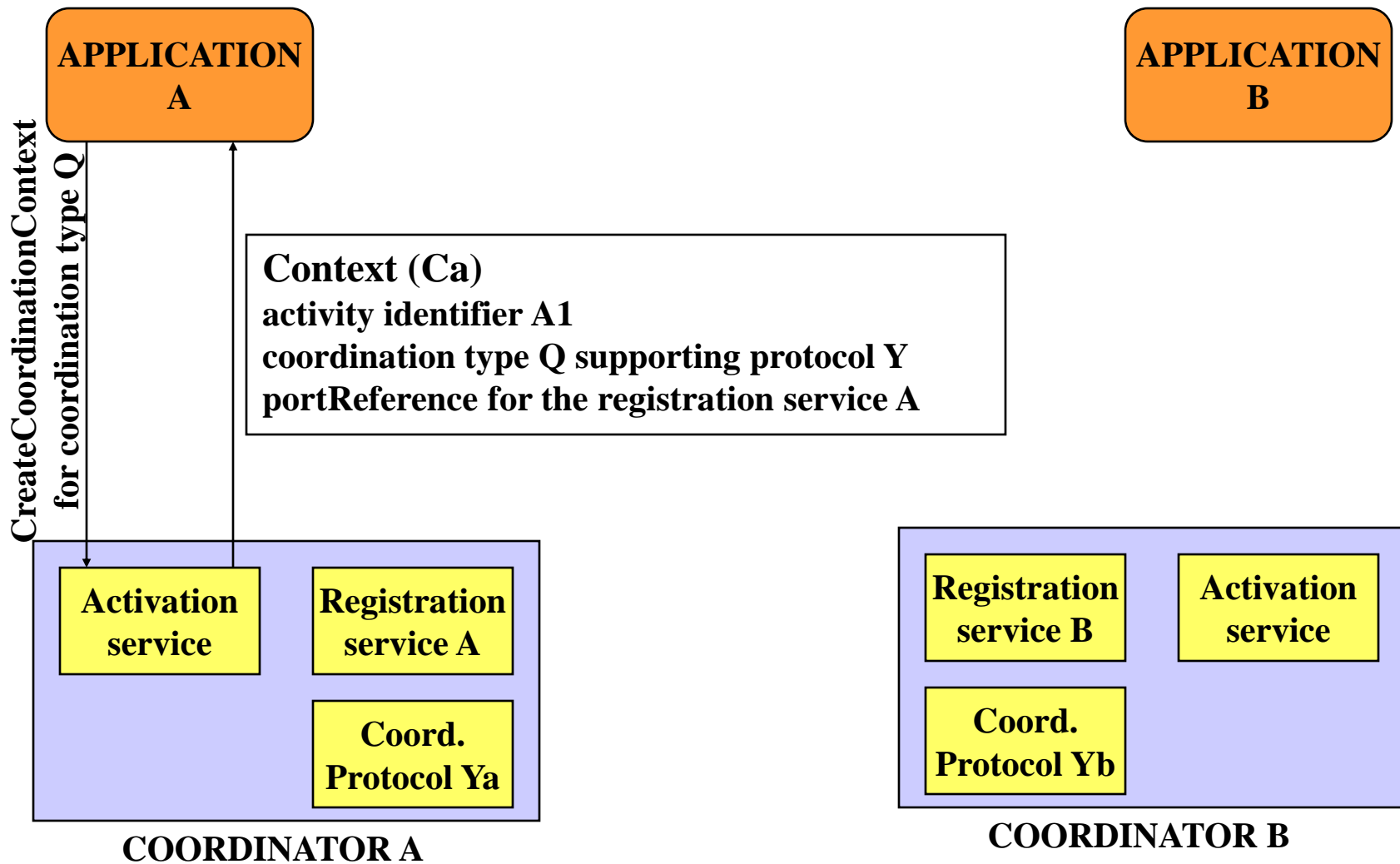
WS-Coordination

WS-Coordination

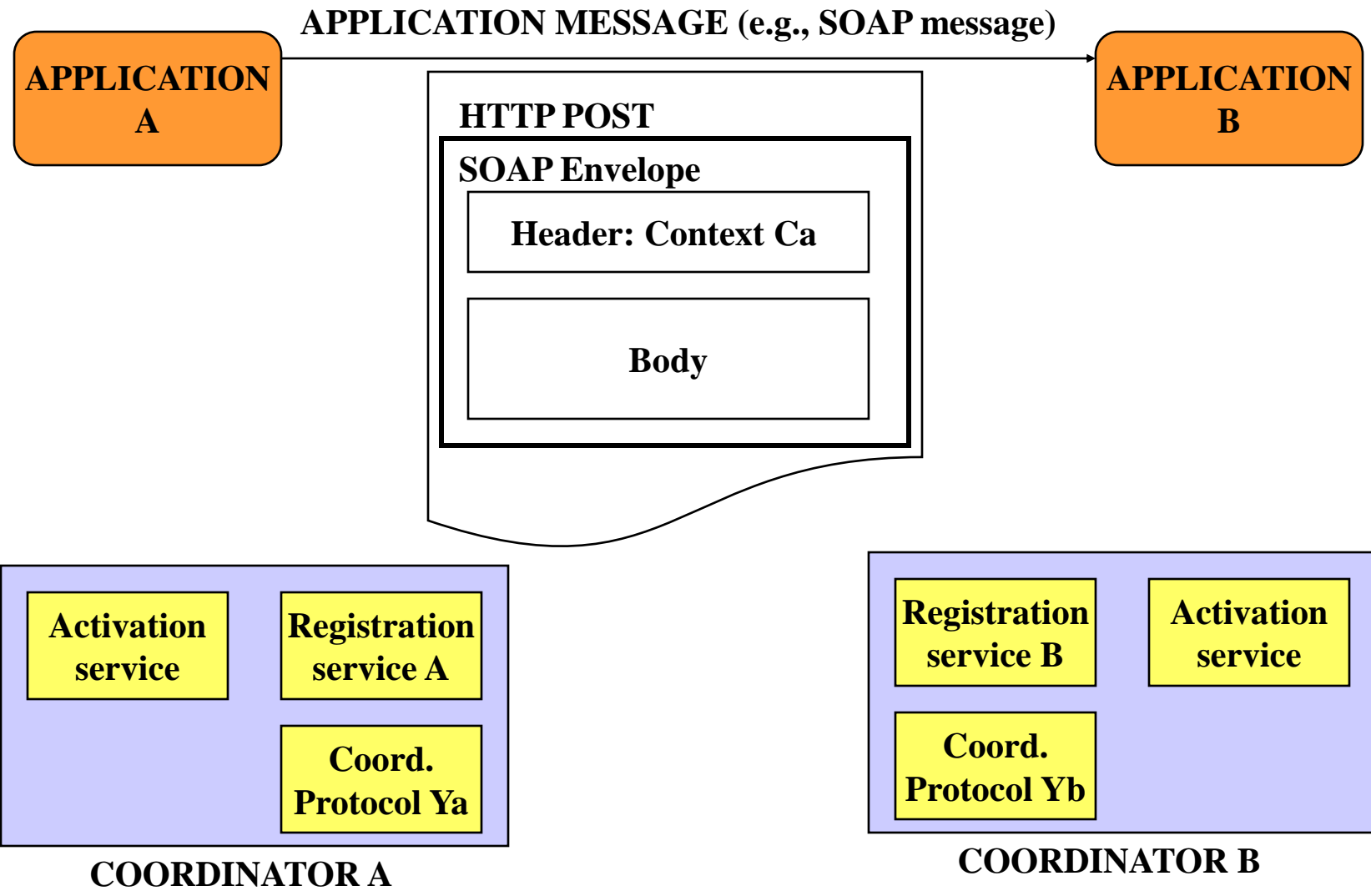
- WS-Coordination is intended as a generic infrastructure to implement coordination protocols between Web services
- Its main goal is to serve as a generic platform for implementing advanced transaction models but it can be used to implement a wide variety of coordination protocols between services (including some forms of conversations)
- WS-Coordination encompasses a set of behaviors and APIs which enable a module to extend Web services with coordination capabilities
- It mirrors the behavior of transactional services in conventional middleware platforms



