

Seminar: Service-orientierte Architektur (SOA)

Thema: WS-Coordination: Zielsetzung, Lösung und Beispiel

von Sven Bohnsack, mi2024

Seminarleiter: Sebastian Iwanowski

Fachhochschule Wedel

07.12.06

Inhaltsangabe:

1. Einleitung
 - 1.1 Was ist WS-Coordination?
 - 1.2 Zielsetzung WS-Coordination
 - 1.3 WS-Coordination – Kontext und Model

2. Coordination service

3. Komponenten WS-Coordination
 - 3.1 Überblick WS-Coordination
 - 3.2 CoordinationContext
 - 3.3 Activation service
 - 3.3.1 CreateCoordinationContext
 - 3.3.2 CreateCoordinationContextResponse
 - 3.4 Registration service
 - 3.4.1 Register Message
 - 3.4.2 RegistrationResponseMessage
 - 3.5 Importieren einer activity
 - 3.6 Coordination Faults

4. Beispiel

1. Einleitung

1.1 Was ist WS-Coordination?

Diese Spezifikation wurde im Rahmen des WS-* Workshopprozesses entwickelt und hat momentan den Status eines *working draft*, steht also für die Öffentlichkeit zum bewerten und testen der Implementation zur Verfügung.

WS-Coordination ist ein erweiterbares Framework, welches Protokolle bereithält, die die Aktionen einzelner Applikationen koordinieren und dadurch die gemeinsame Abarbeitung einer Aufgabe (WebService, z.B.: Börsendaten beim Aktienmarktserver abfragen und bei bestimmten Werten Kaufoption bei Hausbankserver des Nutzers ersteigern, Meldung an den Benutzer aufs Handy, per E-Mail, etc.pp.) erlaubt. WS-Coordination bietet außerdem Standardmechanismen, um WebServices zu entwickeln und zu registrieren.

Die Spezifikation ist ein vorgeschlagener IT-Industriestandard, entwickelt von einem Konsortium bestehend aus der Arjuna Technologies, Ltd., BEA Systems, Hitachi, International Business Machines (IBM) Corporation und der Microsoft Corporation [1]. Die Spezifikation gehört zu einer Serie von Spezifikationen, welches oben genanntes Konsortium seit 2002 entwickelt.

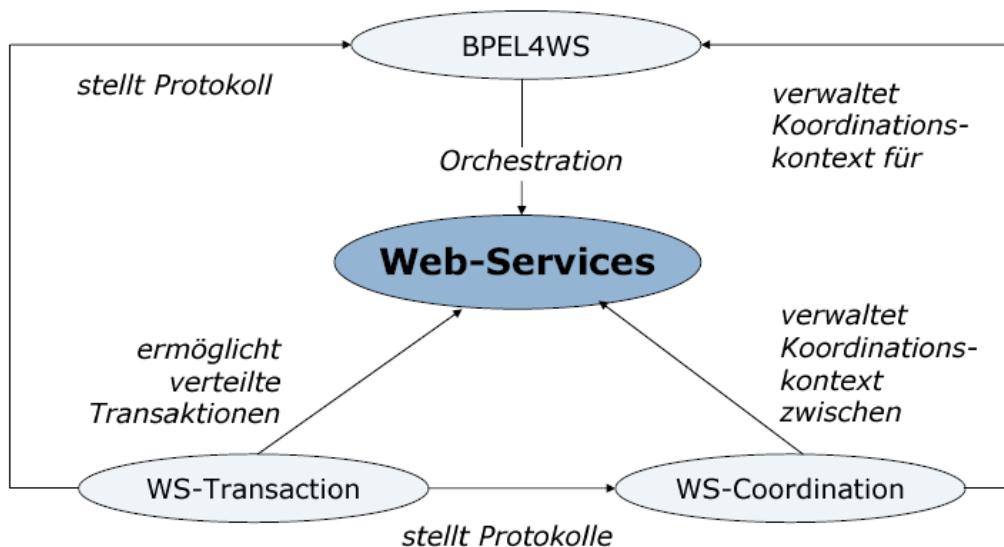
1.2 Zielsetzung WS-Coordination

Durch Nutzung von Web Services wird es Applikationen ermöglicht, verschiedenste Dienste plattformübergreifend via Internet zu nutzen. Von der Applikation initiiert entstehen Einheiten verteilt ablaufender Verarbeitung, wobei die Zahl der Beteiligten beliebig groß sein kann. Die resultierenden Einheiten sind häufig komplex in ihrer Struktur mit ebenso komplexen Verhältnissen zwischen ihren Teilnehmern. Eine solche Einheit wird als *activity* bezeichnet. WS-Coordination bildet ein erweiterbares Framework als Basis für Protokolle zur Sicherstellung konsistenter verteilter Zustandsübergänge im Rahmen einer *activity*. Die Durchführung solcher Tätigkeiten nimmt häufig eine lange Zeit in Anspruch, was einerseits auf die Benutzerinteraktion und andererseits auf die Latenz bei der Aufgabenbearbeitung zurückzuführen ist.

Die WS-Coordination-Spezifikation definiert daher dieses Framework, um die bestehende Komplexität von WebServices zu reduzieren.

Die Spezifikation kann somit als Grundlage für die Integration von Transaktionen in eine Web-Service-Umgebung dienen, sie ist jedoch ausreichend allgemein, um auch andere Formen der Koordination zu unterstützen, die eine wechselseitige Einigung aller Teilnehmer bezüglich des Ausgangs einer *activity* ermöglichen. Wie die Koordination der Teilnehmer letztlich abläuft ist hier jedoch nicht Gegenstand der Betrachtung. Für die Beschreibung konkreter Koordinationsmodelle, die als *coordination types* bezeichnet werden, wird vielmehr eine Basis geschaffen. Die ersten und bis jetzt einzigen beiden Beispiele hierfür sind WS-AtomicTransaction und WS-BusinessActivity.

