# SOAP / WS-*

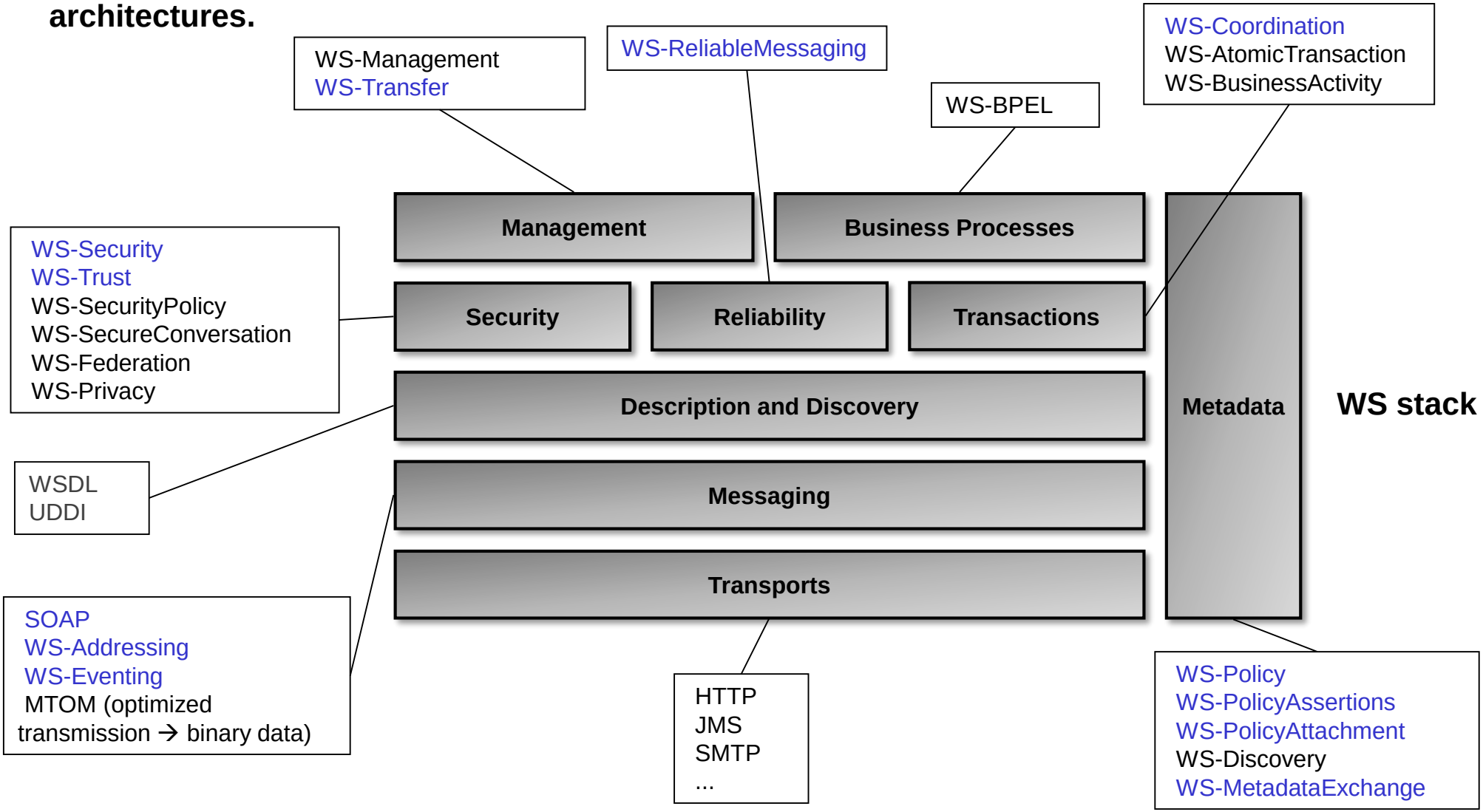
## OVERVIEW OF WS-* SOAP-BASED WEB SERVICE STANDARDS

**PETER R. EGLI**
**peteregli.net**

## Contents

# 1. Stack of WS-standards (1/2)

**The W3C and OASIS WS-stack provide a framework / toolbox for constructing web service architectures.**



# 1. Stack of WS-standards (2/2)

| Aspect | Standard | Description |
|---|---|---|
| Management | WS-Management | Web services for managing IT resources (servers, devices, applications etc.); alternative to SNMP or NetConf |
| | WS-Transfer | Defines the transfer of XML-based resources from a server to another server |
| Reliable messaging | WS-ReliableMessaging | Protocol for the reliable delivery of SOAP messages (at-most-once semantic, in-order delivery) |
| Business processes | WS-BPEL (BPEL4WS) | Defines the interaction of web services with business processes |
| Metadata | WS-Policy | Defines how a web service advertises its policies on security, QoS etc. |
| | WS-PolicyAssertions | Represents an individual preference, requirement or capability; a policy is composed of multiple policy assertions |
| | WS-PolicyAttachment | Binds a policy to a subject ( = typically a web service) |
| | WS-Discovery | Defines a web service multicast discovery protocol to locate web services (IPv4 address 239.255.255.250 port 3702) |
| | WS-MetadataExchange | Allows the retrieval of metadata of a web service endpoint (e.g. web service policies, WSDL file from server etc.) |
| Transactions | WS-Coordination | Protocols that coordinate the actions of distributed applications (web services) |
| | WS-AtomicTransaction | Defines protocols for web service transactions (completion, volatile two-phase commit, durable two-phase commit) |
| | WS-BusinessActivity | Defines business activity coordination |
| Security | WS-Security | Security protocol for web services |
| | WS-Trust | Protocol for establishing trust relationship between web service endpoints (consumers, providers) using tokens |
| | WS-SecurityPolicy | Defines security policy assertions (capabilities, requirements, preferences with regard to security) |
| | WS-SecureConversation | Protocol for securing entire conversations / sequences of messages (WS-Trust encrypts single messages with a token) |
| | WS-Federation | Identity federation for web services; allows different security realms (different organizations) to share security information on identities |
| | WS-Privacy | Defines privacy assertions that are embedded into web service policy descriptions (defines which data may be accessed) (see example http://www.w3.org/TR/P3P/#Example_policy) |
| Messaging | WS-Addressing | Protocol for communicating web service address information (web service endpoint address) |
| | WS-Eventing | Subscribe / publish to web services to receive notifications / events |