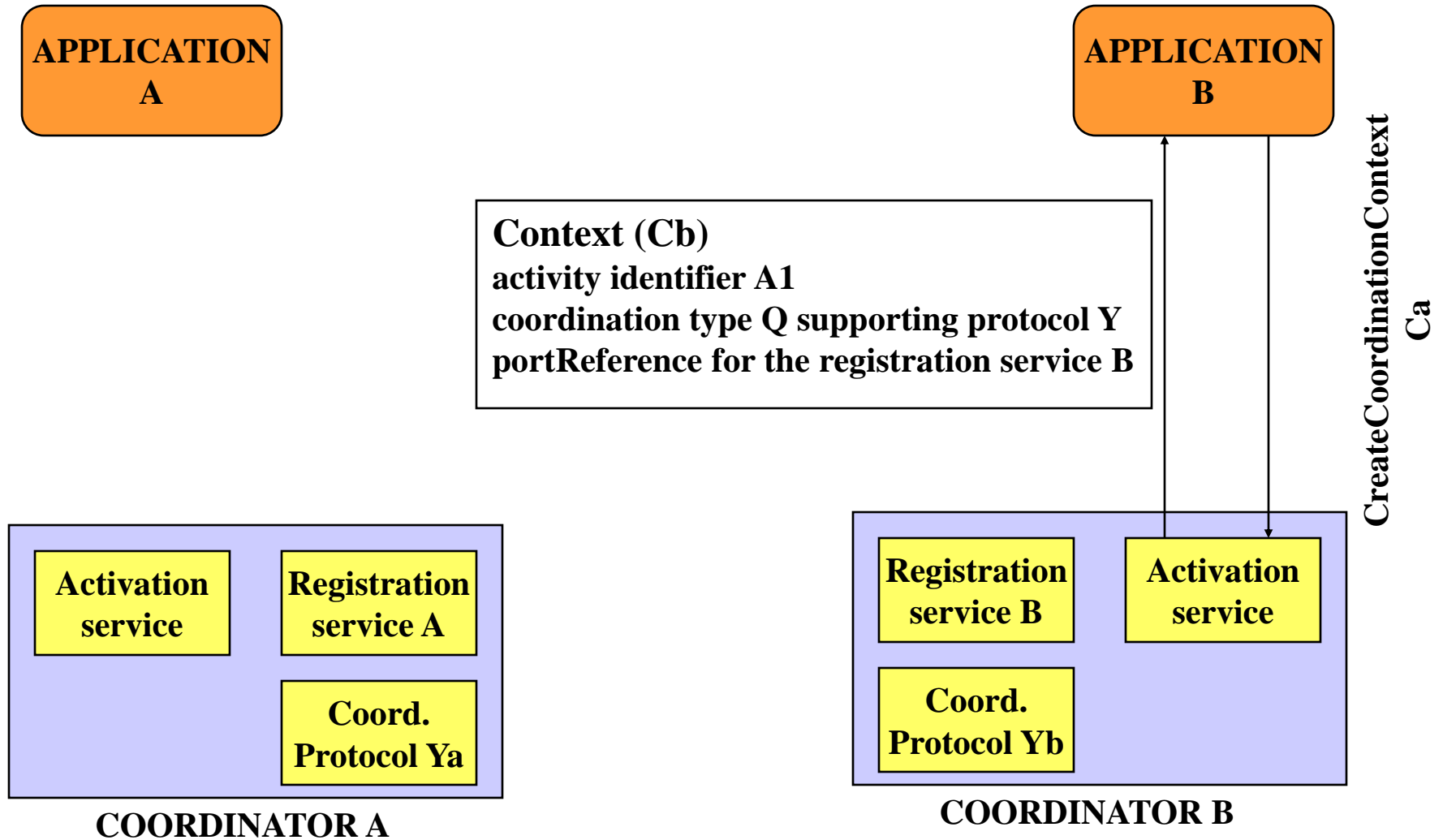
Basics of WS-Coordination (1)



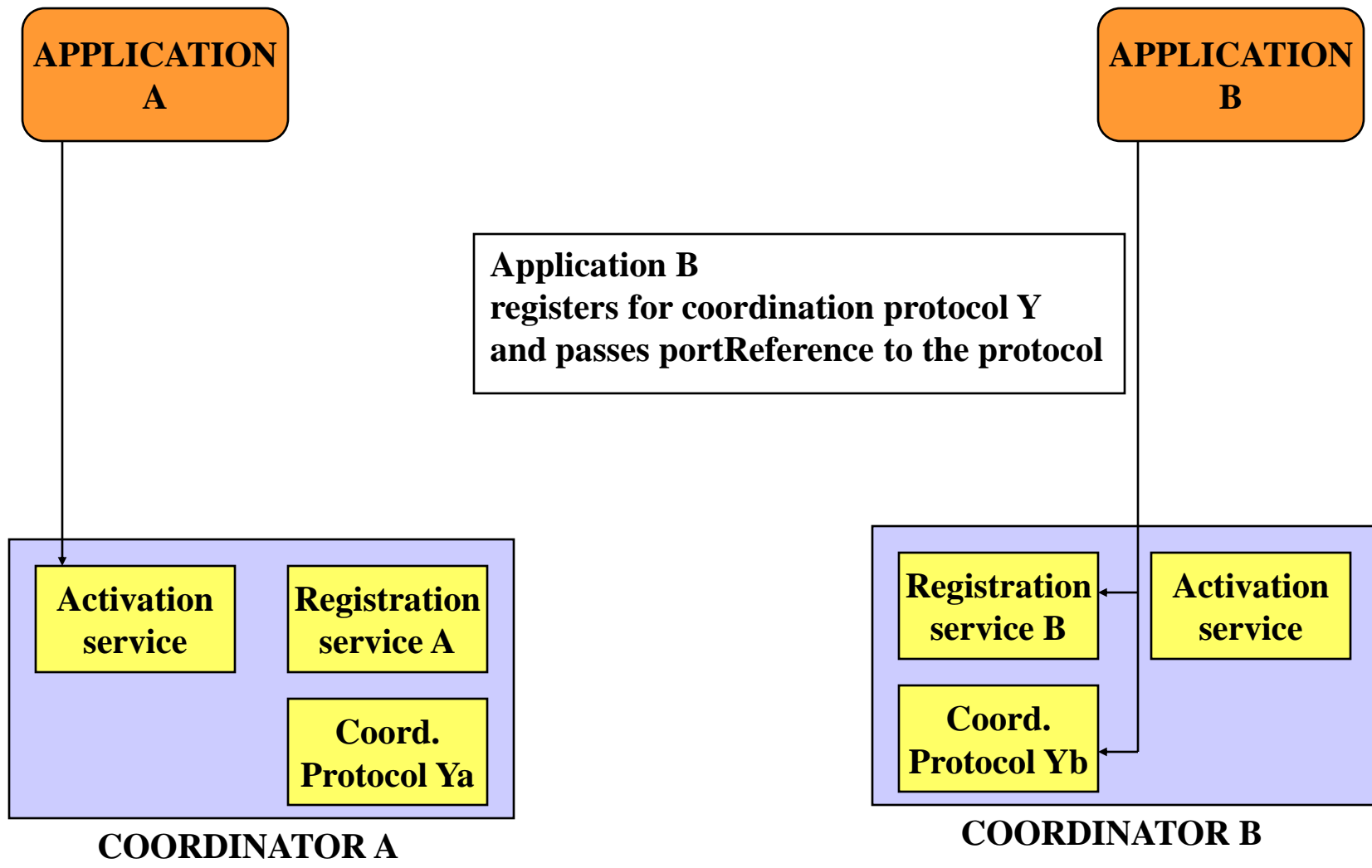
Basics of WS-Coordination (2)



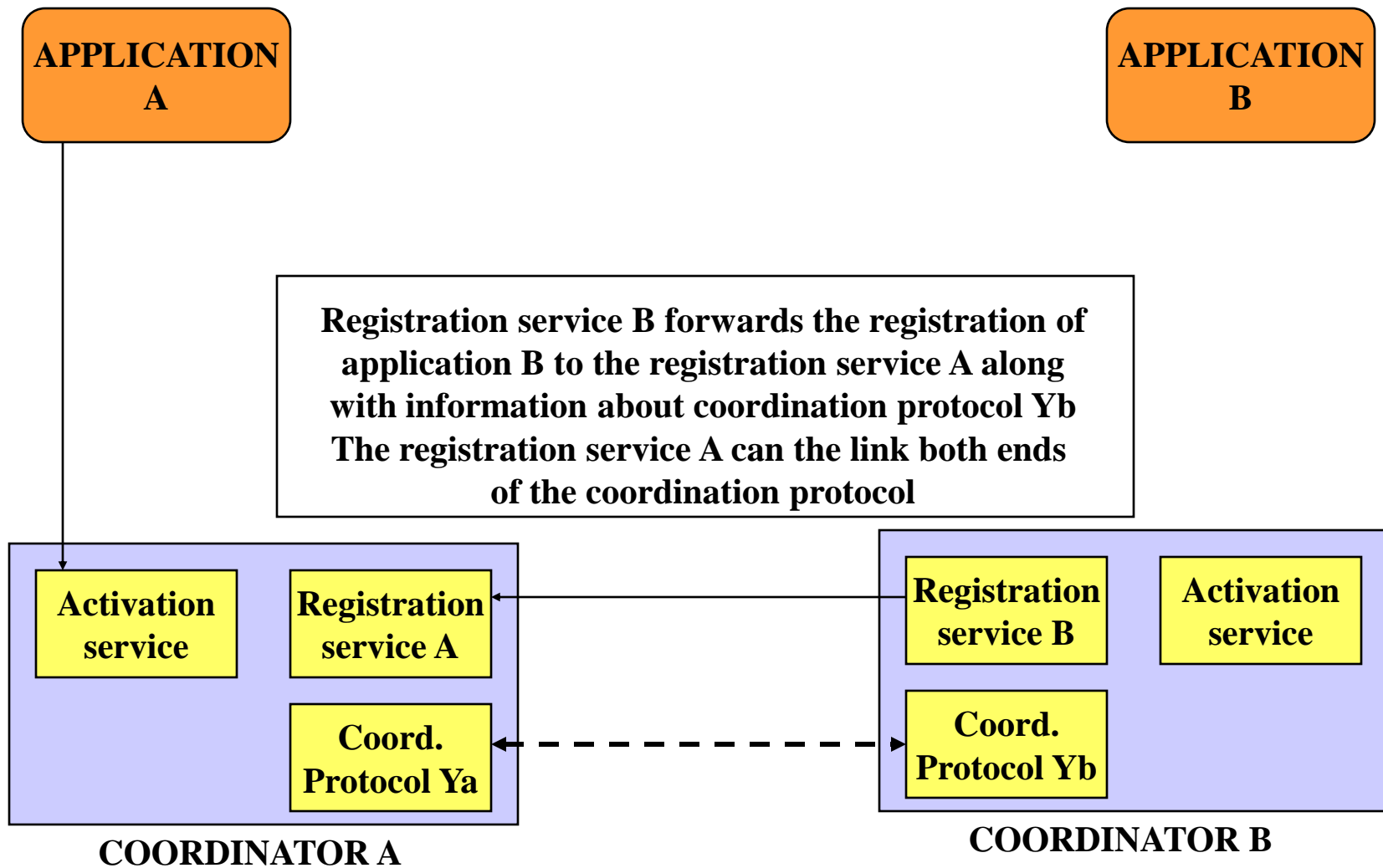
Basics of WS-Coordination (3)