1.3 WS-Coordination – Kontext und Model



Quelle: [2]

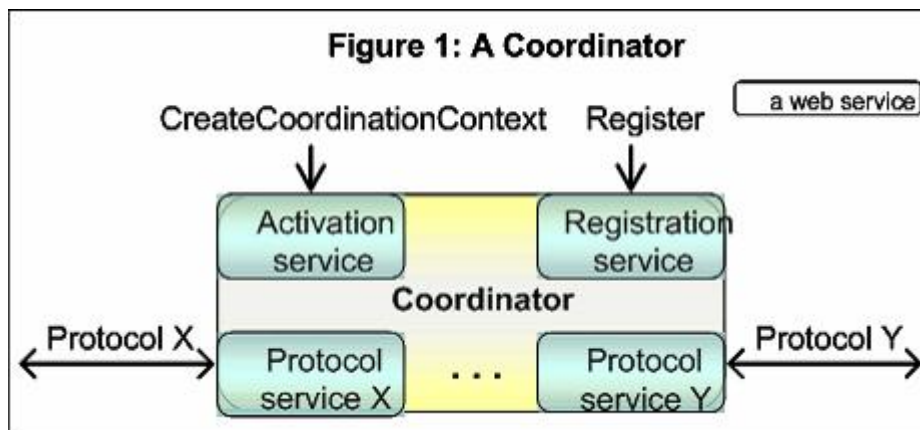
2. Coordination service

WS-Coordination beschreibt ein Framework für einen *coordination service* (oder einen coordinator). Dieser Service besteht aus folgenden Komponenten:

- Activation service – ermöglicht einer Applikation, die auch als Initiator bezeichnet wird, eine *activity* zu beginnen und veranlasst damit den *coordinator* eine neue Koordinationsinstanz und einen Koordinationskontext zu erzeugen
- Registration service – ermöglicht es einem Web Service sich für die Teilnahme an einer laufenden *activity* zu registrieren
- Satz von Koordinationsspezifischen Protokollen

Ein *coordination type* beinhaltet verschiedene *coordination protocols*; eine Menge wohldefinierter Nachrichtenformate und Regeln für den Austausch von konkreten Nachrichten zwischen dem Koordinator und den Teilnehmern einer *activity*. Die Arbeitsweise des *coordination protocol services* wird in der Spezifikation des verwendeten *coordination type* beschrieben.

Dies ist in der Graphik unten dargestellt:

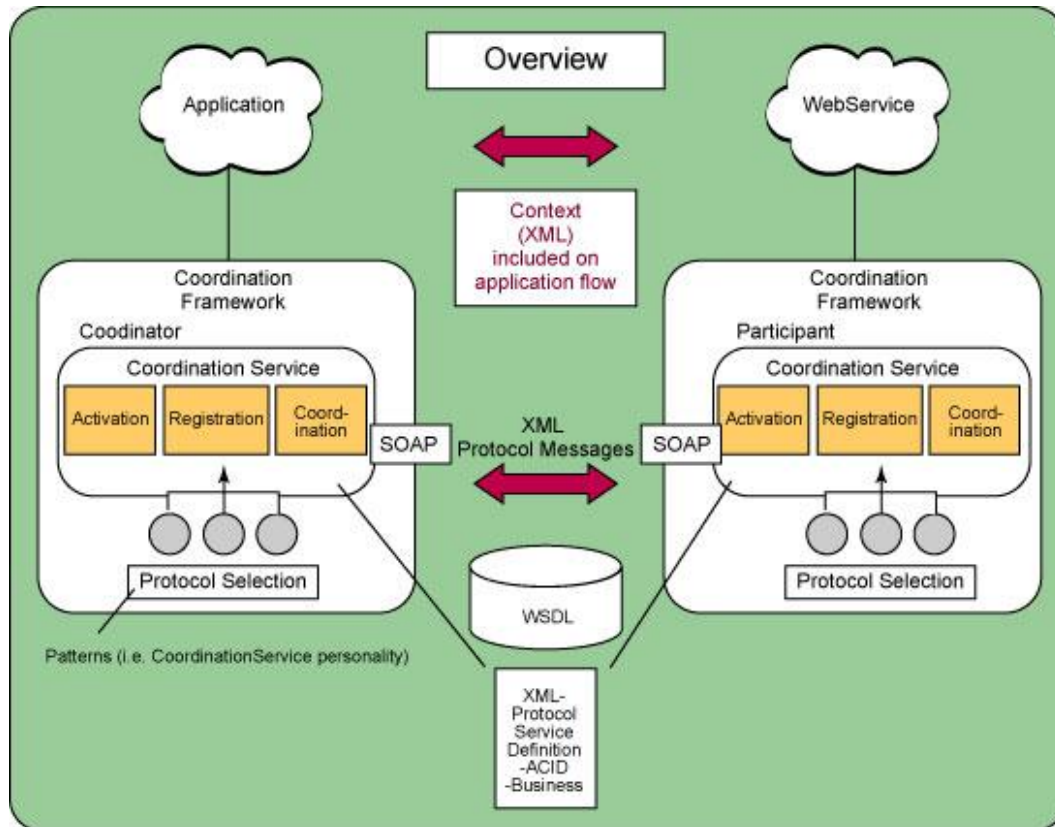


Applikationen benutzen den *activation service* um den *CoordinationContext* zu erstellen. Der *CoordinationContext* enthält die benötigten Informationen, um sich in einer *activity* zu registrieren. Dabei wird das Koordinationsverhalten spezifiziert, welchem die Applikation folgen wird. Eine Applikation, welche einen *CoordinationContext* empfängt, kann den Registrierungsservice der sendenden Applikation benutzen. Es steht ihr aber auch frei einen

eigenen Koordinator (Importieren einer activity – vgl. Kap. 3.5 Importieren einer activity) zu nutzen.

3. Komponenten WS-Coordination

3.1. Überblick WS-Coordination



Quelle: [3]

In der Übersicht sind die drei Komponenten (activation service, registration service, coordination service – vgl. Kap. 2. Coordination service) zu sehen, aus denen das Coordination Framework besteht. Die Operationen und Nachrichten der Services sind in WSDL definiert. Nachrichten werden in SOAP eingebunden.

3.2. CoordinationContext

Ein CoordinationContext wird von einer Applikation mittels der CreateCoordinationContext-Operation des *activation service* erzeugt. Der Kontext wird, eingebettet im Header-Element, mit allen SOAP-Nachrichten im Rahmen einer *activity* verschickt. An dieser Stelle wird kurz auf das Konzept der *endpoint reference* eingegangen, das Teil der WS-Addressing Spezifikation ist und in einem früheren Vortrag im Rahmen dieses Seminars bereits vorgestellt wurde. Zur Erinnerung: Eine *endpoint reference* ermöglicht die Nutzung eines Web Service, indem sie folgende Informationen bereitstellt:

- die URI eines Web Service
- beliebige zusätzliche Daten, unverständlich für jeden, außer für den Erzeuger selbst.