# 2. WS Publish / subscribe with WS-Eventing (1/3)

**WS-Eventing defines a protocol to subscribe to a web service to receive events from that service.**

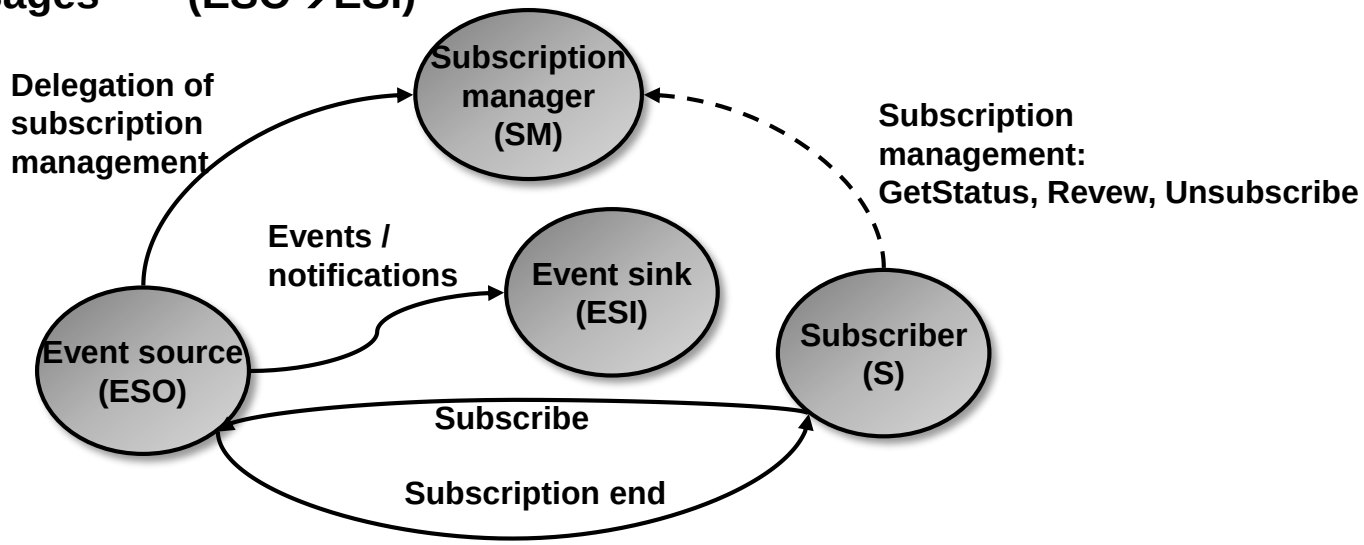
**N.B.: The event source is not necessarily the same application as the event subscriber (subscriber performs subscription on behalf of event sink).**

**2 types of messages:**

**1 Subscription management messages:**

- **1a Subscribe** (S→ESO: Event notification subscription)
- **1c GetStatus** (S→SM: Get status information of existing subscription)
- **1b Renew** (S→SM: Renewal of existing event notification)
- **1d Unsubscribe** (S→SM: Unsubscribe, subscriber initiated)
- **1e Subscription end** (ESO→S: Termination of existing subscription)

**2 Notification messages** (ESO→ESI)



# 2. WS Publish / subscribe with WS-Eventing (2/3)

## Example subscription message:

```
<s12:Envelope
    xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"
    xmlns:ew="http://www.example.com/warnings" >
  <s12:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
    </wsa:Action>
    <wsa:MessageID>
      uuid:d7c5726b-de29-4313-b4d4-b3425b200839
    </wsa:MessageID>
    <wsa:ReplyTo>
     <wsa:Address>http://www.example.com/MyEventSink</wsa:Address>
    </wsa:ReplyTo>
    <wsa:To>http://www.example.org/oceanwatch/EventSource</wsa:To>
  </s12:Header>
  <s12:Body>
    <wse:Subscribe>
      <wse:Delivery>
        <wse:NotifyTo>
          <wsa:Address>
            http://www.example.com/MyEventSink/OnStormWarning
          </wsa:Address>
          <wsa:ReferenceProperties>
            <ew:MySubscription>2597</ew:MySubscription>
          </wsa:ReferenceProperties>
        </wse:NotifyTo>
      </wse:Delivery>
    </wse:Subscribe>
  </s12:Body>
</s12:Envelope>
```

Subscription action

Event sink to receive notifications, identified with WS-Addressing element

Source: http://www.w3.org/Submission/WS-Eventing/#Example

# 2. WS Publish / subscribe with WS-Eventing (3/3)

## Example notification message:

```
<s12:Envelope
    xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:ew="http://www.example.com/warnings"
    xmlns:ow="http://www.example.org/oceanwatch" >
  <s12:Header>
    <wsa:Action>
      http://www.example.org/oceanwatch/2003/WindReport
    </wsa:Action>
    <wsa:MessageID>
      uuid:568b4ff2-5bc1-4512-957c-0fa545fd8d7f
    </wsa:MessageID>
    <wsa:To>http://www.other.example.com/OnStormWarning</wsa:To>
    <ew:MySubscription>2597</ew:MySubscription>
    <ow:EventTopics>weather.report weather.storms</ow:EventTopics>
  </s12:Header>
  <s12:Body>
    <ow:WindReport>
      <ow:Date>030701</ow:Date>
      <ow:Time>0041</ow:Time>
      <ow:Speed>65</ow:Speed>
      <ow:Location>BRADENTON BEACH</ow:Location>
      <ow:County>MANATEE</ow:County>
      <ow:State>FL</ow:State>
      <ow:Lat>2746</ow:Lat>
      <ow:Long>8270</ow:Long>
      <ow:Comments xml:lang="en-US" >
        WINDS 55 WITH GUSTS TO 65. ROOF TORN OFF BOAT HOUSE. REPORTED
        BY STORM SPOTTER. (TBW)
      </ow:Comments>
    </ow:WindReport>
  </s12:Body>
</s12:Envelope>
```

