• [ACKM04] Ch 7
• Docs from IBM website:
WS-coordination

- A framework for supporting coordination
- Standardizes:
  - passing GUID between interacting Web services
  - informing protocol handler about ports
  - informing protocol handler about roles
Infrastructur

B: conversation compliant with a business protocol
H: conversation compliant with an horizontal protocol
Coordinators and Participants

- Coordinators and participants
- Central or distributed coordinator

(a) Central coordination

(b) Distributed coordination
Terminology

- Coordination protocol
  - e.g. 2PC
- Coordination type
  - e.g. atomic transaction
- Coordination context
  - data structure identifying messages as belonging to a conversation
Interactions

• Activation
  • participant requests new context
• Registration
  • participant registers self with coordinator
• Protocol specific interactions
  • interactions specific to the protocol :-)!
• Operational messages
  • not involving coordinator
Activation

CreateCoordinationContext
- ... - coordination type - current context

CreateCoordinationContextResponse
- ... - coordination context - identifier - coordination type - registration service - ...
Registration

register
- ...
- protocol identifier
- participant protocol service

RegistrationRequestorPortType

RegistrationCoordinatorPortType

coordinator

registerResponse
- ...
- coordinator protocol service

Web service
Protocol-Specific

- Convention for protocol specific messages

```
Web service
```

```
XParticipantPortType
```

```
XCoordinatorPortType
```

```
protocol-specific messages
from participant to coordinator
```

```
protocol-specific messages
from coordinator to participant
```

```
coordinator
```

```
```
Central Coordinator

1. create CC
2. X1
3. register
4. protocol coordinator
5. operational message
6. register
7. protocol coordinator
Distributed Coordinator

1. create CC
2. X1
3. register
4. protocol coordinator
5. operational message

6. create CC
7. X2
8. register
9. register
10. protocol coordinator
Summary of WS-Coordination

- Defines needed SOAP extensions
- Metaprotocols
  - activation
  - registration
- Middleware for central or distributed coordination
WS-Transaction

- Extension/Use of WS-coordination
- Two flavors:
  - Atomic transaction
  - Business activity
Atomic Transaction - Ports

WS-Coordination interfaces

- ActivationCoordinatorPortType
- RegistrationCoordinatorPortType

WS-Transaction interfaces

- CompletionParticipantPortType
- CompletionWithAckParticipantPortType
- PhaseZeroParticipantPortType
- 2PCParticipantPortType
- OutcomeNotificationParticipantPortType

- CompletionCoordinatorPortType
- CompletionWithAckCoordinatorPortType
- PhaseZeroCoordinatorPortType
- 2PCCoordinatorPortType
- OutcomeNotificationCoordinatorPortType

WS-Transaction interfaces needed for chaining

WS-Coordination interfaces needed for chaining

RegistrationParticipantPortType
Atomic Transaction - Protocol

- **Web service A**
  - create CC
  - T1
  - register for Completion
  - completion coordinator
  - operational message

- **Web service B**
  - create CC
  - T2
  - register for PhaseZero
  - PhaseZero coordinator
  - register for PhaseZero
  - PhaseZero coordinator
  - register for 2PC
  - 2PC coordinator

- **coordinator Ca**
  - register for Completion
  - completion coordinator

- **coordinator Cb**
  - register for PhaseZero
  - PhaseZero coordinator
  - register for 2PC
  - 2PC coordinator

- **complete**
  - PhaseZero
  - PhaseZeroComplete
  - PhaseZeroComplete
Atomic Transaction - Protocol
Business Activities

- Business Agreement (with Complete)
  - participant initiates by informing the coordinator of its status:
    - exited/completed/faulted
  - Coordinator replies
    - close/complete/compensate/forget
Port Types

WS-Coordination interfaces

- ActivationCoordinatorPortType
- RegistrationCoordinatorPortType

WS-Transaction interfaces

- BusinessAgreementCoordinatorPortType
- BusinessAgreementWithCompleteCoordinatorPortType
- RegistrationParticipantPortType

WS-Transaction interfaces needed for chaining

WS-Coordination interfaces needed for chaining
Execution
Service Composition

- [ACKM04] Ch 8
A client engages in different conversations with several Web services. In general, these conversations may be regulated by different protocols, and each invoked Web service may not be aware that the client is invoking other Web services.