Basics of WS-Coordination (4)



Basics of WS-Coordination (5)



Messages and Interfaces

- The coordinator defined by WS-Coordination is described using WSDL and offers a number of services to the application.
- The application accesses these services by sending, e.g., SOAP messages to the coordinator which then responds with new SOAP messages. Interactions with the protocol would then also be in terms of SOAP messages (but other protocols are possible, one simply needs to provide alternative bindings for the coordinator services)
- The example shown considers the case where application B decides to use its own coordinator. Application B could also decide to use the same coordinator as application A but in the cases where A and B are independent services, provided by different organizations, a coordinator per application makes more sense
- WS-Coordination is an attempt at standardizing:
 - the use of SOAP headers for coordination protocols
 - the basic operations for most coordination protocols
 - the functionality a Web service middleware platform must support for allowing coordination protocols to be implemented

WS-Coordinator in XML

ACTIVATION SERVICE:

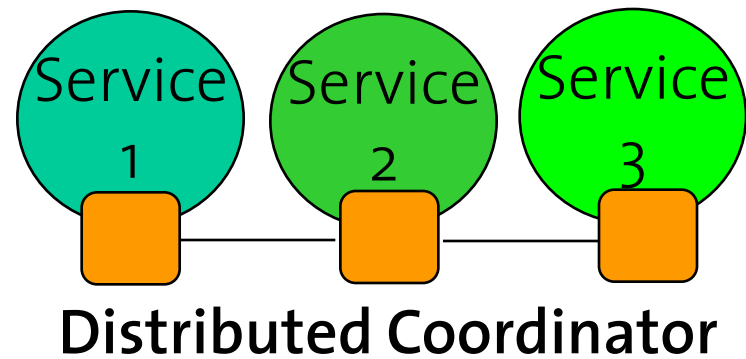
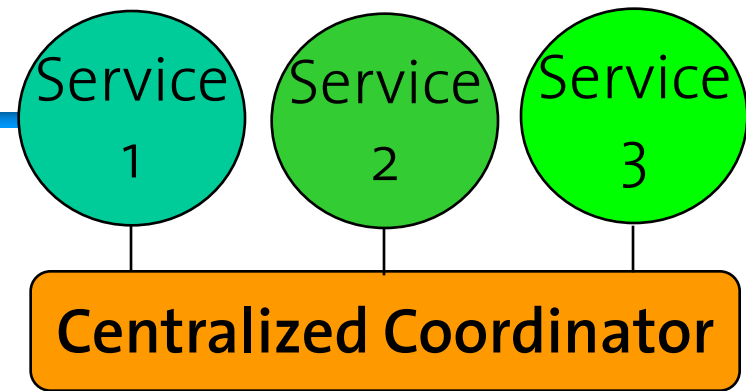
```
<wsdl:portType name="ActivationCoordinatorPortType">  
  <wsdl:operation name="CreateCoordinationContext">  
    <wsdl:input message="wscoor:CreateCoordinationContext"/>  
  </wsdl:operation>  
</wsdl:portType>
```

RESPONSE ACTIVATION SERVICE

```
<wsdl:portType name="ActivationRequesterPortType">  
  <wsdl:operation name="CreateCoordinationContextResponse">  
    <wsdl:input message="wscoor:CreateCoordinationContextResponse"/>  
  </wsdl:operation>  
  <wsdl:operation name="Error">  
    <wsdl:input message="wscoor:Error"/>  
  </wsdl:operation>  
</wsdl:portType>
```

Summary

- WS-Coordination is a standard coordination infrastructure proposed in August 2002
- It defines a **coordination context** to be included in the header of SOAP messages. The context stores unique conversation identifiers used for routing and protocol verification.
- It includes a solution for **registering** the protocol handlers of the Web services with the coordination infrastructure. With it, protocol handlers can be notified when specific steps of the protocol are carried out.
- It contains an **activation** interface, used to create a new coordination context and inform each peer about the role it should assume while running the protocol.



- WS-Coordination attempts to generalize existing middleware solutions (e.g., CORBA OTS) to provide a common foundation of specific forms of coordination
- *Note:* WS-Coordination is not a language for defining coordination protocols.



*Information and
Communication Systems
Research Group*



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

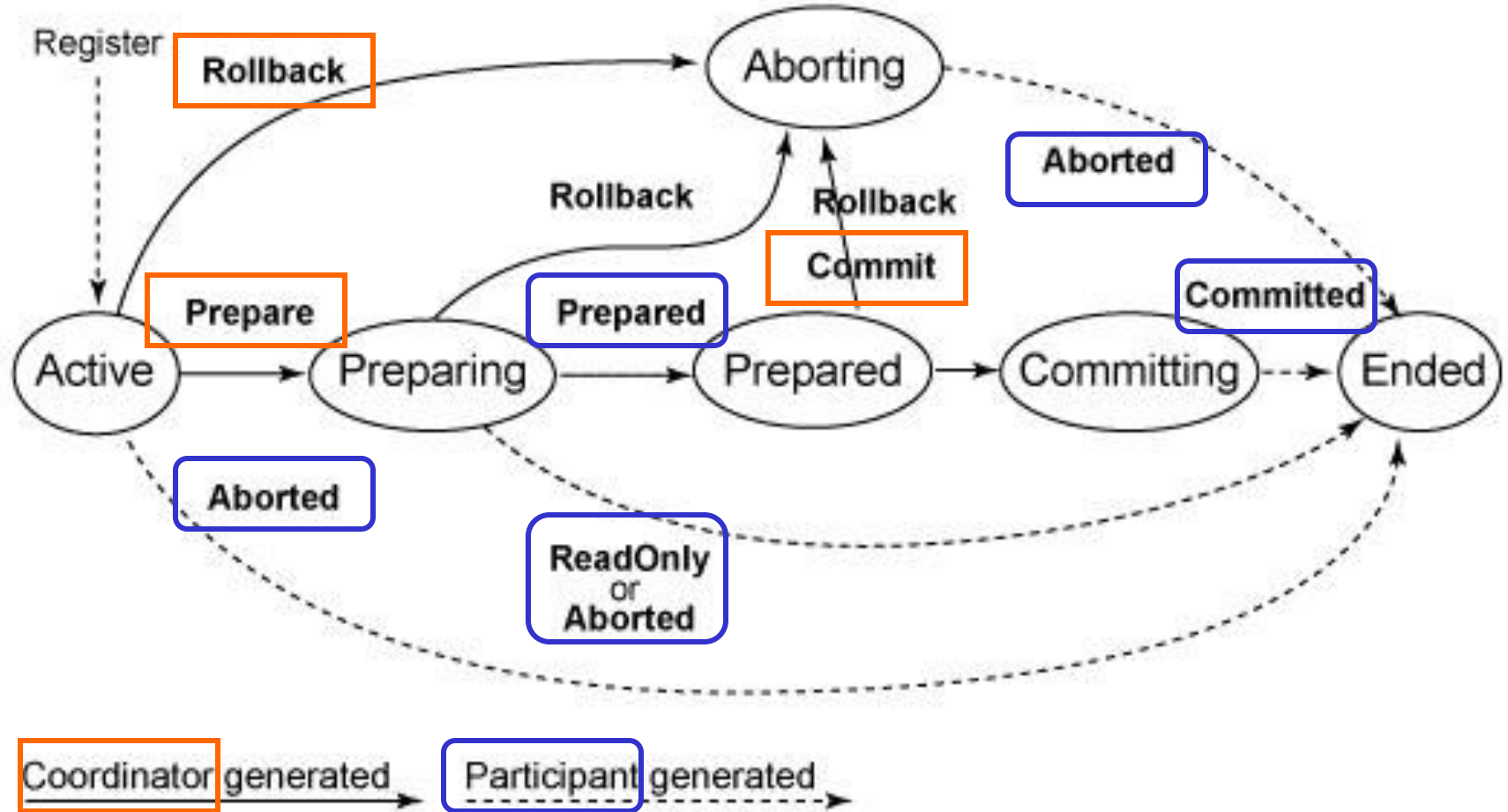
WS-Transactions

WS-Transactions

- WS-Transactions builds directly upon WS-Coordination to specify different coordination protocols related to transaction processing
 - atomic transactions (governed by 2 Phase Commit)
 - business activities (transactional but based on compensation activities)
 - business agreement
 - business agreement with complete
- WS-Transactions specify the coordination protocol to be used as part of WS-Coordination. The specification deals with the nature of the interaction, the syntax and semantics of the messages to be exchanged as part of the coordination protocol, and the expected responses of all participants involved
- Like WS-Coordination, WS-Transactions follows very closely the transactional model found in conventional middleware platforms