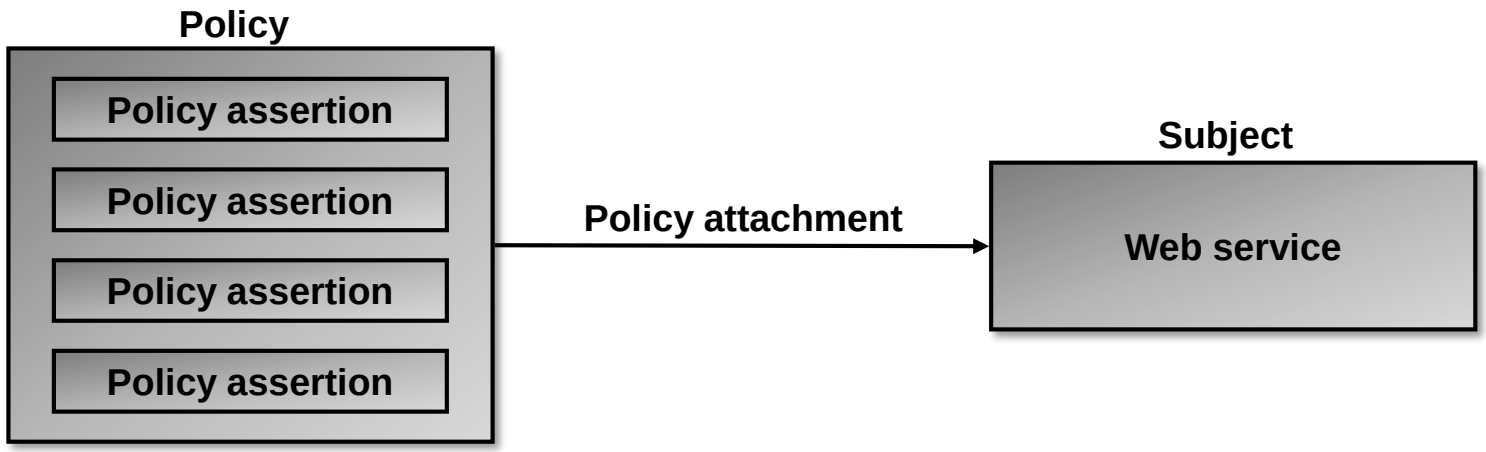
Source: http://www.w3.org/Submission/WS-Eventing/#Example

# 3. WS-Policy / WS-PolicyAttachment (1/3)

**WS-Policy allows web services to advertise XML-representations of their policies (e.g. security, or Quality of Service) and web service consumers to specify their policy requirements.**

**Terminology / model:**

- **Policy** — Set of information being expressed as *policy assertions.*
- **Policy Assertion** — Represents an individual preference, requirement, capability.
- **Policy Expression** — Set of one or more *policy assertions, expressed in XML.*
- **Policy Subject** — An entity to which a *policy expression* can be associated (typically a web service or a web service endpoint).
- **Policy Attachment** — Describes to which element a policy applies. A policy may be placed directly into an element it applies to or be referenced from an element. The attachment is described by WS-PolicyAttachment.



# 3. WS-Policy / WS-PolicyAttachment (2/3)

## Examples:

```
<sp:TransportBinding>
  <Policy>
    <sp:TransportToken>
      <Policy>
        <sp:HttpsToken>
          <wsp:Policy/>
        </sp:HttpsToken>
      </Policy>
    </sp:TransportToken>
    <sp:AlgorithmSuite>
      <Policy>
        <sp:Basic256Rsa15/>
      </Policy>
    </sp:AlgorithmSuite>
  </Policy>
</sp:TransportBinding>
```

**Policy specifies that HTTPs is required as transport.**

**Policy defines that HTTPs is to use the RSA15 key wrap algorithm with 256 bit basic encryption algorithm.**

```
<wsp:Policy
xmlns:rmp=http://schemas.xmlsoap.org/ws/2005/02/rm/policy
xmlns:wsp=http://schemas.xmlsoap.org/ws/2004/09/policy
xmlns:wsu=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
wsu:Id="RmPolicy" >
  <rmp:RMAssertion>
    <rmp:InactivityTimeout Milliseconds="600000" />
    <rmp:BaseRetransmissionInterval Milliseconds="3000" >
    <rmp:ExponentialBackoff />
    <rmp:AcknowledgementInterval Milliseconds="200" />
  </rmp:RMAssertion>
</wsp:Policy>
```

**Properties for reliable messaging.**

# 3. WS-Policy / WS-PolicyAttachment (3/3)

**WS-PolicyAttachment defines how to associate a policy with a subject.**

## Example:

```
<wsp:PolicyAttachment>
  <wsp:AppliesTo>
    <wsa:EndpointReference xmlns:indigoo="…">
      <wsa:Address>http://www.indigoo.example.com/wsa</wsa:Address>
      <wsa:PortType>indigoo:InventoryPortType</wsa:PortType>
      <wsa:ServiceName>indigoo:InventoryService</wsa:ServiceName>
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  <wsp:PolicyReference URI="http://www.indigoo.example.com/policies#RmPolicy" />
</wsp:PolicyAttachment>
```

**Subject to which policy applies.**

**Referenced policy.**

# 4. Web service security (1/8)

**General security aspects:**

**Authentication:**

**Ensure the identity of a user.**

**Authorization:**

**Ensure that only authorized users can access an application.**
**Control the level of access (read/write, access to which data / resources).**

**Confidentiality:**

**Keep data secret for unauthorized parties (machines, users).**

**Integrity:**

**Verify that data has not been changed by unauthorized third parties in transmit.**

**Non-repudiation:**

**Ensure that the sender of a message can not deny having sent it.**

# 4. Web service security (2/8)

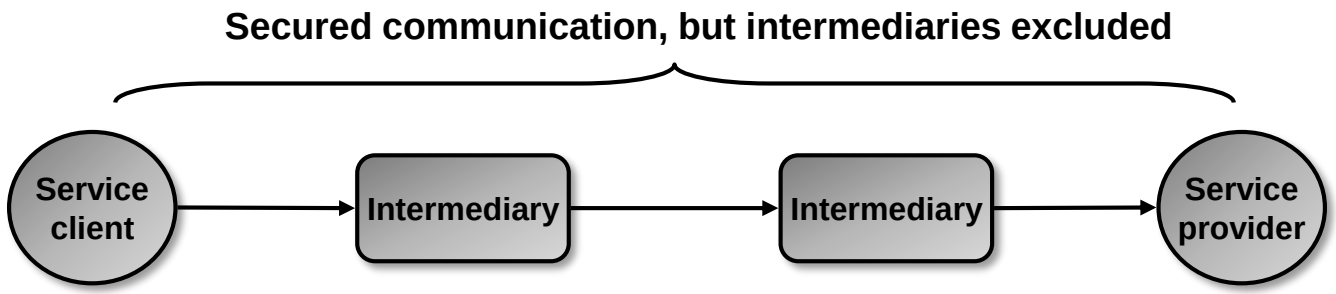
## Scenarios for security (1/2)

**1. Security on application layer:**

**Messages are secured on the application layer, i.e. entire SOAP messages are encrypted and authenticated by the web service stubs / skeletons on the service client and service provider.**

- ☺ End-to-end security.
- ☹ Web intermediaries (caches, proxies, logging servers etc.) excluded (cannot be used).
- ☹ Complete web service architecture with proxies etc. not possible.