Diese Daten müssen in allen Nachrichten an den Web Service enthalten sein und werden in der Regel genutzt, um bestimmte Ressourcen in diesem anzusprechen.

Der CoordinationContext ist ein Kontexttyp der dazu benutzt wird Koordinationsinformation an die im *coordination service* involvierten Parteien zu vermitteln.

Die Übermittlung des Kontext mittels einer Applikationsnachricht wird allgemein als „flowing the context“ bezeichnet.

Ein CoordinationContext stellt Zugriff auf den *registration service*, einen Koordinationstypen (vgl. WS-AtomicTransaction, etc.) und relevante Erweiterungen bereit.

Das folgende Beispiel zeigt einen CoordinationContext, der einen Transaktionservice unterstützt ([4]):

```
<soap:Envelope xml-ns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    ...
    <wscoor:CoordinationContext
      xml-ns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
      xml-ns:wscoor="http://schemas.xmlsoap.org/ws/2003/09/wscoor"
      xml-ns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
      xml-ns:myApp="http://myApplication.de/myApp"

      soap:mustUnderstand="true">

      <wsu:Expires>
        2004-05-31T12:30:00Z
      </wsu:Expires>
      <wsu:Identifier>
        http://myCoordinator.de/activity1
      </wsu:Identifier>
      <wscoor:CoordinationType>
        http://schemas.xmlsoap.org/ws/2003/09/wsat
      </wscoor:CoordinationType>
      <wscoor:RegistrationService>
        <wsa:Address>
          http://myCoordinator.de/registrationService
        </wsa:Address>
        <wsa:ReferenceProperties>
          <TransactionId="efb01dc0-5073-405a-a3aea6038ecc476e">
        </wsa:ReferenceProperties>
        </wscoor:RegistrationService>
      </wscoor:CoordinationContext>
      <myApp:IsolationLevel>
        RepeatableRead
      </myApp:IsolationLevel>
      ...
    </soap:Header>
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelope>
```

- CoordinationContext/Expires

Dieses Element gibt einen TimeOut-Wert für den CoordinationContext an. Im Beispiel wäre dies der 30. Mai 2004, 12:30Uhr (Z-Zeit bzw. UTC). Dieser Wert hat je nach verwendetem coordination type unterschiedliche Bedeutung für den Koordinationsprozess.

- CoordinationContext/Identifier

Eine activity erhält bei ihrem Beginn einen Identifikator. So kann jede Nachricht eindeutig einer *activity* zugeordnet werden.

- CoordinationContext/CoordinationType

Gibt den *coordination type* an, der in der *activity* Verwendung findet. Dieser stellt ein bestimmtes Koordinationsmodell dar, er regelt das Verhalten aller Beteiligten sowie den Nachrichtenfluss zwischen ihnen über die Spezifikationen aus WS-Coordination hinaus. Im Beispiel handelt es sich um eine ACID-Transaktion, die in WS-AtomicTransaction spezifiziert ist und durch das Schlüsselwort <http://schemas.xmlsoap.org/ws/2003/09/wsat> identifiziert wird.

- CoordinationContext/RegistrationService

Wenn ein Web Service eine SOAP-Nachricht erhält, die einen CoordinationContext aufweist, muss er sich zunächst beim Koordinator der *activity* als Teilnehmer registrieren.

Dazu verwendet er die hier angegebene *endpoint reference*. Im Beispiel würde der Web Service eine register-Nachricht an <http://myCoordinator.de/registrationService> senden (siehe hierzu Kap. 3.4 Registration service). Um die Transaktion zu identifizieren, für die sich der Dienst registrieren will, wird im Beispiel eine TransactionId eingesetzt. Diese Information ist nur für den Koordinator selbst von Relevanz, der Web Service kopiert diese blind in die Header der Protokollnachrichten, die an den Koordinator gerichtet sind. Der Einsatz einer TransactionId ist nicht in WS-Coordination vorgesehen, in der Praxis bietet sich dieses Verfahren jedoch an.

Um zu verhindern, dass sich ein Web Service blind an einer *activity* beteiligt, deren *coordination type* nicht von ihm unterstützt wird, muss das mustUnderstand-Attribut gesetzt sein.

- CoordinationContext/{any}

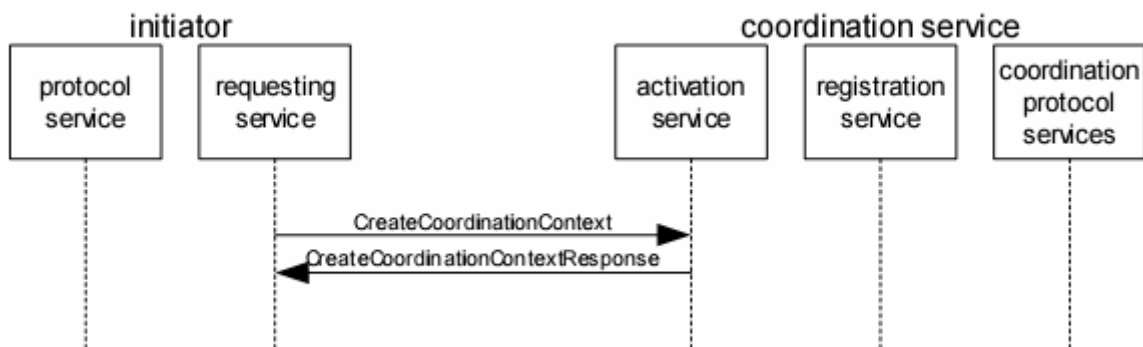
Es ist möglich, applikationsspezifische Informationen im CoordinationContext zu übermitteln. Im Beispiel wird ein zusätzliches IsolationLevel-Element verwendet.

- CoordinationContext/@{any}

Auch der Einsatz zusätzlicher applikationsspezifischer Attribute im CoordinationContext ist erlaubt. Im Beispiel wird hiervon kein Gebrauch gemacht.