The internal business logic of clients and Web services is quite sophisticated, as it must support the execution of different conversations so that each party can properly interact with every other party.
Hierarchy of Composite Services

Customer

- supply chain
  - inventory planning
    - ...
    - ...
  - accounting
  - procurement
    - approval
  - supplier
  - another supplier
Most Common Approach

Customer

- supply chain
  - inventory planning
  - accounting
  - procurement
    - approval

supplier

another supplier

Implement “root” of composite service in conventional high-level language
The internal application implements the composition logic, by invoking Web services as needed. No support is provided by the Web services middleware in this case.
Web Services Composition Middleware

- Composition Model and Language
  - Composition Schema specifies business logic
- Development Environment
  - GUI tools for development and deployment
- Run Time Environment
  - A composition engine
Web Services Composition Middleware

- Service composition model and language (usually characterized by a graphical and a textual representation)
- The run-time environment executes the Web service business logic by invoking other services (through SOAP and HTTP modules)

Web service composition middleware

- Development environment
- Run-time environment (composition engine)
- Other Web services middleware (e.g., SOAP engine and conversation controller)

- Services offered by other providers
  - Supplier
  - Warehouse
  - Accounting

- A service provider

- Schema designer

- Services offered by other providers
  - Supplier
  - Warehouse
  - Accounting
The composite service is directly implemented at the middleware level, by the composition engine.
Contrasting the two schemes...

- Lower tiers not involved in composition...
the procurement business protocol executed among Web services

Contrast - Composition vs Coordination

if the supplier is implemented by means of composition technologies, then its business logic is defined by a composition schema and its execution is driven by a composition engine.

1: requestQuote
2: orderGoods
3: confirmOrder
4: makePayment

depending on the implementation of the (composite) service, the supplier may contact other Web services. The customer is unaware of these interactions, that may occur according to other protocols.

another Web service, possibly offered by another company

yet another Web service

the Conversation controller

CIS
Potential for Success?

- More likely than traditional WfMS ...
- Because all services have uniform WS definitions
- Analogy: relational data model
Composition Models

- Dimensions of Composition Model:
  - Component
  - (Web Service)
  - Orchestration
    - activity diagram, Petri net, pi-calculus, ...
  - Data
    - specification and exchange
Composition Models - II

• Service Selection
  • static or dynamic
• Transactions
  • transactional semantics of composition
• Exception Handling
  • abortion and recovery
Component Model - How General?

- WSDL
  - restrict applicability
  - easy implementation
- XML messaging
  - more general
  - harder integration
- Adapters
  - we have been here ...
Orchestration Models

• Illustration:
  • Customer places order with supplier
  • Supplier implemented using composition
Orchestration Models

• Activity Diagram ...

```
receive orderGoods
invoke checkLocalStock
invoke checkShipAvailable
send confirmOrder
inStock=false
send cancelOrder
inStock=true
shippingAvail=true
shippingAvail=false
receive orderGoods
invoke checkShipAvailable
```

- Customer
- Local service offered by the supplier
- Warehouse
Orchestration Models

- Activity Hierarchy ...

```
process order
  sequence
  receive orderGoods
  invoke checkLocalStock
  discriminate based on local search
    inStock=false
      search external supplier
        sequence
        invoke checkShipAvailable
        discriminate based on external search
          shippingAvail=true
            send confirmOrder
          shippingAvail=false
            send cancelOrder
    inStock=true
      send confirmOrder
```
Orchestration Models

- Statecharts

[condition]/action
Orchestration Models

- Petri Nets

START (upon invocation of orderGoods operation)

invoke checkLocalStock

LOCAL SYSTEM ACCESSED

inStock=true

invoke checkShipAvailable

Do nothing inStock=false

EXTERNAL SUPPLIER ACCESSED

shippingAvail=false

send cancelOrder

READY TO SEND CONFIRMATION

shippingAvail=true

Do nothing

COMPLETE (CANCEL)

send confirmOrder

COMPLETE (CONFIRM)
Orchestration Models

- Pi-calculus

\[A ::= B . C \text{ (sequential)}\]
\[\text{or } B | C \text{ (parallel)}\]
\[\text{or } B + C \text{ (choice)}\]
\[\text{or } [\text{var}=\text{value}] B \text{ (condition)}\]

\[A = \text{receiveOrderGoods . invokeCheckLocalStock}\]
\[B = [\text{shippingAvail=false}] \text{sendCancelOrder}\]
\[\quad + [\text{shippingAvail=true}] \text{sendConfirmOrder}\]
\[C = \text{invokeCheckShipAvail . B}\]

Procurement = A . ( ([\text{inStock=false}]C)
\[\quad + ([\text{inStock=true}] \text{sendConfirmOrder}) ) \]
Orchestration Models