→ Recommended in closed-world scenarios only (company-internal).



# 4. Web service security (3/8)

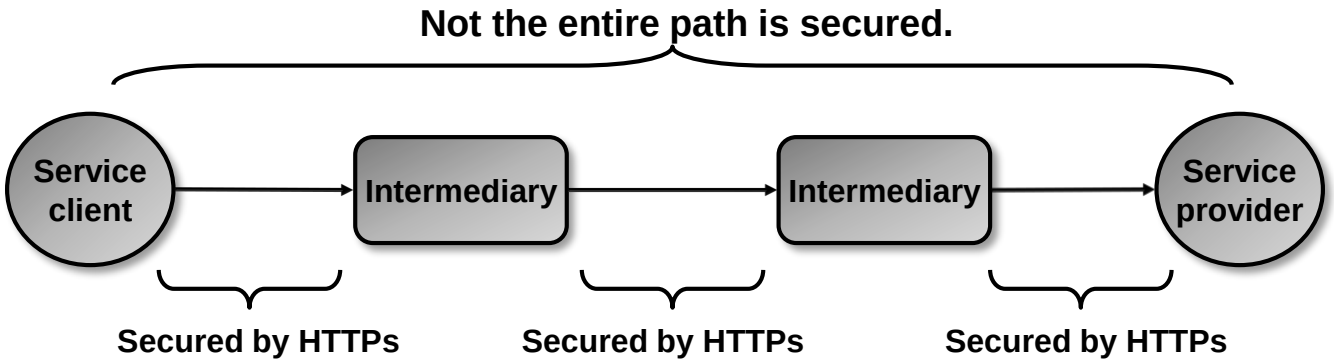
## Scenarios for security (2/2)

**2. Security on transport layer:**

**SOAP intermediaries (caches etc.) along the path unpack / decrypt SOAP messages on receive and encrypt the messages again on transmit.**
**Normally HTTPs is used for securing the SOAP message transmission.**

- ☺ **Intermediaries are possible.**
- ☹ **Security not sufficient (HTTPs only secures hop-to-hop connection, but not the entire path.**

**→ Required but not sufficient for achieving reasonable security.**




**3. Message-level security control with WS-Security:**

**WS-Security affords fine-grain security control to achieve the required security and make it possible to use web intermediaries (proxies, caches etc.).**

# 4. Web service security (4/8)

**WS-Security (1/2):**

**→ WS-Security is a framework for building security protocols.**

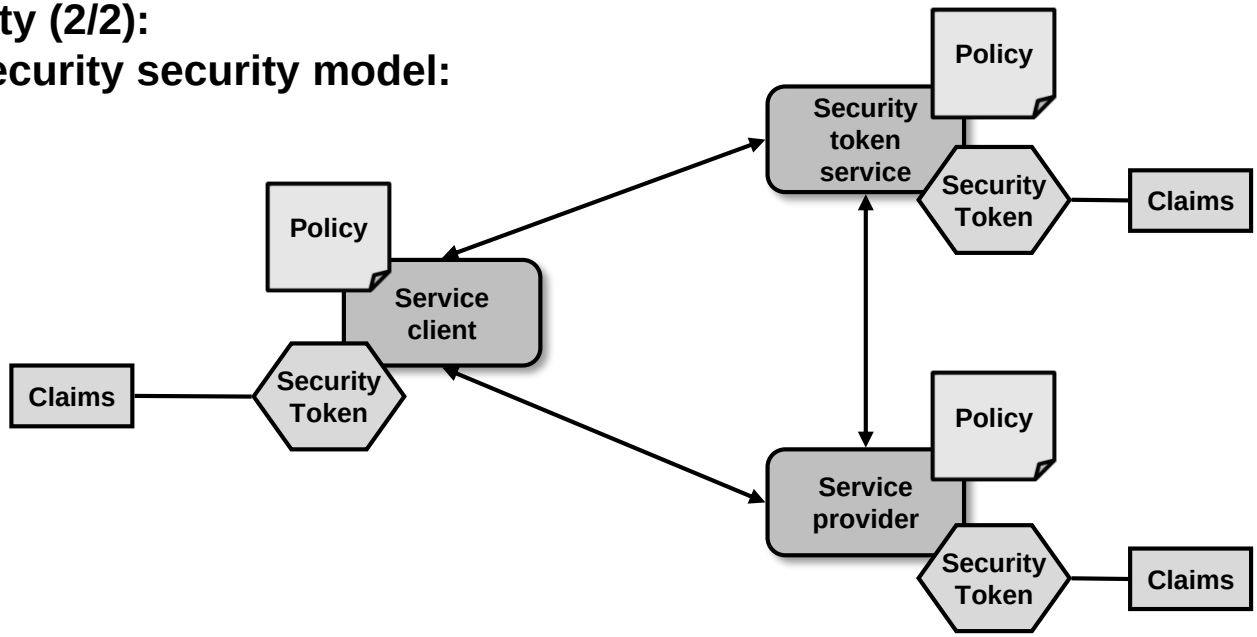
**It includes:**

**a. Authentication and Integrity (signature).**
**b. Confidentiality (encryption).**
**c. Propagation of security tokens.**
**d. Support for pluggable security algorithms (e.g. for encryption).**
**e. Encryption (control of encryption; the encryption algorithm itself is outside the scope of WS-Security).**
**f. Digest (procedure for calculating message digests for authentication).**
**g. Signature (message integrity).**

# 4. Web service security (5/8)

## WS-Security (2/2):

## The WS-Security security model:




| Entity | Description |
|---|---|
| Claim | A claim is a declaration made by an entity (e.g. name, identity, key, group, privilege, capability, etc). Example: Client claims to be the one with identity <identity>. This has to be verified in the process „Claim confirmation“. |
| Claim confirmation | A claim confirmation is the process of verifying that a claim applies to an entity. |
| Subject | An item about which the claims expressed in the security token apply (e.g. a web service). |
| Security token | A security token represents a collection of claims. |
| Policy | The claims and related information that web services require in order to process messages. |

# 4. Web service security (6/8)

**WS-Trust (1/3):**

**WS-Trust is an add-on „standard“ that builds on WS-Security and adds the possibility to establish trust relationship between web service client and service by exchanging security tokens (e.g. SAML tokens).**