Coordination Protocol for 2PC

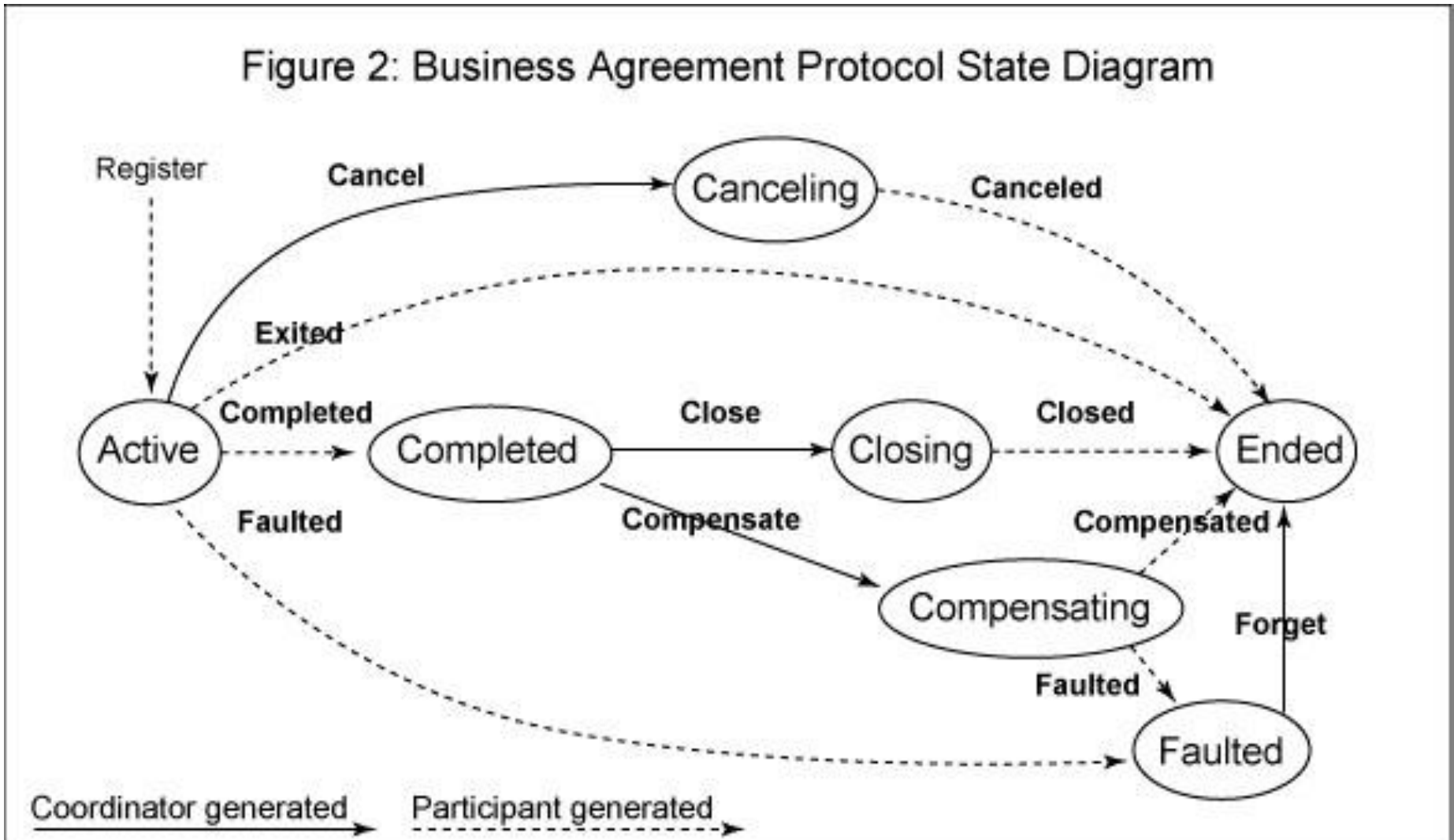
Figure 5: 2PC Protocol State Diagram



From Web Services Transaction (WS-Transaction) 9 August 2002

Business Agreement

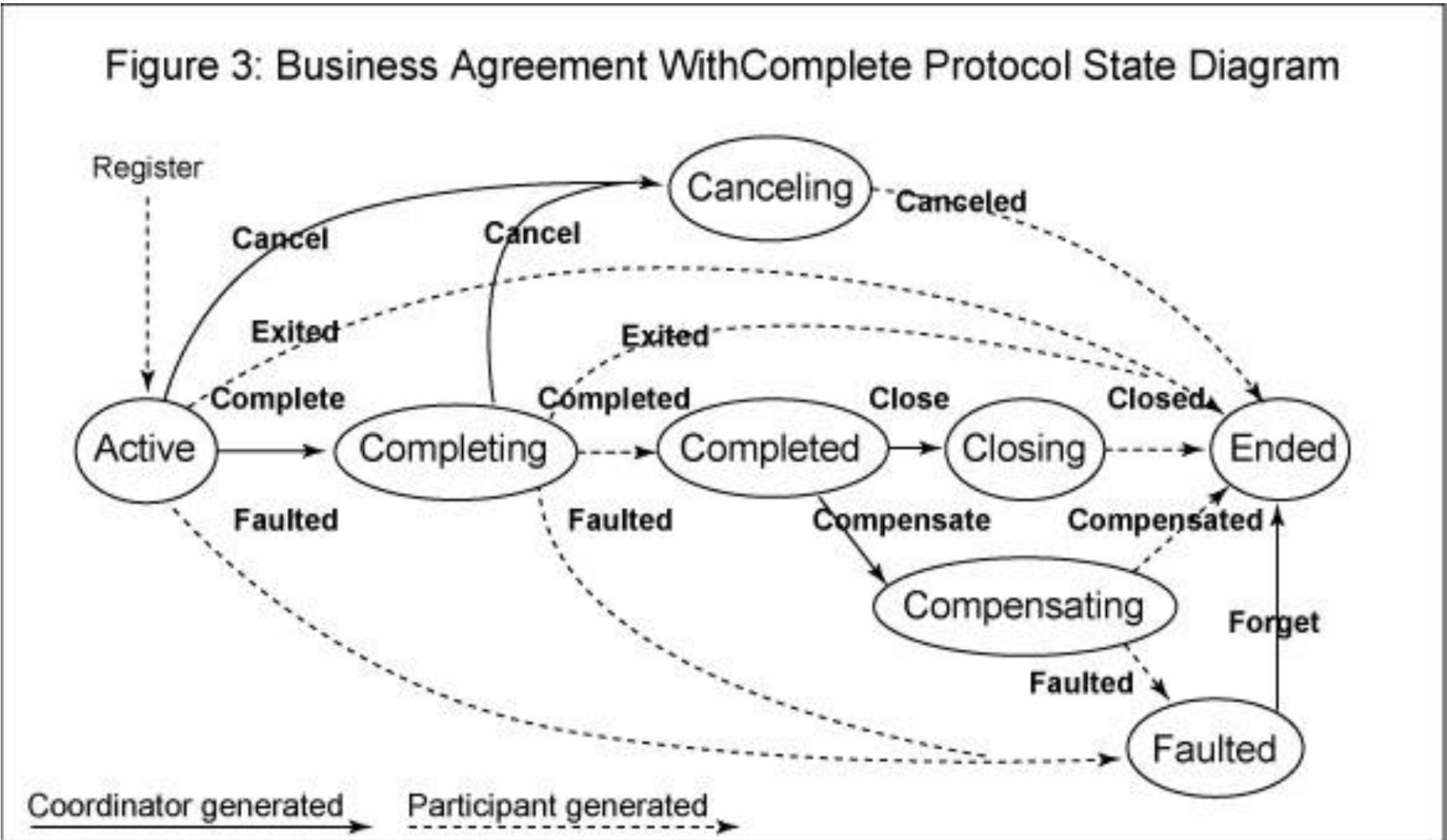
Figure 2: Business Agreement Protocol State Diagram