3.3. Activation service

Um eine neue *activity* zu beginnen, sendet eine Applikation eine CreateCoordinationContext-Nachricht an den *activation service* eines ihr bekannten Koordinators. Diese Applikation wird als Initiator bezeichnet. Der Koordinator erzeugt einen CoordinationContext und liefert diesen mittels einer CreateCoordinationContextResponse-Nachricht an die Applikation zurück. Dies ist in der untenstehenden Grafik dargestellt:



3.3.1. CreateCoordinationContext

Eine CreateCoordinationContext-Nachricht ist nach folgendem Schema aufgebaut:

```
<CreateCoordinationContext ...>
  <CoordinationType> ... </CoordinationType>
  <wsu:Expires> ... </wsu:Expires>
  <CurrentContext> ... </CurrentContext>
  ...
</CreateCoordinationContext>
```

- CreateCoordinationContext/CoordinationType

Der *coordination type*, der in der *activity* verwendet werden soll, wird durch einen eindeutigen Identifikator beschrieben. Im Fall von WS-BusinessActivity wäre diese beispielsweise <http://schemas.xmlsoap.org/ws/2004/01/wsba>.

- CreateCoordinationContext/wsu:Expires (optional)

Dieses Element gibt einen TimeOut-Wert für den CoordinationContext an. Die Semantik dieses Parameters ist je nach verwendetem *coordination type* unterschiedlich.

- CreateCoordinationContext/CurrentContext (optional)

Wenn dieses Element nicht vorkommt, wird ein neuer CoordinationContext erzeugt. Wird hingegen der CoordinationContext einer laufenden *activity* übergeben, beteiligt sich der Koordinator an dieser. Dieser Vorgang wird als Importieren einer *activity* bezeichnet (vgl. Kap. 3.5 Importieren einer *activity*).

Die WS-Coordination-Spezifikation schlägt den Einsatz des CurrentContext-Elements auch bei Recovery vor, bleibt jedoch eine genauere Erläuterung dieses Aspektes schuldig. Die Verwendung beliebiger zusätzlicher Elemente und Attribute zum Versenden weiterer Informationen ist möglich.

3.3.2. CreateCoordinationContextResponse

Die CreateCoordinationContext-Nachricht wird vom ActivationService mit einer CreateCoordinationContextResponse-Nachricht beantwortet, nachdem der Kontext erzeugt wurde. In dieser Nachricht ist der erzeugte Kontext enthalten.

Der Aufbau der Nachricht sieht folgendermaßen aus:

```
<CreateCoordinationContextResponse ...>
  <CoordinationContext> ... </CoordinationContext>
  ...
</CreatCoordinationCotextResponse>
```

Das einzig geforderte Element enthält den erzeugten CoordinationContext. Dessen Aufbau wurde bereits in Abschnitt 3.2 erläutert. Einziger Unterschied ist, dass der Kontext hier im Body einer SOAP-Nachricht versendet wird, nicht im Header.

Die Verwendung von beliebigen zusätzlichen Elementen und Attributen zum Versenden weiterer Informationen ist ebenfalls möglich.

Ein komplettes Beispiel ist in der folgenden Abbildung zu sehen:

```
<CreateCoordinationContextResponse>
  <CoordinationContext>
    <Identifier>
      http://Business456.com/tm/context1234
    </Identifier>
    <CoordinationType>
      http://schemas.xmlsoap.org/ws/2004/10/wsat
    </CoordinationType>
    <RegistrationService>
      <wsa:Address>
        http://Business456.com/tm/registration
      </wsa:Address>
      <wsa:ReferenceProperties>
        <myapp:PrivateInstance>
          1234
        </myapp:PrivateInstance>
      </wsa:ReferenceProperties>
    </RegistrationService>
  </CoordinationContext>
</CreateCoordinationContextResponse>
```

3.4. Registration service

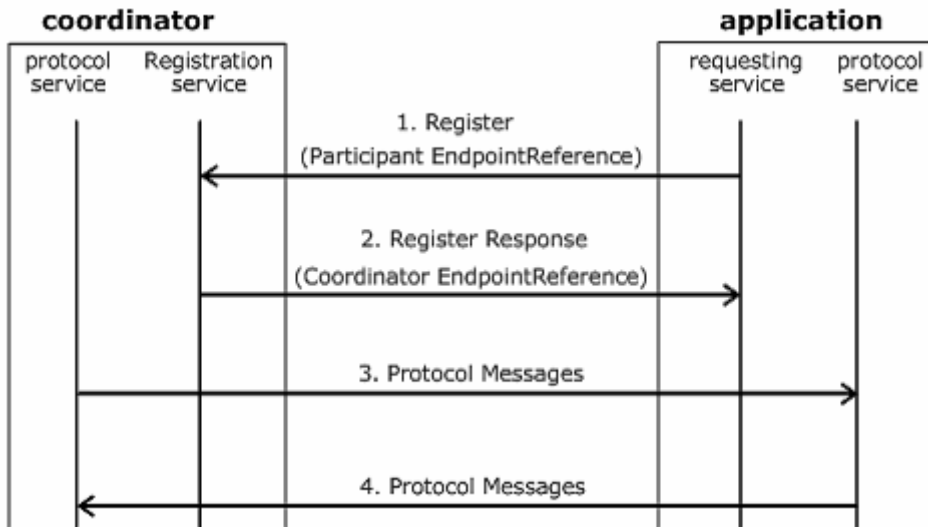
Sobald der Initiator eine *activity* begonnen hat, die *CreateCoordinationContext*-Nachricht gesendet und den Response mit dem Kontext empfangen hat, kann er sich selbst in der *activity* als Teilnehmer registrieren. Dies geschieht, indem der Initiator aus dem Koordinationskontext die *endpoint reference* auf den *registration service* des Koordinators ausliest und an diese URI eine register-Nachricht sendet. Die Syntax und Semantik wird im nächsten Teilabschnitt erläutert werden.

Der Initiator hat die Möglichkeit sich für mehrere Protokolle zu registrieren, indem er die register-nachricht mehrmals sendet.

Der *registration service* des Koordinators antwortet auf die register-nachricht mit einem *registration response*. Der Koordinator gibt in einer register response-nachricht eine *endpoint*

reference zurück. An diese URI sollen ab sofort alle Protokollnachrichten der Applikation versendet werden.