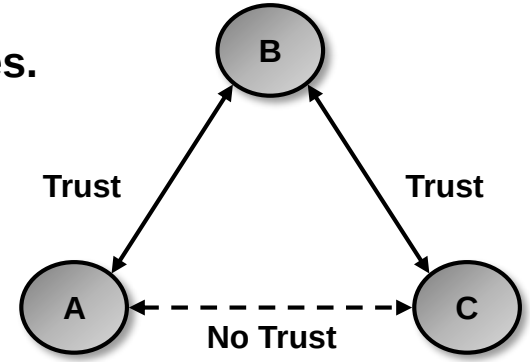
**Problem:**

**Trust relationship is an intrinsic security problem of networked entities.**

**Example:**

**A trusts B, C trusts B, but C does not trust A, A does not trust C.**

**How to establish trust between A and C?**




**Solution for WS:**

**B (security token service) issues a security token for A (service client) and C (service provider).**
**B (security token service) verifies the token on request of A (service client) & C (service provider).**

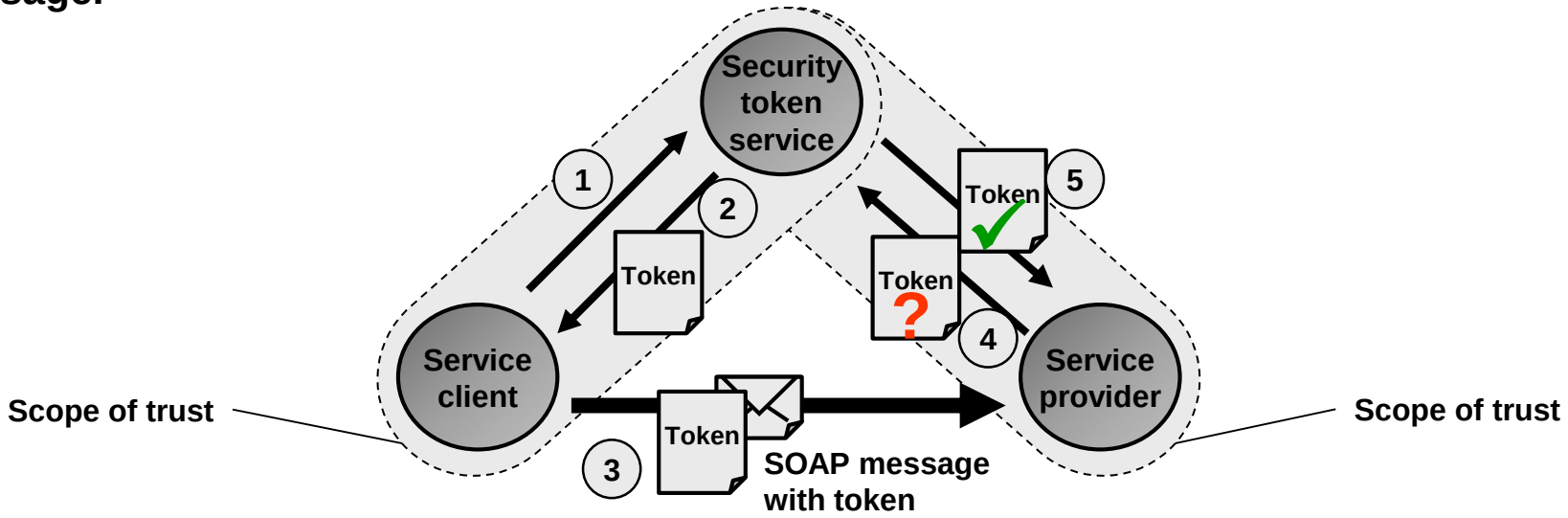
**The security token service acts like a person of trust to establish the trust relation between service client (A) and service provider (C).**

# 4. Web service security (7/8)

## WS-Trust (2/3):

**WS-Trust protocol:**

1. The client initiates a token request to the security token service.
2. The security token service returns a new security token.
3. The client sends a SOAP message (to access the service) along with the token to the service provider.
4. Before processing the SOAP message, the service provider sends the token to the security token service for verification.
5. The security token service sends back an confirmation of the validity of the token.
6. Now the trust relationship is established. The service provider processes the SOAP message.



# 4. Web service security (8/8)

## WS-Trust (3/3)

## WS-Trust example:

```
<wst:RequestSecurityTokenResponse>
  <wst:TokenType>
  http://example.org/mysecuritytoken
  </wst:TokenType>
  <wsp:AppliesTo>
   <wsa:EndpointReference>
    <wsa:Address>http://example.org/webservice</wsa:Address>
   </wsa:EndpointReference>
  </wsp:AppliesTo>
  <wst:Lifetime>
   <wsu:Created>2004-05-06T22:04:34</wsu:Created>
   <wsu:Expires>2004-05-07T10:04:34</wsu:Expires>
  </wst:Lifetime>
  <wst:RequestedSecurityToken>
   <p:MySecurityToken xmlns:p='http://example.org/mytoken' >
   <!-- Token data -->
   </p:MySecurityToken>
  </wst:RequestedSecurityToken>
  <wst:RequestedProofToken>
   <xenc:EncryptedKey xmlns:xenc='http://www.w3.org/2001/04/xmlenc#' >
    <xenc:EncryptionMethod
      Algorithm='http://www.w3.org/2001/04/xmlenc#kw-rsa-oaep-mgf1p' />
    <ds:KeyInfo>
     <wss:SecurityTokenReference>
      <wss:Reference URI='#You' ValueType='http://docs.oasis-
        open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3' />
     </wss:SecurityTokenReference>
    </ds:KeyInfo>
    <xenc:CipherData xmlns:xenc='http://www.w3.org/2001/04/xmlenc#' >
     <xenc:CipherValue>1yLLyrm3qudM1b89dYsRGw==</xenc:CipherValue>
    </xenc:CipherData>
   </xenc:EncryptedKey>
  </wst:RequestedProofToken>
</wst:RequestSecurityTokenResponse>
```

**Lifetime elements defines validity period of token.**

**Security token data.**

# 5. Web service reliability / QoS (1/2)

**Problems with HTTP / SMTP or other protocols used for SOAP message transport:**

1. Messages arrive 0...n times (there is no guarantee of delivery or prevention of duplicates depending on the transport binding).
2. Messages may not arrive in order (messages may be re-ordered somewhere in the transmission path, e.g. because multiple threads in SOAP processor work in parallel, receive messages and process them).

**Solution 1:**

Problems / exceptions are handled in the application.
But: Duplication of functionality common to almost all applications.