From Web Services Transaction (WS-Transaction) 9 August 2002

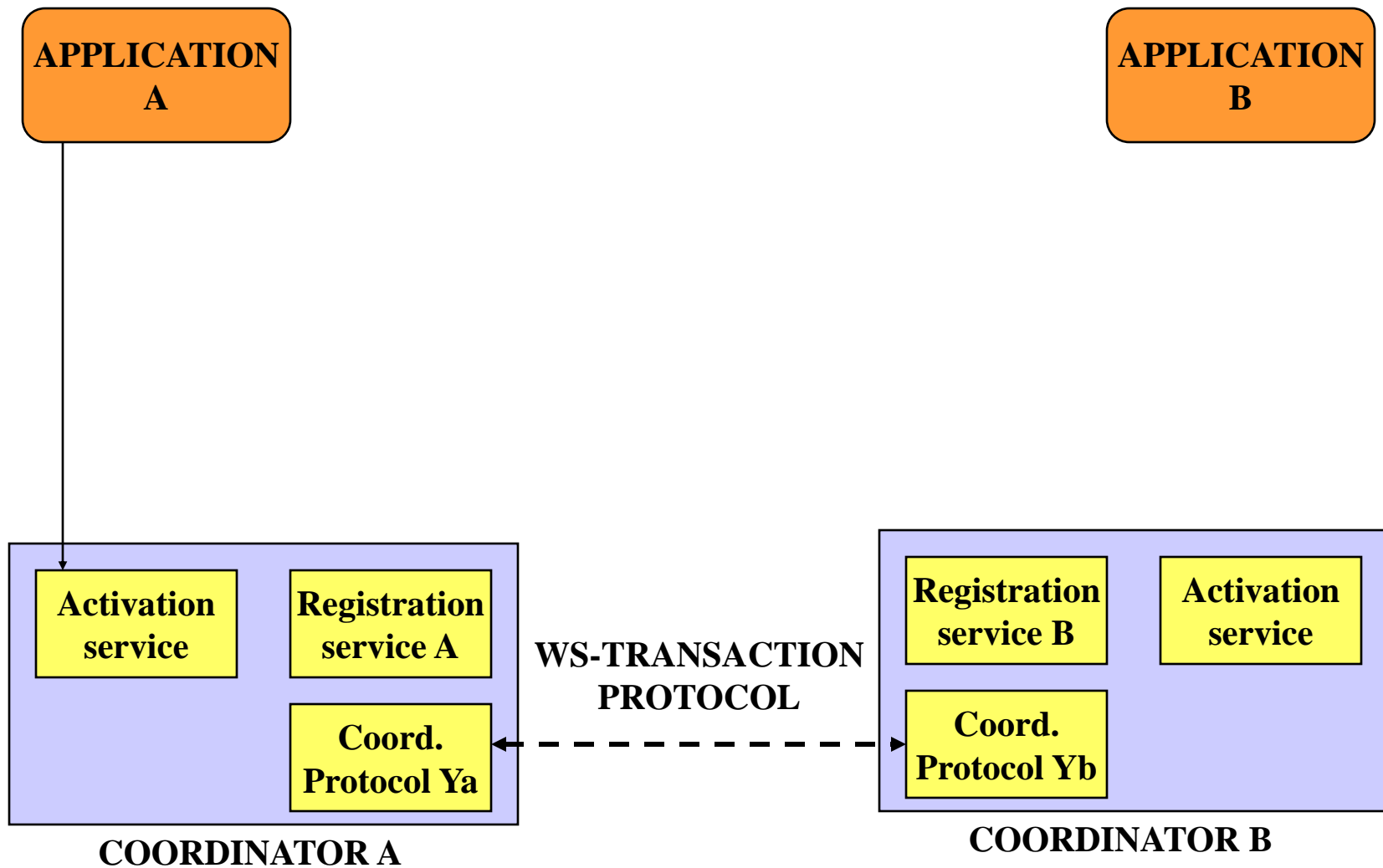
Business Agreement with Completion

Figure 3: Business Agreement With Complete Protocol State Diagram



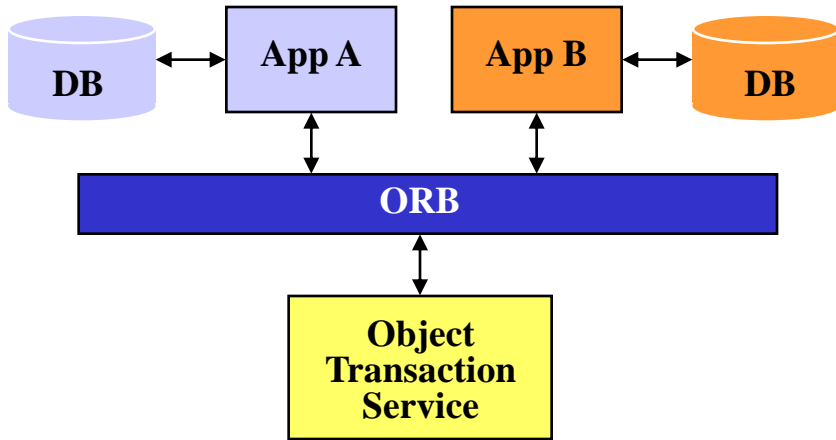
From Web Services Transaction (WS-Transaction) 9 August 2002

WS-Transactions

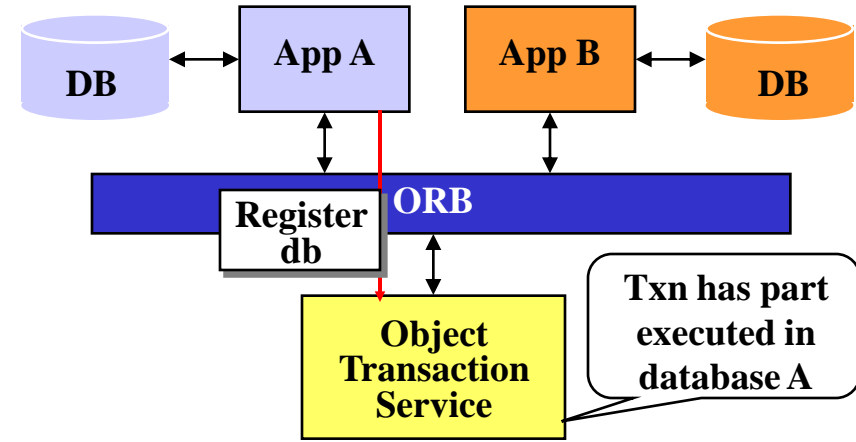


CORBA transactions (1)

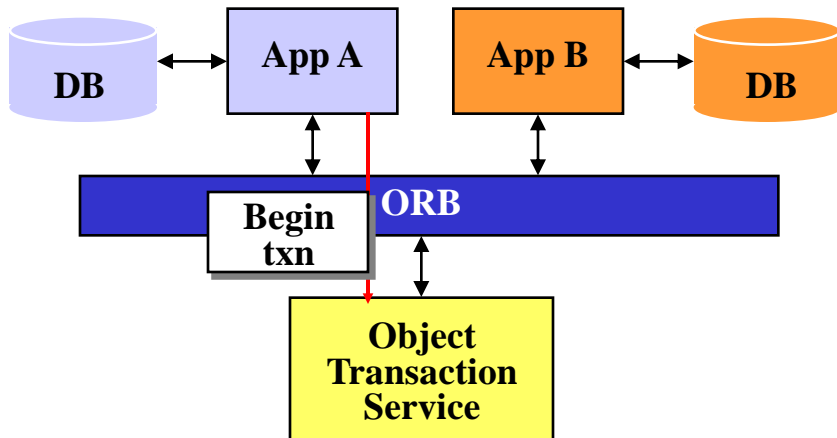
1) Assume App A wants to update databases A and B



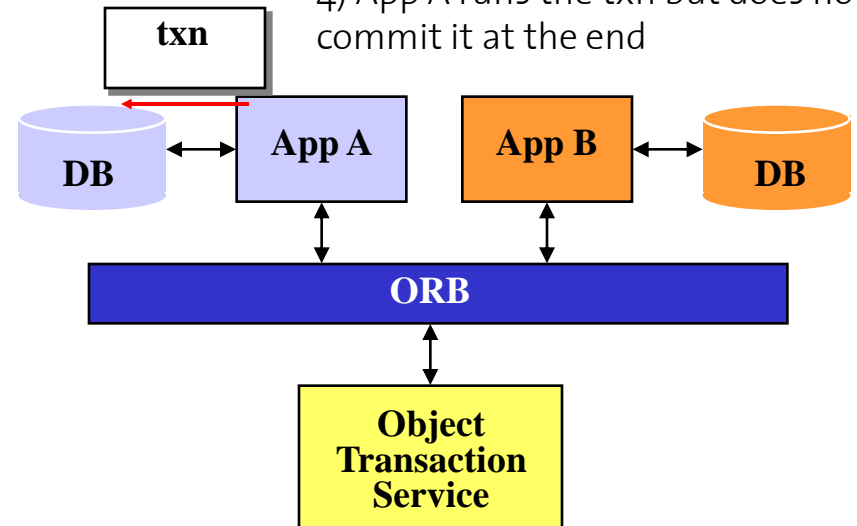
3) App A registers the database for that transaction



2) App A obtains a txn identifier for the operation

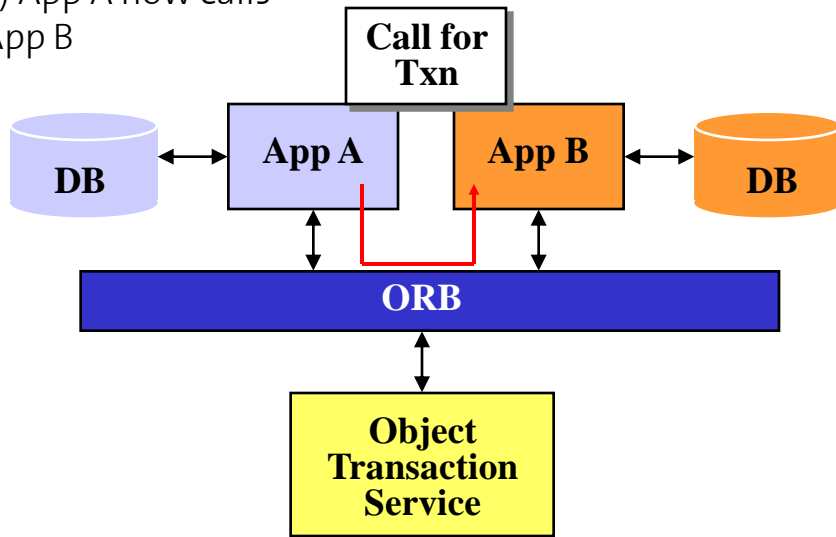


4) App A runs the txn but does not commit it at the end

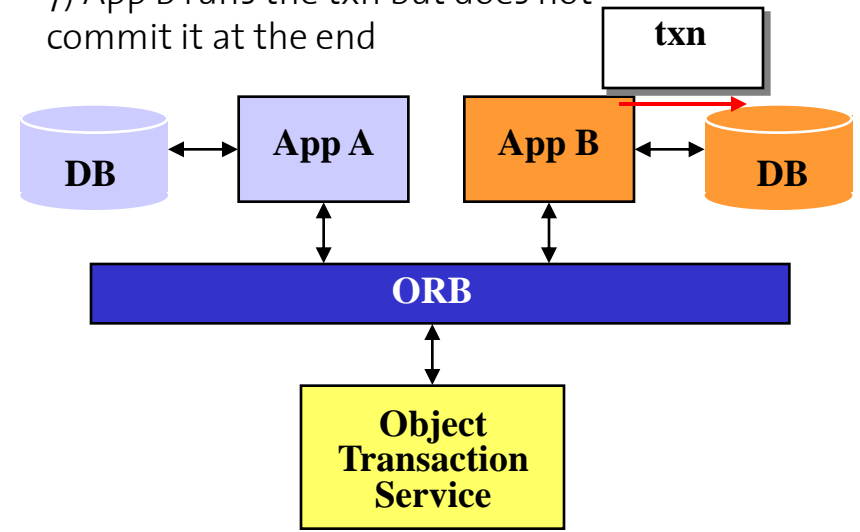


CORBA transactions (2)