Die folgende Graphik verdeutlicht die Nutzung von *endpoint references* während der Registrierung:



In der obigen Abbildung ist dargestellt, wie der *coordinator* die *registration endpoint reference* im *CoordinationContext* während der *CreateCoordinationContext*-Operation zur Verfügung stellt. Die Applikation empfängt die *endpoint reference* in einer Applikationsnachricht.

1. Die Registernachricht adressiert diese *endpoint reference* und bindet die Protokollservice-Endpoint-Reference der Applikation in die Nachricht ein.
2. Die Antwort darauf enthält die *endpoint reference* auf den Protokollservice des Koordinators.
3. & 4. Nun haben beide Parteien die jeweiligen Referenzen auf die Protokollservices des anderen Teilnehmers und können diese, beim Versand protokollspezifischer Nachrichten im weiteren Verlauf der *activity*, direkt adressieren.

3.4.1. Register Nachricht

Die Registrierungsanfrage wird folgenderweise verwendet:

- Teilnehmerauswahl und Registration durch den *coordination service*, für ein bestimmtes Koordinationsprotokoll, welches der gegenwärtige *coordination type* unterstützt
- Austausch der *endpoint references* durch beide Seiten (sowohl Initiator, als auch Koordinator)

Der Aufbau einer Register-Nachricht ist in diesem XML-Fragment dargestellt:

```
<Register ...>
  <ProtocolIdentifier> ... </ProtocolIdentifier >
  <ParticipantProtocolService> ... </ParticipantProtocolService>
  ...
</Register>
```

- Register/ProtocolIdentifier
Die Applikation entscheidet sich für ein bestimmtes *coordination protocol*, das vom *coordination type* der *activity* unterstützt wird und übergibt dessen Identifikator. Im Fall des *completion protocol* wäre das beispielsweise folgende URI:
<http://schemas.xmlsoap.org/ws/2003/09/wsat#Completion>.
- Register/ParticipantProtocolService
Die Applikation übergibt außerdem eine *endpoint reference*, an die der Koordinator künftige Protokollnachrichten senden soll.

Beispiel für eine Registernachricht:

```
<Register>
  <ProtocolIdentifier>
    http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC
  </ProtocolIdentifier>
  <ParticipantProtocolService>
    <wsa:Address>
      http://Adventure456.com/participant2PCservice
    </wsa:Address>
    <wsa:ReferenceProperties>
      <BetaMark> AlphaBetaGamma </BetaMark>
    </wsa:ReferenceProperties>
  </ParticipantProtocolService>
</Register>
```

3.4.2. RegistrationResponse Nachricht

Die Antwort auf einer Registernachricht enthält die *endpoint references* des Koordinators.

Diese hat folgenden Aufbau:

```
<RegisterResponse ...>
  <CoordinationProtocolService> ... </CoordinationProtocolService>
  ...
</RegisterResponse>
```

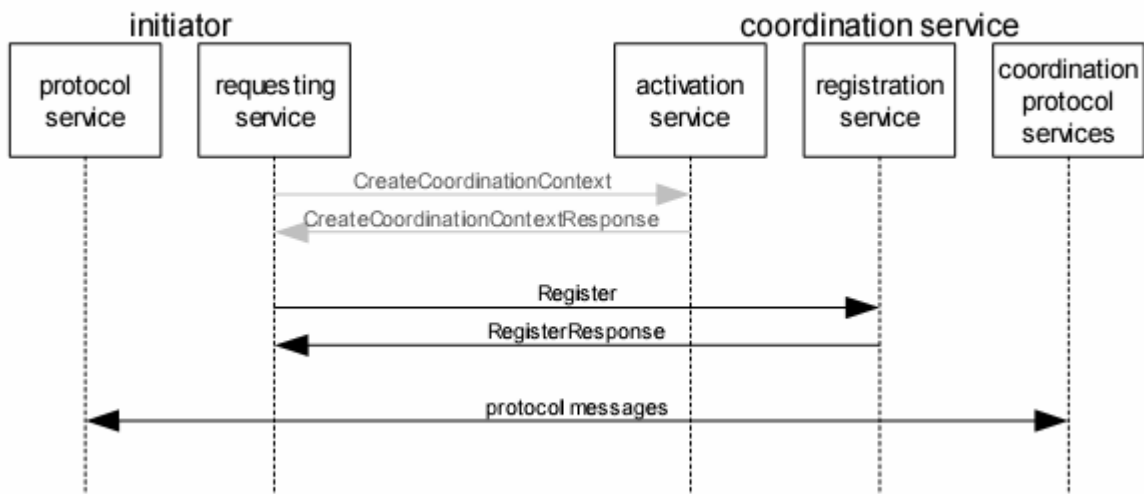
- RegisterResponse/CoordinatorProtocolService

Die vom *coordination service* gewählte *endpoint reference*, die der Teilnehmer (die Applikation) für das Koordinations-Protokoll verwenden soll