- Event-Condition-Action rules
- Analogous to triggers, active databases

```plaintext
ON receive orderGoods
 IF true
 THEN invoke checkLocalStock;

ON complete(checkLocalStock)
 IF (inStock==true)
 THEN send confirmOrder;

ON complete(checkLocalStock)
 IF (inStock==false)
 THEN invoke checkShipAvailable;

ON complete(checkShipAvailable)
 IF (shippingAvail ==true)
 THEN send confirmOrder;

ON complete(checkShipAvailable)
 IF (shippingAvail ==true)
 THEN send cancelOrder;
```
Data

• Data Types
  • Control flow data (simple)
  • Application data
    • “black box” - URLs (simple, opaque)
    • typed data / XML (inefficient, blobs)
### Data Transfer

<table>
<thead>
<tr>
<th>custName</th>
<th>&quot;Homer Smith&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>partNum</td>
<td>123-456-98765</td>
</tr>
<tr>
<td>quantity</td>
<td>3</td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
</tbody>
</table>

**Blackboard**

Blackboard is more “intuitive”

**Data Flow**

Data Flow makes control dependencies and data races explicit
Service Selection

- Static binding
  - service at fixed URL
- Dynamic binding by reference
  - service URL in a process variable
- Dynamic binding by lookup
  - look up service URL in directory
- Dynamic operation selection
  - like procedure valued variable
Newly added node that accesses a (statically specified) UDDI registry to determine which warehouse should be contacted for the product being ordered. The result is stored into the warehouse process variable. Note that in practice several invocations of the UDDI API may be needed to get the desired information.

Subsequent nodes can use the reference approach and determine the URI based on the value of the warehouse variable.

Variables:
- warehouse: URI
- inStock, shippingAvail: bool
- customer: String
...

supplier

- receive orderGoods
- invoke checkLocalStock
  - inStock=false
    - invoke get_bindingDetail
    - invoke checkShipAvailable
      - shippingAvail=false
        - send cancelOrder
      - shippingAvail=true
        - send confirmOrder
Transactions

• ACID transactions
• Sagas (compensation)
ACID Transactions

atomic region
Sagas

Long-lived transaction $T$ (saga)

Forward execution

Compensation flow

Subtransaction $S_1$

Subtransaction $S_2$

...$

Subtransaction $S_n$

Compensating Subtransaction $CS_1$

Compensating Subtransaction $CS_2$

...$

Compensating Subtransaction $CS_n$
Sagas

- Not highly successful in past
- Web Services trend: service responsible for
  - transaction
  - compensation
- (but note WS-Transaction compensates all-at-once rather than in reverse sequence)
Exceptions

- Flow-based
  - test return values and write code ...
- Try-catch-throw
  - e.g. Java
- Rule-based
  - ECA
• Already said: composition is private, coordination is public ...

• So: must design composed services that can participate in coordination protocols
Procurement Coordination Protocol

customer

requestQuote (to supplier)

orderGoods (to supplier)

makePayment (to supplier)

confirmShipment (to warehouse)

supplier

checkShipAvailable (to warehouse)

warehouse confirms

warehouse cancels

confirmOrder (to customer)
cancelOrder (to customer)

orderShipment (to warehouse)

warehouse

getShipmentDetails (to customer)

confirmShipment (to supplier)

warehouse

confirmShipment (to supplier)
Supplier View

customer

- requestQuote (to supplier)
- orderGoods (to supplier)
- makePayment (to supplier)

supplier

- checkShipAvailable (to warehouse)
- warehouse confirms
  - confirmOrder (to customer)
- warehouse cancels
  - cancelOrder (to customer)
- orderShipment (to warehouse)

warehouse

- confirmShipment (to supplier)
Translate Operations to Activities

- Request/Reply from R -> Receive/Reply
- Send from R -> Receive
- Send to R -> Send
- Request/Reply to R -> Invoke
Result of Translation

invoke checkShipAvailable

receive requestQuote

reply requestQuote

receive orderGoods

send confirmOrder

send cancelOrder

receive makePayment

send orderShipment

receive confirmShipment
From Protocol to Executable Process

1. receive requestQuote
2. invoke lookupQuote
3. reply requestQuote
4. receive orderGoods
5. invoke checkShipAvailable
   - shippingAvail=false: send cancelOrder
   - shippingAvail=true: send confirmOrder, receive makePayment, invoke collectPayment, send orderShipment, receive confirmShipment