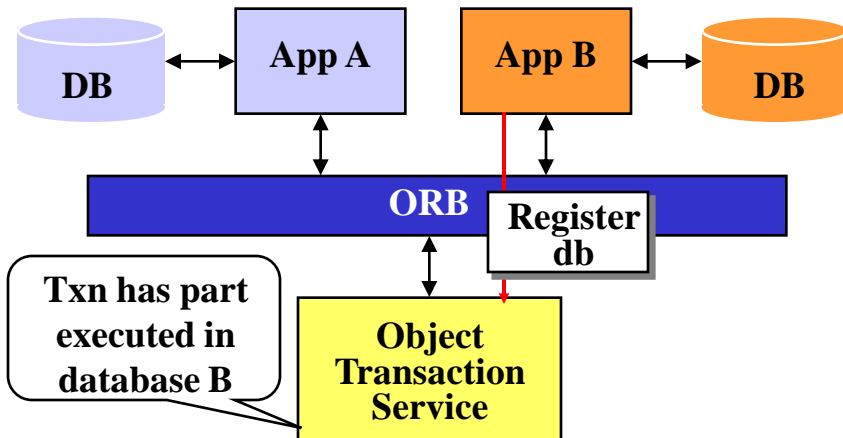
5) App A now calls App B



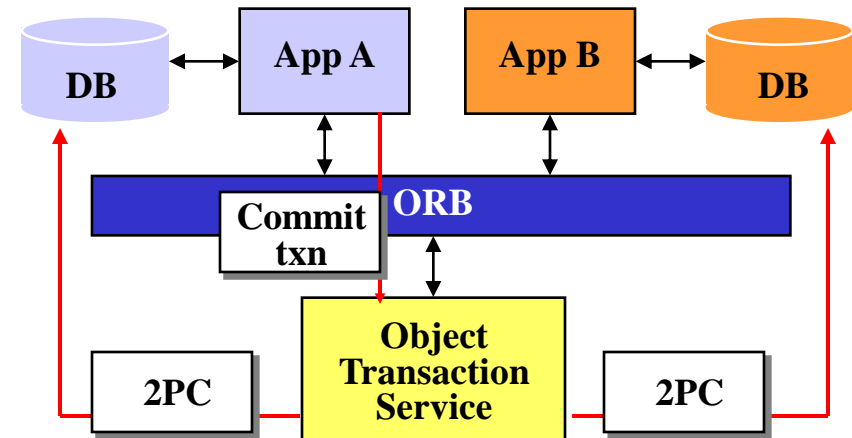
7) App B runs the txn but does not commit it at the end



6) App B registers the database for that transaction



8) App A request commit and the OTS runs 2PC



Summary WS-Transaction

- WS-Transaction is an example of how to apply the framework defined by WS-Coordination to define a specific protocol.
- WS-Transaction defines short lived **atomic transactions** standardizing the interfaces provided by traditional TP-monitor tools.
- However, in an Web services scenario, transactions may also take a long time to complete. For this case, WS-Transaction uses the notion of **business activity** and defines a protocol based on compensation (as opposed to locking) used to achieve distributed consensus on whether the results of a long-running message exchange should be made persistent.
- This standard defines the port types (WSDL interfaces) that must be implemented by each participant service depending on its role in the transaction (e.g., initiator, outcome listener). It also specifies the port types provided by the coordinator.
- The actual implementation of the operations corresponding to a commit, abort or compensate message are left unspecified as they are highly dependent on the business logic of the specific Web service.



Information and
Communication Systems
Research Group



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

9 WSIF: Web Services Invocation Framework



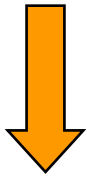
<http://ws.apache.org/wsif>

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
<http://www.iks.inf.ethz.ch/>

SOA and web services

There is no problem in system **design** that cannot be solved by adding a level of indirection.

There is no **performance** problem that cannot be solved by removing a level of indirection.



Take advantage of
Middleware but let the
system decide what
to use

- WS Invocation Framework
 - Use WSDL to describe a service
 - Use WSIF to let the system decide what to do when the service is invoked:
 - If the call is to a local EJB then do nothing
 - If the call is to a remote EJB then use RMI
 - If the call is to a queue then use JMS
 - If the call is to a remote Web service then use SOAP and XML
 - There is a single interface description, the system decides on the binding
 - This type of functionality is at the core of the notion of Service Oriented Architecture

Interfaces

- Web Services are intended to provide an standardized interface to conventional integration functionality
- SOAP provides an abstract, standardized way to exchange messages
- WSDL provides an abstract, standardized way to define interfaces
- UDDI provides an abstract and standardized way to look for services
- ...

- The problem is that the abstract description must always have a binding to a concrete implementation (e.g., RPC over HTTP)

- Web services lose some advantages if they always have to be used with a concrete implementation
 - Developers have to know the concrete implementation
 - It is difficult to change the concrete implementation once fixed
 - In most cases, the most generic implementation is chosen, which is typically the most expensive (XML format for SOAP messages)

- The ultimate goal of SOA is to use services as abstract concepts and ignore their concrete implementation (let the middleware take care of the protocol details)

The case for abstract interfaces

Fast prototyping

- ❑ Build a simple service in Java
- ❑ Test it and see how it works
- ❑ Replace it with a more robust EJB implementation incorporating additional features
- ❑ The change from prototype to final implementation is done without changing the interface and without requiring changes to the application using the service

Service evolution

- ❑ Replace an RPC based service with a queue based service for added persistence and delivery guarantees
- ❑ Include redundant service implementations working on the queue for added reliability
- ❑ The change can be made without affecting the client and application code

Alternative channels

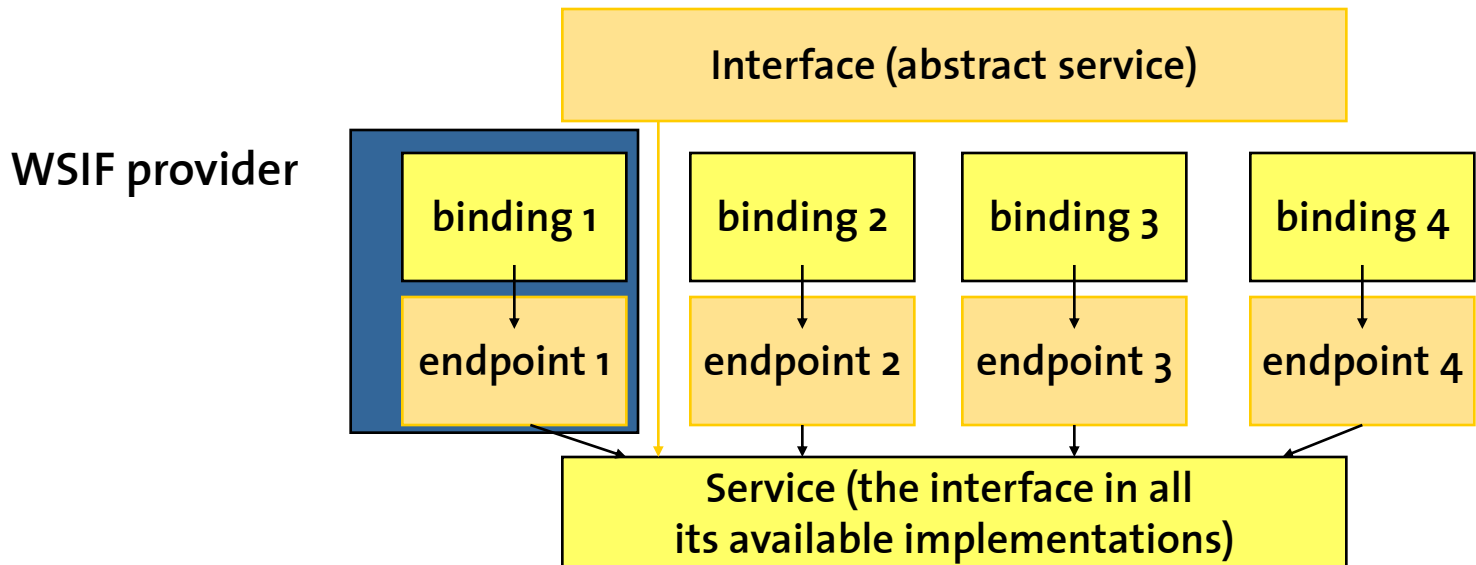
- ❑ For a given service, provide different access channels to be used according to Service Level Agreements
- ❑ Add alternative channels dynamically as requirements change