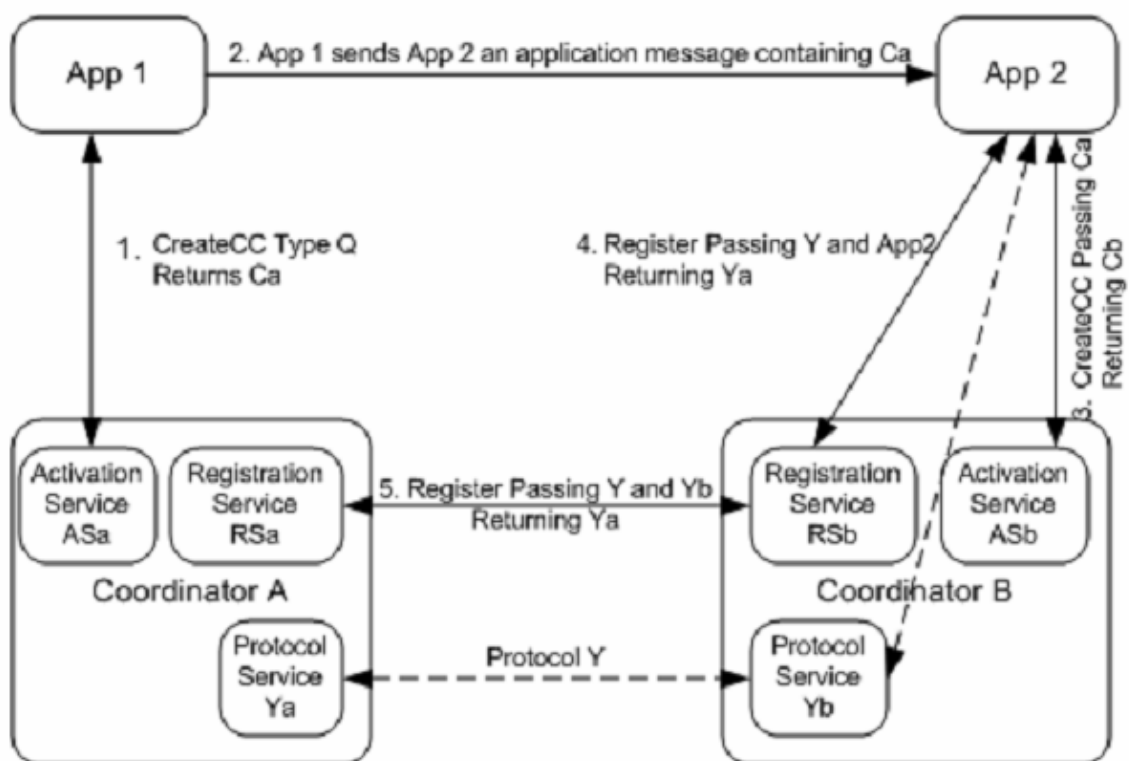
Beispiel einer RegistrationResponseMessage:

```
<RegisterResponse>
  <CoordinatorProtocolService>
    <wsa:Address>
      http://Business456.com/mycoordinationsservice/coordinator
    </wsa:Address>
    <wsa:ReferenceProperties>
      <myapp:MarkKey> %%F03CA2B%% </myapp:MarkKey>
    </wsa:ReferenceProperties>
  </CoordinatorProtocolService>
</RegisterResponse>
```

Der gesamte Vorgang, inklusive der Kontexterstellung, wird in folgender Grafik dargestellt:



3.5. Importieren einer activity



Der *activation service* kann genutzt werden, um einen Koordinator in eine bestehenden *activity* einzubinden. Dieser Vorgang wird „importieren“ oder „interposing“ genannt. Erhält ein Web Service eine Applikationsnachricht, die einen *CoordinationContext* beinhaltet, kann er sich beim Koordinator registrieren, der diesen erzeugt hat. Es gibt jedoch auch Gründe,

einen eigenen Koordinator heranzuziehen, da dieser beispielsweise eine höhere Leistungsfähigkeit aufweist oder als vertrauenswürdiger gilt.

Die Grafik zeigt zwei Applikationen (App1 und App2), die jeweils einen eigenen Koordinator benutzen (CoordinatorA und CoordinatorB), während sie miteinander agieren. Die Protokolle Ya und Yb gehören zu einem bestimmten *coordination type*, welcher hier nicht genauer definiert wird.

Der Ablauf zum importieren einer *activity* gestaltet sich im Einzelnen wie folgt:

1. App1 sendet eine *CreateCoordinationContext* für *coordination type* Q und erhält den Kontext Ca. Dieser enthält A1 (Identifiziert die *activity*), den *coordination type* Q und eine *endpoint reference* auf CoordinatorA's *Registration service* RSa.
2. App1 sendet daraufhin eine *application message* an App2, welche den Kontext Ca enthält.
3. App2 bevorzugt allerdings CoordinatorB, daher nutzt App2 den *CreateCoordinationContext* mit Ca als optionalem „currentContext-Element“ um CoordinatorB zu importieren. CoordinatorB erzeugt daraufhin den *CoordinationContext* Cb, der sich lediglich durch die *endpoint reference* auf seinen eigenen *registration service* RSb von Ca unterscheidet.
4. App2 bestimmt die *coordination protocols* welche vom *coordination type* Q unterstützt werden und registriert sich anschließend für ein *coordination protocol* Y bei CoordinatorB. Dabei werden die *endpoint references* für App2 und der *protocol service* Yb ausgetauscht. Dadurch entsteht eine logische Verbindung zwischen den *endpoint references* von App2 und CoordinatorB die Protokoll Y nun benutzen kann.
5. Diese Registration zwingt CoordinatorB dazu, die Registration an CoordinatorA's *registration service* RSa weiterzuleiten und dabei die *endpoint references* für Yb und den *protocol service* Ya auszutauschen. CoordinatorB registriert sich bei CoordinatorA sozusagen als Stellvertreter für App2. Dadurch entsteht eine logische Verbindung zwischen den beiden *endpoint references*, welche Protokoll Y verwenden kann.

Es entsteht eine neue Protokoll Instanz, in der A die Rolle des Koordinators und B die Rolle eines Teilnehmers zukommt.

Koordinator B wird nun als *interposed* oder *subordinate coordinator* bezeichnet. Durch den

beschriebenen Mechanismus ist die Bildung hierarchischer Strukturen beliebiger Tiefe möglich. Ein Koordinator kann hierbei auch mehrere *subordinate coordinators* unter sich haben.

3.6 Coordination Faults

WS-Coordination faults müssen als [action] Eigenschaft folgende *fault action* URI einbinden:

<http://schemas.xmlsoap.org/ws/2004/10/wscoor/fault>

Die Definitionen der Störungen in diesem Abschnitt besitzen folgende Eigenschaften:

[Code] Code der Störung.

[Subcode] Subcode der Störung.

[Reason] Der Grund der Störung (in Englischer Sprache)

[Detail] Detaillierte Angaben zum aufgetretenen Fehler. Sollte dieses Element nicht definiert werden, dann wird kein Detail-Element angelegt.