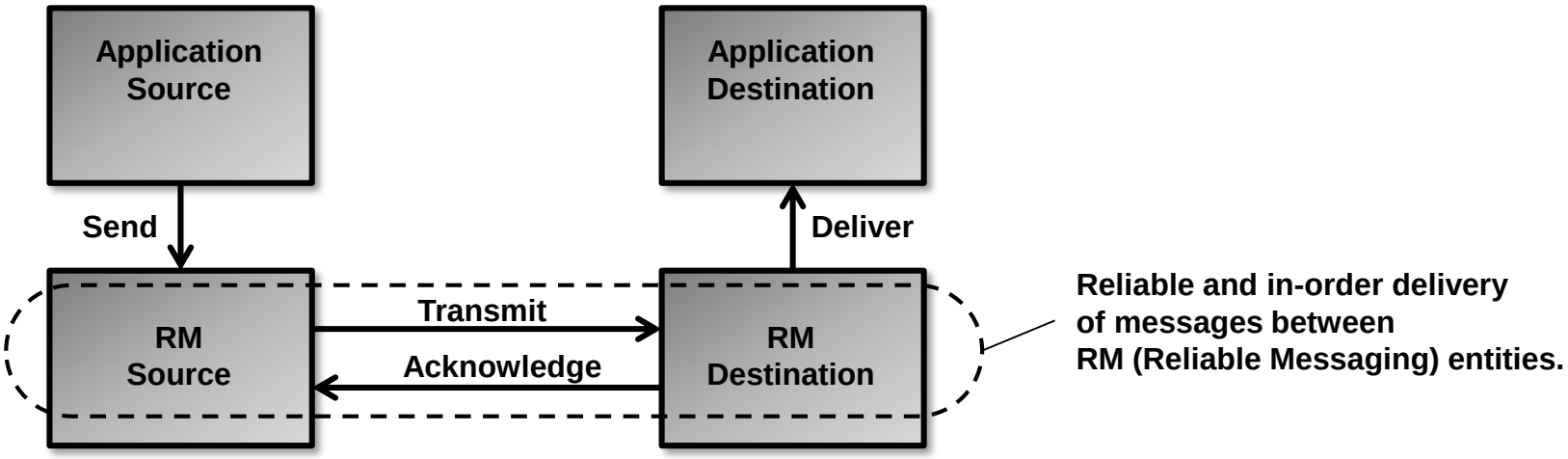
**Solution 2:**

Use WS-ReliableMessaging to achieve web service QoS (Quality of Service).
→ Exactly-once transmission semantics (retransmission of lost messages, no duplicates).
→ In-order delivery.

# 5. Web service reliability / QoS (2/2)

**WS-ReliableMessaging (WS-RM) messaging model:**

**The source (web service client) sends its SOAP message to an intermediary „RM Source“ which forwards the message reliably to the „RM Destination“.**
**The „RM Destination“ delivers the message to the final application destination (web service).**




**Possible WS-RM delivery assurances:**

1. AtLeastOnce.
2. AtMostOnce.
3. ExactlyOnce.
4. InOrder (can be combined with 1. – 3.).

# 6. WS-Addressing (1/3)

**Problems with message routing in web architectures:**

**When only using the <endpoint> information in the WSDL containing an URL for addressing a web service, the destination and reply-to address are the same and the web service provider has to send back the SOAP-response message to the requestor address. So the response must be sent in the same HTTP connection as the request was received by the service.**
**Sometimes it is desireable to use a different transport for the response and maybe also specify a different target address for fault messages.**
**→ SOAP does not specify the destination address (service address).**
**→ SOAP does not specify how and where to return a response.**
**→ SOAP does not specify how and where to report errors.**

**Solution 1:**

**Add the extra addressing information as a parameter to the request URL (?replyTo=<URL>).**
**But:**
**Service is still bound to HTTP as transport protocol (it is not possible to use SMTP or JMS).**
**And: This is not SOAP-ish, it is RESTful!**

# 6. WS-Addressing (2/3)

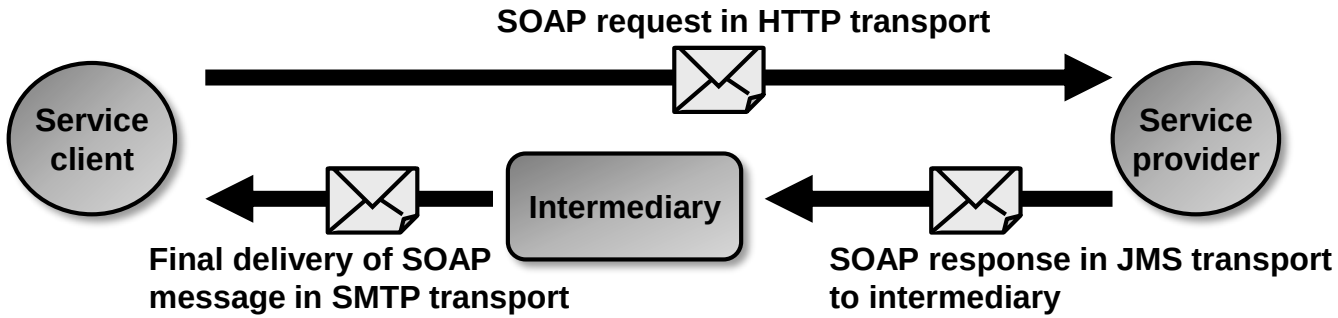
**Solution 2:**

**Use of WS-Addressing which allows to put endpoint references (EPR) into SOAP headers for specifying:**

**a. From address (address as a URL where the request came from).**
**b. To address (address as a URL to identify a specific instance of the service provider).**
**c. ReplyTo address (address as a URL where to send the response).**
**d. FaultTo address (address as a URL where to send fault messages).**

**EPRs allow to address individual instances of web services, e.g. individual resources (like REST does with the URL query string).**

**With WS-Addressing it is possible to use different transports for the request and response messages, e.g. request in HTTP, response in SMTP or JMS message.**
**This further improves the temporal and protocol-decoupling of service client and service provider:**



# 6. WS-Addressing (3/3)

## WS-Addressing example:

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
<S:Header>
  <wsa:MessageID>uuid:6B29FC40-CA47-1067-B31D-00DD010662DA</wsa:MessageID>

  <wsa:ReplyTo>
    <wsa:Address>http://indigoo.example/client1</wsa:Address>
  </wsa:ReplyTo>