Loose coupling C/S

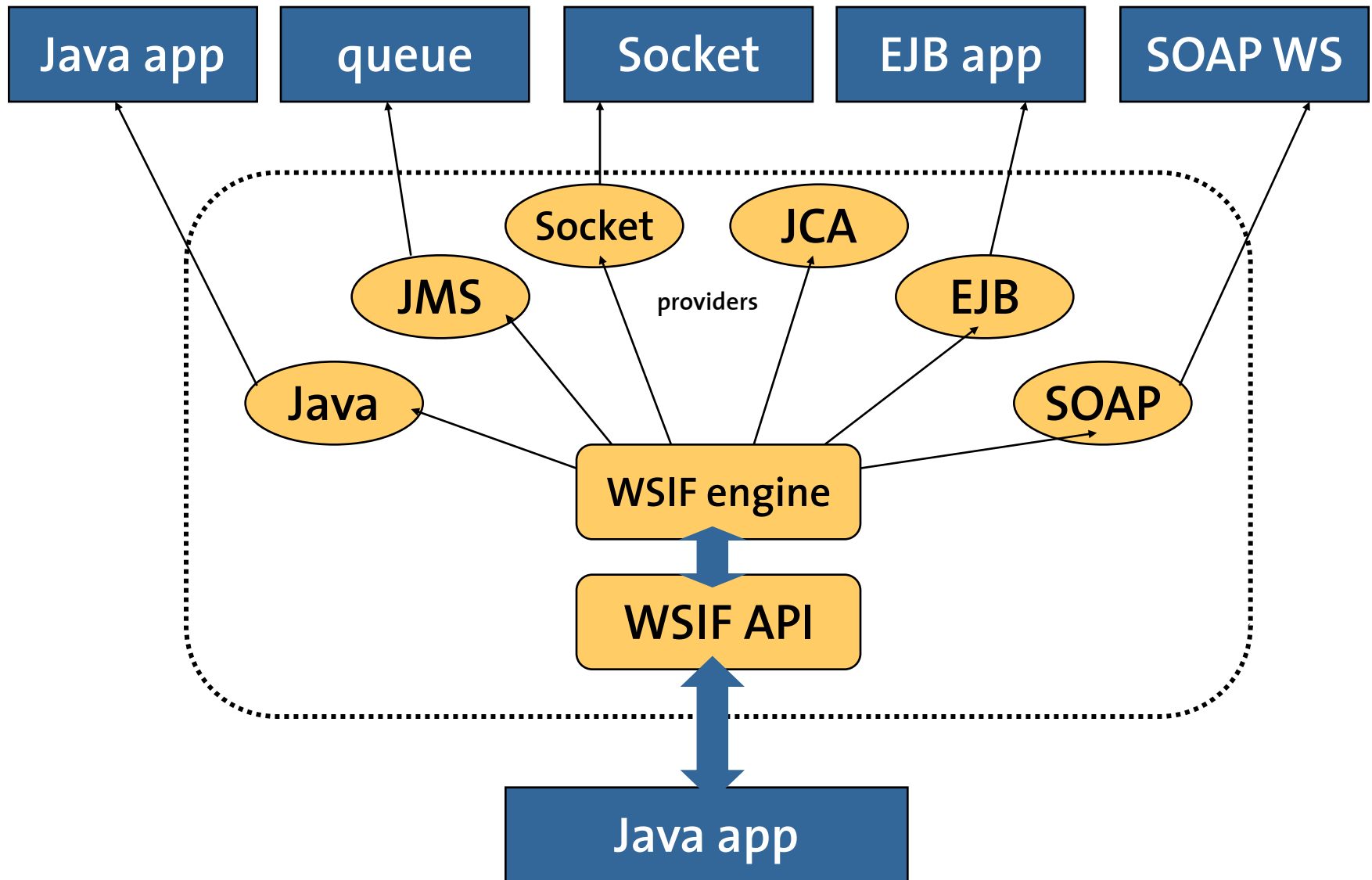
- ❑ The client invokes a service interface
- ❑ A dynamic binding determines at run time which concrete implementation of the abstract interface is to be used
- ❑ Which concrete implementation to use can change over time (e.g., based on autonomic functionality)

WSIF and WSDL

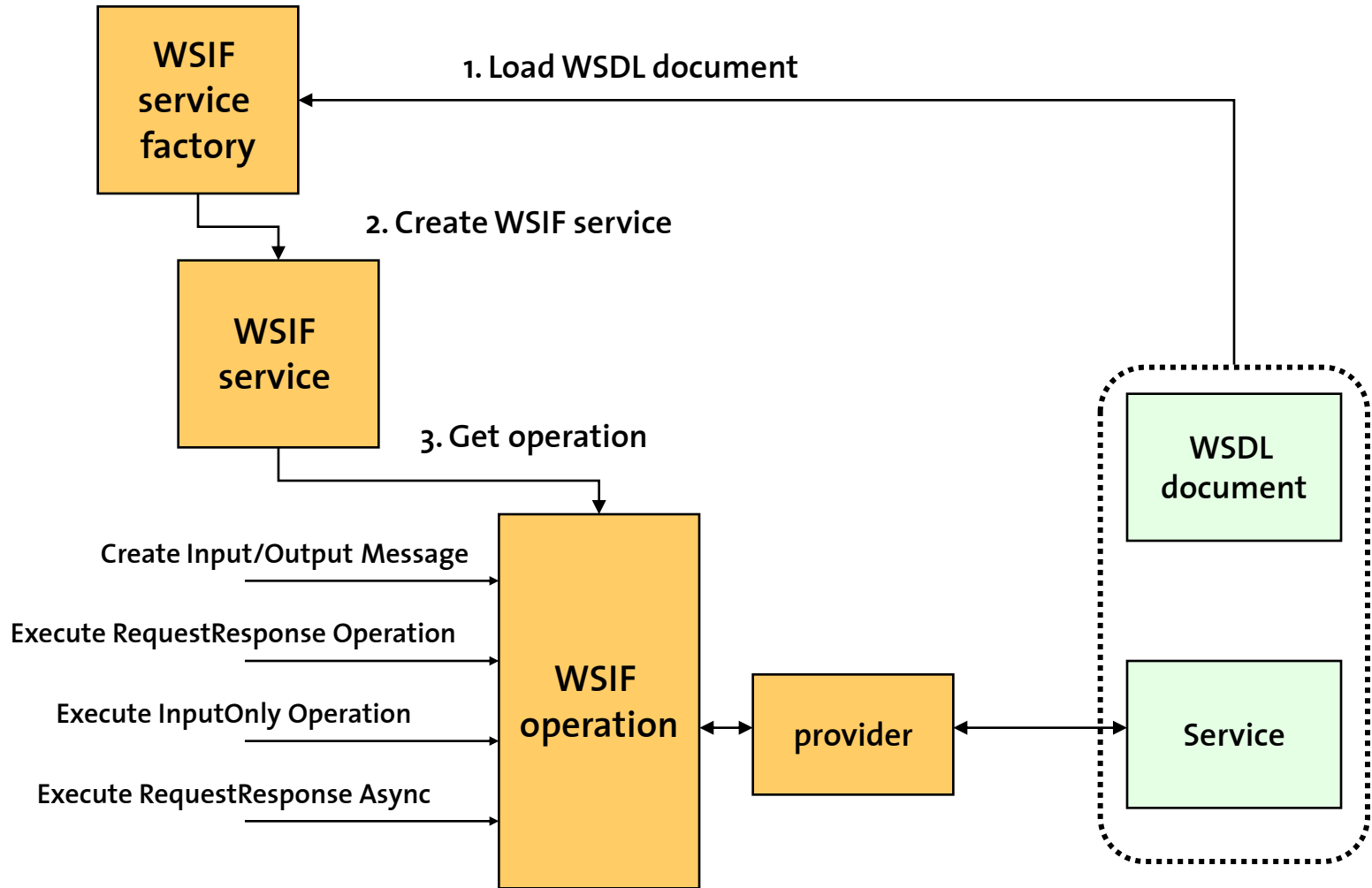
- WSIF has two parts:
 - A meta data description of the service (equivalent to the WSDL abstract service)
 - A set of providers implementing different concrete implementations of the interface
- A provider is a code module that implements a single WSDL binding and endpoint
- Providers use the J2SE JAR Service provider specification, making them discoverable at run time.