Bei SOAP 1.2, muss [Code] entweder "Sender" oder "Receiver" sein. Diese Eigenschaften werden in text XML in dieser Reihenfolge wie folgt dargestellt:

SOAP Version	Sender	Receiver
SOAP 1.2	S:Sender	S:Receiver

In der folgenden Grafik wird diese Darstellung in SOAP 1.2 dargestellt:

```
<S:Envelope>
  <S:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/10/wscoor/fault
    </wsa:Action>
    <!-- Headers elided for clarity. -->
  </S:Header>
  <S:Body>
    <S:Fault>
      <S:Code>
        <S:Value>[Code]</S:Value>
        <S:Subcode>
          <S:Value>[Subcode]</S:Value>
        </S:Subcode>
      </S:Code>
      <S:Reason>
        <S:Text xml:lang="en">[Reason]</S:Text>
      </S:Reason>
      <S:Detail>
        [Detail]
        ...
      </S:Detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

Während die Darstellung in SOAP 1.1 so aussieht:

```
<S11:Envelope>
  <S11:Body>
    <S11:Fault>
      <faultcode>[Subcode]</faultcode>
      <faultstring xml:lang="en">[Reason]</faultstring>
    </S11:Fault>
  </S11:Body>
</S11:Envelope>
```

Die einzelnen Fehlermeldungen im Detail:

- Invalid State

Diese Fehlermeldung wird entweder vom Koordiantor oder einem Teilnehmer gesendet, um anzuzeigen, dass die *endpoint reference*, die den Fehler gemeldet hat, eine Nachricht bekommen hat, die für den momentanen Status ungültig ist.

Dies ist ein unbehebbarer Zustand.

Eigenschaften:

[Code] Sender

[Subcode] wscoor:InvalidState

[Reason] The message was invalid for the current state of the activity.

[Detail] unspecified

- Invalid Protocol

Diese Fehlermeldung wird entweder vom Koordiantor oder von einem Teilnehmer gesendet, um anzuzeigen, dass die *endpoint reference*, die den Fehler erzeugt hat, eine Nachricht von einem ungültigen Protokoll empfing.

Dies ist ein unbehebbarer Zustand.

Eigenschaften:

[Code] Sender

[Subcode] wscoor:InvalidProtocol

[Reason] The protocol is invalid or is not supported by the coordinator.

[Detail] unspecified

- Invalid Parameters

Diese Störung wird entweder vom Koordinator oder von einem Teilnehmer gesendet, um anzuzeigen, dass die *endpoint reference* die den Fehler erzeugt hat unzulässige Parameter auf oder innerhalb einer Nachricht empfing. Dies ist ein unbehebbarer Zustand.

Eigenschaften:

[Code] Sender

[Subcode] wscoor:InvalidParameters

[Reason] The message contained invalid parameters and could not be processed.

[Detail] unspecified

- No Activity

Diese Störung wird vom Koordinator gesendet, wenn der Teilnehmer für zu lange inaktiv gewesen ist und vermutet wird, dass er die *activity* beendet hat.

Eigenschaften:

[Code] Sender

[Subcode] wscoor:NoActivity

[Reason] The participant is not responding and is presumed to have ended.

[Detail] unspecified

- Context Refused

Einem Koordinator wird diese Störung geschickt, um anzuzeigen, dass die *endpoint reference* einen Kontext nicht annehmen kann, den er übergeben bekam.

Eigenschaften:

[Code] Sender

[Subcode] wscoor:ContextRefused

[Reason] The CoordinationContext that was provided could not be accepted.

[Detail] unspecified

- Already Registered

Einem Teilnehmer wird diese Störung geschickt, wenn der Koordinator

feststellt, dass der Teilnehmer versucht hat, sich für das gleiche Protokoll der gleichen Tätigkeit mehr als einmal zu registrieren.

Eigenschaften:

[Code] Sender

[Subcode] wscoor:AlreadyRegistered

[Reason] The participant has already registered for the same protocol.

[Detail] unspecified

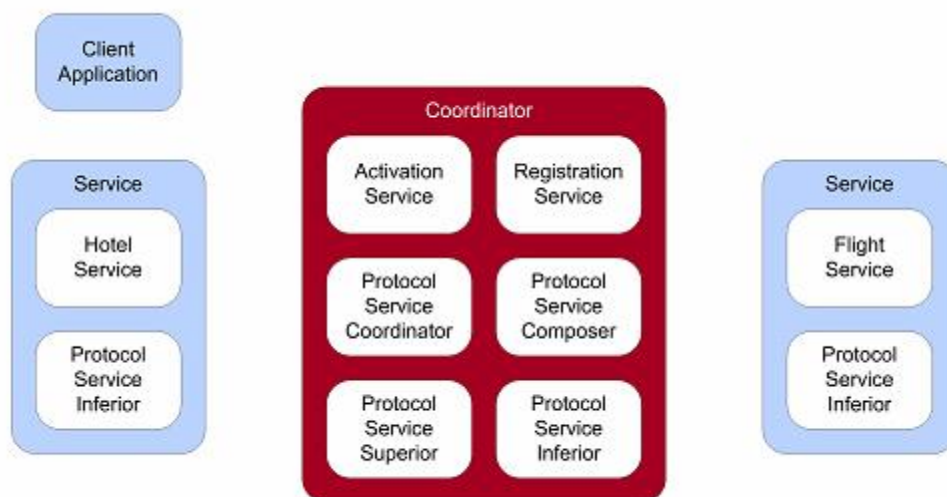
4 Beispiel

In WS-Coordination können ACID-Transaktionen und Business-Transaktionen kombiniert werden. Die Grundlage dafür ist, dass eine Interposition (s. Kap. 3.5) erzeugt wird. Es müssen mindestens zwei Koordinatoren existieren, wovon einer ein Sub-Koordinator des anderen ist.

Beim Koordinator wird eine WS-BusinessActivity erzeugt, beim Sub-Koordinator wird eine WS-AtomicTransaction erzeugt. Da die beiden Koordinatoren miteinander kommunizieren müssen, existiert folgendes Problem. Beide Koordinatoren enthalten verschiedene Protokolle (WS-BA-Protokolle und WS-AT-Protokolle), das heißt sie sprechen nicht die gleiche Sprache. Damit sie miteinander kommunizieren können, muss der Sub-Koordinator Protokollnachrichten übersetzen können. Wie er Nachrichten übersetzen muss, ist in WS-Coordination nicht spezifiziert.

Für das Beispiel nehmen wir an, es seien ein Koordinator, zwei Services und ein Client implementiert wurden. Der erste Service entspricht einer Hotelbuchung, welcher die Operation `bookHotel` implementiert. Der zweite Service repräsentiert eine Flugbuchung, welche die Operation `bookFlight` implementiert. Mit dem Client können diese Services sowie der Koordinator angesprochen werden. Der Activation Service und der Registration Service entsprechen der Implementierung von WS-Coordination. Der Activation Service implementiert die Operation `createCoordinationContext` (vgl. Kap. 3.2), der Registration Service die Operation `register` (vgl. Kapitel 3.4). Die Protocol Services entsprechen der