  <wsa:To>http://indigoo.example/Purchasing</wsa:To>

  <wsa:Action>http://indigoo.example/SubmitPO</wsa:Action>

</S:Header>
<S:Body>
...
</S:Body>
```

**EPR address where the service should send respones.**

**EPR address of the ultimate receiver of the message (web service instance to process the message).**

**Action from HTTP-header mapped to SOAP-header so that the service becomes fully transport-independent.**

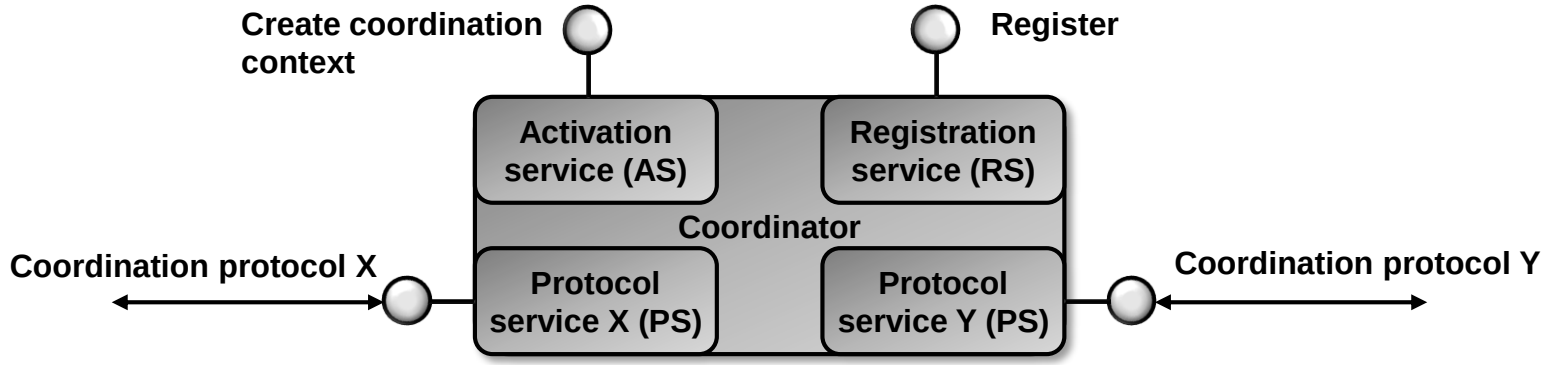
# 7. WS-Coordination (1/4)

**WS-Coordination is a framework for defining protocols that coordinate the actions of distributed applications (web services clients, service providers, intermediaries etc.).**

**WS-Coordination itself is only a framework and only works in conjunction with concrete standards like WS-Transaction and WS-AtomicTransaction.**

**The core concept of WS-Coordination is a „coordination context“ that is created each time a new activity is started in which multiple applications are involved and need to be coordinated in a controlled way (workflow with defined sequences of actions).**
**Example: Flight booking involving several web services (flight list, reservation, money transfer etc.).**

**Participants in a coordinated activity use coordinators to have the activity run in a coordinated / well-defined way (proper sequence of actions).**

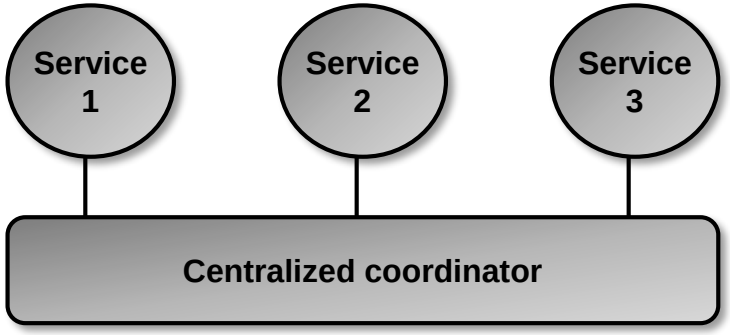


# 7. WS-Coordination (2/4)

**WS-Coordination architecture:**

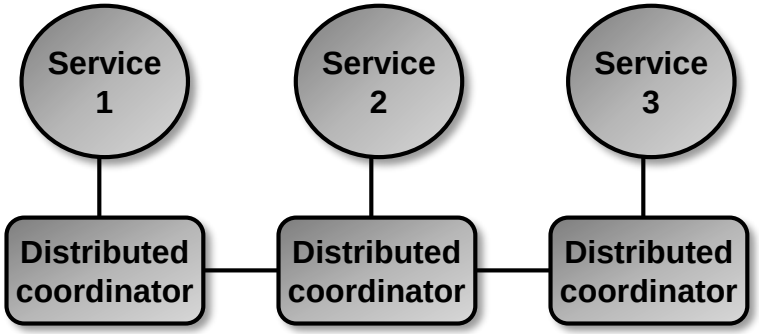
**A. Centralized coordinator:**

**→ All services share a common coordinator.**



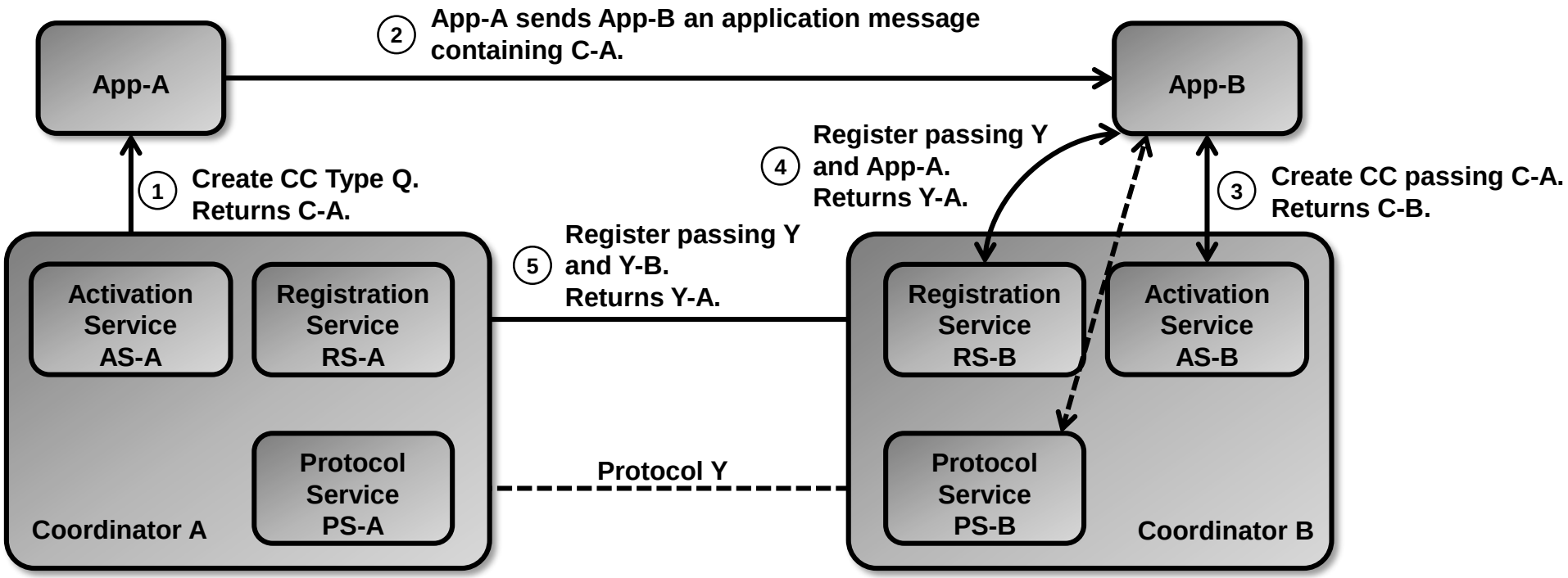

**B. Distributed coordinator:**

**Each service has its own coordinator. The coordinators run the coordination protocol between them.**



# 7. WS-Coordination (3/4)

## Example of coordination (1/2):




Source: http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os/wstx-wscoor-1.2-spec-os.html

# 7. WS-Coordination (4/4)

## Example of coordination (2/2):

**1. Coordination context creation A:**
**App-A sends a CreateCoordinationContext for coordination type Q, getting back a Context C-A that contains the activity identifier A1, the coordination type Q and a PortReference to CoordinatorA's registration service RS-A.**

**2. Coordination context forwarding:**
**App-A then sends an application message to App-B containing the Context C-A.