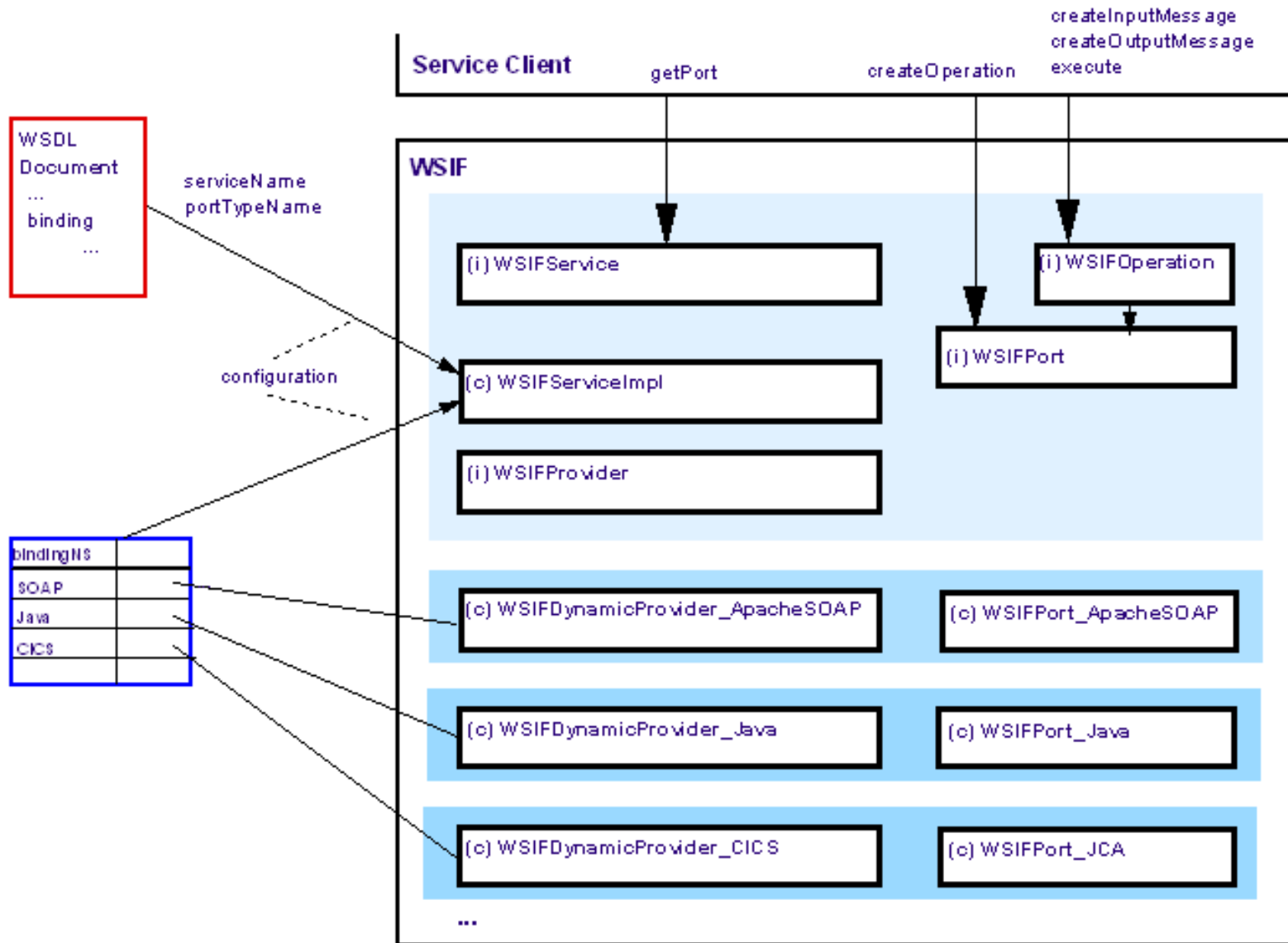
WSIF Structure



WSIF architecture



Basic interactions with WSIF



<http://ws.apache.org/wsif>

WSIF Use

- ❑ Create a Service Factory
- ❑ Use the Factory to read the WSDL describing the service to call and create a Service object that can be used to refer to the service
- ❑ If necessary, map the types used in the WSDL document to Java objects
- ❑ Use the Service object to create a Port object:
 - The port determines the binding
 - One can specify which one to use or choose a default
- ❑ Use the Port object to create the operation to invoke
 - Create the messages
 - Map the data to the messages
- ❑ Call the operation

- ❑ It is possible to invoke an operation asynchronously
 - By indicating a handler to be invoked when the response arrives
 - By simply updating the message output when the response arrives

- ❑ It is possible to use stubs for invocation (created at run time) through the interface of the Service object (`service.getStub`)

Example 1

```
WSIFServiceFactory factory = WSIFServiceFactory.newInstance();
```

```
WSIFService service = factory.getService(  
    wsdlLocation, // location of the wsdl file  
    null, // service namespace  
    null, // service  
    name "http://www.ibm.com/namespace/wsif/samples/ab", // port type namespace  
    "AddressBookPT" // port type name  
);
```

```
service.mapType(  
    new javax.xml.namespace.QName(  
        "http://www.ibm.com/namespace/wsif/samples/ab/types",  
        "address"),  
    Class.forName(  
        "com.ibm.www.namespace.wsif.samples.ab.types.WSIFAddress")  
    );  
service.mapType( new javax.xml.namespace.QName(  
    "http://www.ibm.com/namespace/wsif/samples/ab/types",  
    "phone"),  
    Class.forName(  
        "com.ibm.www.namespace.wsif.samples.ab.types.WSIFPhone")  
    );
```

```
port = service.getPort(portName);
```

Obtains a ServiceFactory and then creates a Service By reading the WSDL file

The WSDL contains complex type Elements. These Elements are Mapped to Java Classes using the Service just created

Generates a WSIF port for the service

Example 2

```
WSIFOperation operation =  
    port.createOperation("addEntry", "AddEntryWholeNameRequest", null);
```

```
WSIFMessage inputMessage = operation.createInputMessage();  
WSIFMessage outputMessage = operation.createOutputMessage();  
WSIFMessage faultMessage = operation.createFaultMessage();
```

```
String nameToAdd="Chris P. Bacon";  
WSIFAddress addressToAdd =  
    new WSIFAddress (1,  
        "The Waterfront",  
        "Some City",  
        "NY", 47907,  
        new WSIFPhone (765, "494", "4900"));
```

```
inputMessage.setObjectPart("name", nameToAdd);  
inputMessage.setObjectPart("address", addressToAdd);
```

```
operation.executeRequestResponseOperation( inputMessage, outputMessage, faultMessage);
```

Creates an
Operation object
for the operation
we want to call

Creates the
messages for
the operation

Prepares the data
for the messages
that will call the
operation

Calls the operation

Java binding

- Maps the WSDL abstract service directly to a Java class

```
<!-- Java binding -->
```

```
<binding ... >
```

```
  <java:binding/>
```

```
  <format:typeMapping style="uri" encoding="..." />?
```

```
    <format:typeMap typeName="qname"|elementName="qname" formatType="nmtoken" />*
```

```
  </format:typeMapping>
```

```
  <operation>*
```

```
    <java:operation
```

```
      methodName="nmtoken"
```

```
      parameterOrder="nmtoken"?
```

```
      returnPart="nmtoken"?
```

```
      methodType="instance|static|constructor"? />?
```

```
    <input name="nmtoken"? />?
```

```
    <output name="nmtoken"? />?
```

```
    <fault name="nmtoken"? />?
```

```
  </operation>
```

```
</binding>
```

```
<service ... >
```

```
  <port>*
```

```
    <java:address
```

```
      className="nmtoken"
```

```
      classPath="nmtoken"?
```

```
      classLoader="nmtoken"? />
```

```
  </port>
```

```
</service>
```

http://ws.apache.org/wsif/providers/wSDL_extensions/java_extension.html

EJB binding

- Maps a WSDL operation to an EJB

```

<!-- EJB binding -->
<binding ... >
  <ejb:binding/>
  <format:typeMapping style="uri" encoding="..."/>?
    <format:typeMap typeName="qname"|elementName="qname" formatType="nmtoken"/>*
  </format:typeMapping>
  <operation>*
    <ejb:operation
      methodName="nmtoken"
      parameterOrder="nmtoken"?
      returnPart="nmtoken"?
      interface="home|remote"? />?
    <input name="nmtoken"? />?
    <output name="nmtoken"? />?
    <fault name="nmtoken"? />?
  </operation>
</binding>
<service ... >
  <port>*
    <ejb:address
      className="nmtoken"
      jndiName="nmtoken"?
      initialContextFactory="nmtoken"?
      jndiProviderURL="url"? />
  </port>
</service>

```

JMS binding

- The JMS binding maps operations to JMS messages

```
<binding name="xxx" type="xxx">  
  <jms:binding type="ObjectMessage"/>  
  <format:typeMapping encoding="Java" style="Java">  
    <format:typeMap typeName="xxx" formatType="xxx" />  
  </format:typeMapping>  
  <operation name="xxx">  
    <input name="xxx" />  
    <output name="xxx" />  
  </operation>  
</binding>
```

```
<jms:address jmsVendorURI="xxx"  
  jndiDestinationName="xxx"  
  destinationStyle="xxx"  
  jndiConnectionFactoryName="xxx"  
  initialContextFactory="xxx"  
  jndiProviderURL="xxx"  
  jmsProviderDestinationName="xxx"  
  jmsImplementationSpecificURI="xxx" />
```

JMS fault message

```
<binding name="xxx" type="xxx">
  <jms:binding type="ObjectMessage"/>
  <format:typeMapping encoding="Java" style="Java" />
  <operation name="xxx">
    <input name="xxx" />
    <output name="xxx" />
    <fault name="xxx">
      <jms:faultIndicator type="property">
        <jms:faultProperty
          name="xxx"
          type="xxx"
          value="xxx"
          part="xxx"/>
      </jms:faultIndicator>
      <jms:fault parts="a b c"/>
      <jms:property
        name="xxx"
        type="xxx"
        part="xxx"/>
    </fault>
  </operation>
</binding>
```

WSIF and BPEL

- WSIF is an example of the flexibility that web services offer:
 - BPEL only considers web services (an often heard critique)
 - The cost of XML web services is very large compared to native operations
 - Web services cannot support all the attached contexts of native systems
 - If BPEL tries to standardize connections to all possible systems, it would fail as a standard
 - Instead, use WSIF to treat everything as a web service and let the infrastructure decide on the best binding (native, messaging, or XML-RPC)

- Oracle BPEL Process Manager uses this technique to extend the type of tasks that can be invoked from a BPEL process
 - Java classes, EJB, JCA, HTTP GET and POST, and sockets