Implementierung vom Business Transaction Protocol (s. BTP-Spezifikation von OASIS). Der Protocol Service Coordinator repräsentiert das CompletionProtokoll von bpm-atom (s. BTP-Spezifikation von OASIS), der Protocol Service Composer das Completion-Protokoll von bpm-cohesion (s. BTP-Spezifikation von OASIS). Die Services Superior und Inferior benutzen die CoordinationTypes gemeinsam. Der Superior-Service entspricht der Implementierung des 2-Phasen-Protokolls vom BTP (vgl. Tobias Ramin – Transaktionen in WebServices). Er enthält unter anderem die Operationen prepared, cancelled und confirmed. Der Inferior-Service erweitert diesen Service, um die Sub-Koordinator-Funktionalität anbieten zu können. Er erweitert den Superior-Service unter anderem um die Operationen prepare, cancel und confirm. Der Aufbau der Komponenten (Hotelservice, Flugbuchungsservice, Coordinator) ist in folgendem Schaubild dargestellt.



Die Kommunikation zwischen einem Client und dem Activation Service oder dem Registration Service kann synchron oder asynchron erfolgen. Die WS-Coordination Services implementieren sowohl die synchronen als auch die asynchronen PortTypes. Die Services Hotelbuchung und Flugbuchung sowie die Protokoll-Services von BTP werden asynchron aufgerufen. Für die asynchronen Aufrufe werden WS-Addressing (vgl. Vortrag zu WS-Addressing) Message-Headers verwendet. Die Request-Messages enthalten einen eindeutigen Message-Identifizierer und ein ReplyTo-Element (vgl. Vortrag zu WS-Addressing). Die Response-Messages enthalten ein RelatesTo-Element. Zur Erinnerung: Damit weiß ein

Service, an welche Adresse die Antwort gesendet werden soll und der Requester kann die Antwort einem Request zuordnen.

Für die Antworten asynchroner Services implementiert die Client-Anwendung die Services *ActivationRequester*, *HotelBookingRequester* und *FlightBookingRequester*. Der Hotel-Service implementiert als *Endpointreference* für die Antwort des Registration Service den Service *HotelBookingRegistrationRequester*. Der Flug-Service implementiert entsprechend den Service *FlightBookingRegistrationRequester*.

Mit diesen drei Komponenten könnte nun folgender Ablauf basierend auf *AtomicTransaction* (Koordinator <-> Services) und *BusinessTransaction* (Client <-> Koordinator) (AT und BT vgl. Tobias Ramin – Transaktionen in WebServices) stattfinden:

Der Client erzeugt beim Activation Service durch die Angabe des *CoordinationType* *btp-atom* einen Kontext. Der Client registriert sich für das Coordinator-Protokoll. Mit dem erzeugten Context im Message-Header ruft der Client den Hotel-Service und den Flug-Service auf.

Die Services registrieren darauf einen Teilnehmer für das Superior-Protokoll beim Registration Service.

Will der Client die Transaktion beenden, sendet er dem Coordinator-Service eine *confirmTransaction*-Nachricht. Darauf wird das 2-Phasen-Protokoll von BTP durchgeführt. Der BTP-Koordinator kommuniziert nur mit den registrierten *ServiceParticipants* (Hotel- und Flugbuchung). Nach Beendigung des Protokolls leitet der Koordinator das Ergebnis der Transaktion an den Client weiter. Im Beispiel will der Client einen Flug buchen und in einem bestimmten Hotel übernachten. Der Koordinator bekommt von dem Hotel-Service eine negative Antwort, von der Flugbuchung eine positive. Nehmen wir an, dass es sich um eine „all-or-nothing“-Transaktion handelt. Der Koordinator erwirkt daher dass auch der *FlightParticipant* eine *cancel*-Nachricht bekommt, da die Hotelbuchung nicht positiv abgeschlossen werden konnte. Das, insgesamt, negative Ergebnis des Buchungsvorganges wird dann an den Client weitergeleitet.

Handelt es sich um nicht um eine „all-or-nothing“-Transaktion, würde der Client den *CoordinationType* *btp-cohesion* für die Aktivierung verwenden. Anschließend registriert sich der Client für das Composer-Protokoll. Wie bei der Atom-Transaktion ruft der Client beide Buchungsdienste auf, welche daraufhin beim Registration Service jeweils einen *Superior*-Teilnehmer registrieren.

Sobald die Antworten der Buchungsdienste vorliegen, sendet die Client-Anwendung eine *prepareInferiors*-Nachricht, mit dem *Hotel-Participant* als Parameter, an den Composer. Der Koordinator schickt *prepare* an den *Hotel-Participant*, welcher darauf in den *prepared*-Zustand

wechselt. Der Client sieht, dass der Hotel-Participant prepared ist und schickt dem Composer eine confirmTransaction-Nachricht, welche nur den HotelParticipants Parameter enthält. Darauf erhält der Flug-Participant eine cancel-Nachricht vom Koordinator, der Hotel-Participant erhält eine confirm-Nachricht. In diesem Fall wird nur die Hotelbuchung durchgeführt. Das Ergebnis der Transaktion wird an den Client weitergeleitet.

Die genauen Abläufe der Transaktionen sind in der Seminararbeit von Tobias Ramin „Transaktionen in WebServices“ erläutert.

Quellenangabe:

[1]: <http://ftpna2.bea.com/pub/downloads/ws-standards-coordination.pdf>

[2]: www.bis.uni-leipzig.de/studium/vorlesungen/2004_ss/eb1/2004s_eb1_v_06a_1.pdf

[3]: <http://www-128.ibm.com/developerworks/library/ws-wstx1/>

[4]: <http://msdn.microsoft.com/ws/2003/09/wscoor/>

[5]: ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-2386/DIP-2386.pdf