**

**3. Coordination context creation B:**
**App-B prefers CoordinatorB, so it uses CreateCoordinationContext with C-A as an input to interpose CoordinatorB. CoordinatorB creates its own CoordinationContext C-B that contains the same activity identifier and coordination type as C-A but with its own registration service RS-B.**

**4. Coordination protocol selection:**
**App-B determines the coordination protocols supported by the coordination type Q and then registers for a coordination protocol Y at CoordinatorB, exchanging PortReferences for App-B and the protocol service Y-B. This forms a logical connection between these PortReferences that protocol Y can use.**

**5. Coordination protocol forwarding:**
**This registration causes CoordinatorB to forward the registration onto CoordinatorA's registration service RS-A, exchanging PortReferences for Y-B and the protocol service Y-A. This forms a logical connection between these PortReferences that the protocol Y can use.**

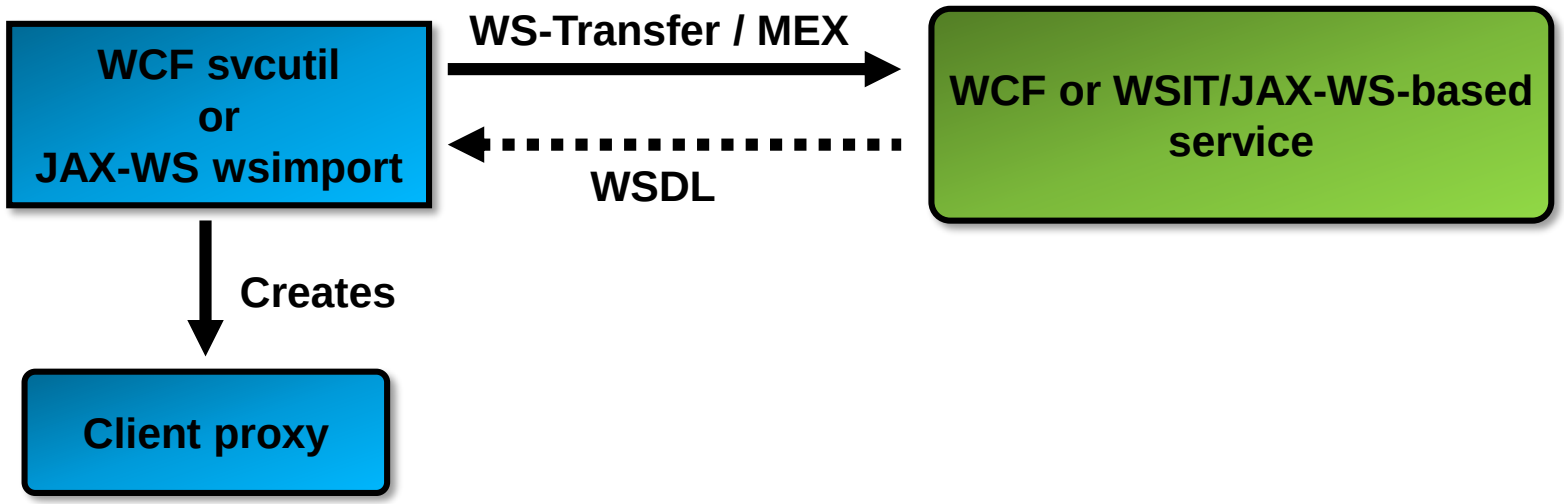
# 8. WS-MetadataExchange

**WS-MetadataExchange defines a protocol for bootstrapping a web service with MEX (Metadata Exchange = download of the WSDL and other files).**

**WS-MetadataExchange allows dynamically downloading the following WS metadata files:**

**a. WS-Policy (XML description of WS capabilities, requirements and general characteristics like QoS, security etc.).**
**b. WSDL file (description of WS operations).**
**c. XML schema (description of messages exchanged with WS).**

**WS-MetadataExchange uses WS-Transfer as protocol for the transfer / download of the metadata files.**



# 9. SOAP-WS versus REST-WS

**SOAP-WS:**

- Rather complete (there is a WS-standard for almost every aspect / problem).
- Modular (take what you need and compose your web service architecture).
- Complex (too many different WS-standards with dependencies to each other, difficult to find a common base that is supported by all participants).
- Performance penalty due to chatty protocols with large overhead (SOAP).

**Applicability / suitability:**
**Enterprise SOA-architecture which requires security, orchestration, management etc.**

**REST-WS:**

- Simple.
- Fits the bill for most applications?
- No standard, semantics of service mostly described in human readable form, not machine processable without description language (e.g. WADL or WSDL 2.0).
- Too simple (missing functionality for advanced services which require coordination etc.).

**Applicability:**
**Simple and isolated access (read) to data / resources.**