Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

# Chapter 7 - Web Service Composition and E-Business Collaboration

# Motivation

- Complex web services
  - Need to interact with business partners through web services
  - May combine/utilize existing web services
- Web services composition
  - Ability to create new web services out of existing (web service) components
  - Requirements similar to BPM, Workflow Management
    - separate function from composition logic, …
- Composition can be iterated
  - Composition result is again a web service
  - Can be used as a building block for further composition steps

⇨ Middleware for web service composition

# Web Services Composition Middleware

- Main elements
  - composition model and language
    - composed WS is expressed by a composition schema (script)
  - development environment
    - graphical end user tools
  - run-time environment
    - composition "engine"
- Composition vs. coordination middleware
  - composition: focus is on implementation of operations in a web service
    - internal, private
    - for automation of the execution of a composite web service
  - coordination: focus is on conversation protocols
    - public, standardized protocols
    - external coordination for verifying compliance

# Web Services vs. WFMS

- Limitations of conventional composition middleware (e.g., WFMS)
  - Significant effort to integrate existing applications
    - application-specific adapters, wrappers
    - no standard model for component description, interoperability
  - Limited success of composition model standardization
    - WfMC standard is not widely implemented
- Opportunities for Web Services
  - Web Services seem to be adequate components
    - well-defined interfaces, described using WSDL
    - standardized invocation (SOAP)
  - Significant efforts in standardizing WS composition languages
  - Reuse of existing WS "infrastructure" (directory, service selection, ...)
    - WS composition tools are less expensive to develop

# Business Processes and Web Services

- Business Process Execution Language for Web Services (BPEL4WS)
  - XML-based language for specifying business process behavior based on web services
  - Describe business processes that both provide and consume web services
    - Steps (activities)
      - Implemented as an interaction with a web service
    - Information flow into/out of the process
      - Externalized as web service
- Complemented by
  - WS Coordination specification
    - Allows to web services involved in a process to share information that "links" them together
      - Shared coordination context
  - WS AtomicTransaction, WS BusinessActivity specifications
    - Allows to monitor the success/failure of each coordinated activity
      - Reliably cancel the business process, involves compensating activities
- Standardization through OASIS

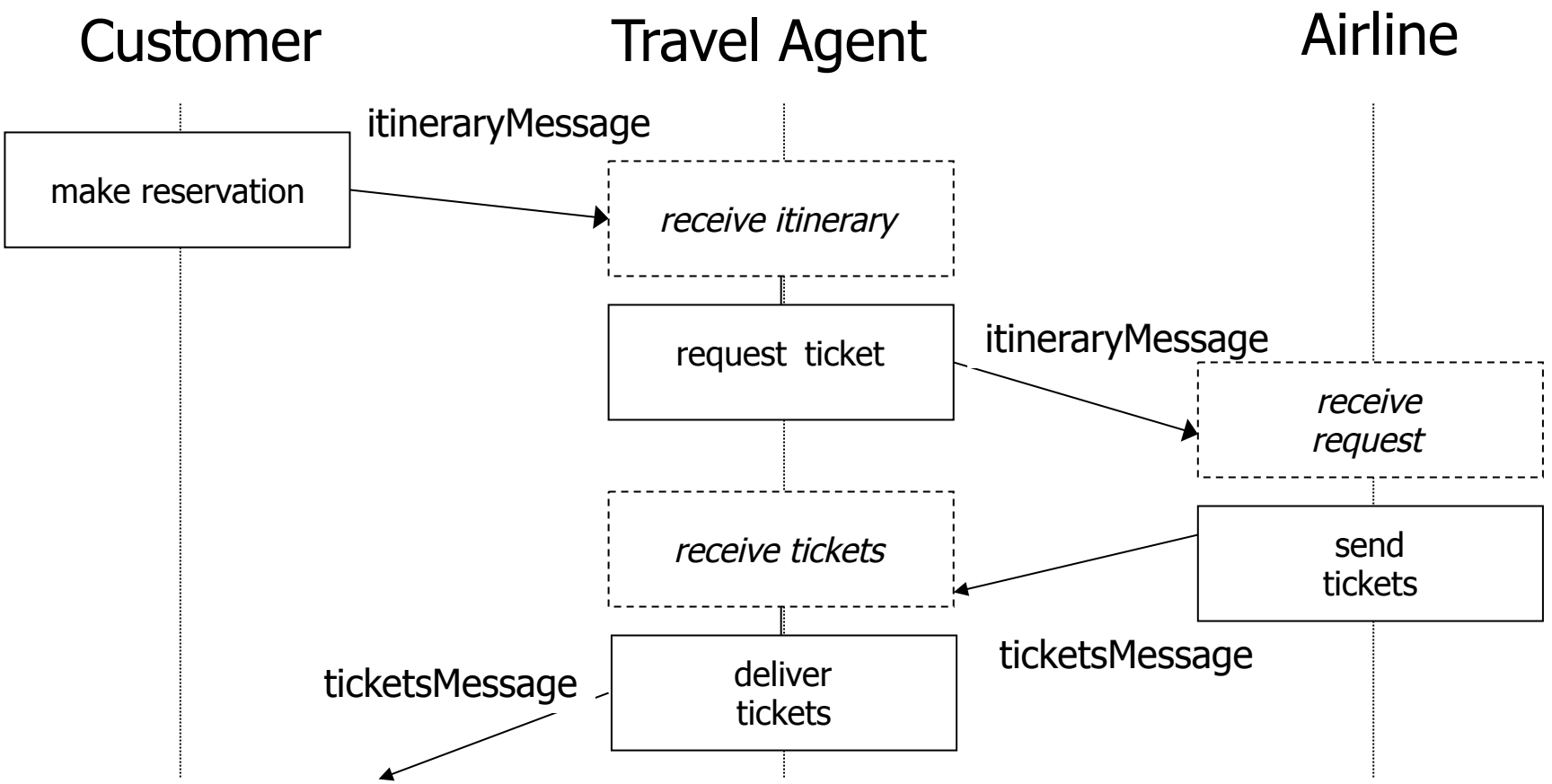
# BPEL4WS

- BPEL can support specification of both, composition schemas and coordination protocols
  - can be used in both composition and coordination middleware
- Two types of processes
  - executable process (-> composition)
    - defines implementation logic for a composite web service
    - portable between BPEL-conformant environments
  - abstract process (-> coordination)
    - service-centric perspective on coordination protocols
    - describe message exchange between partners
- Business process defines
  - potential execution order of operations (web services)
  - data shared between the web services
  - correlation information
  - partners involved in business process and interfaces they need to implement
  - joint exception handling for collection of web services

# BPEL Component Model

- Components are web services described using WSDL
  - abstract WSDL interfaces are referenced in BPEL scripts
  - no reference to bindings, endpoints, or services
- Basic activities in BPEL represent components, correspond to WSDL operations
  - Invoke
    - Issue an asynchronous request, or
    - Synchronously invoke a request/reply operation of a web service provided by a partner
  - Receive
    - Wait for a message to be received from a partner
    - Specifies partner from which message is to be received, as well as
    - The port and operation provided by the process
      - Used by the partner to pass the message
  - Reply
    - Synchronous response to a request corresponding to a receive activity
    - Combination of Receive/Reply corresponds to request-response operation in WSDL

# Example


Customer
Travel Agent
Airline
itineraryMessage
make reservation
receive itinerary
request ticket
itineraryMessage
receive request
receive tickets
send tickets
ticketsMessage
ticketsMessage
deliver tickets

# Service Selection: Partner Links

- Partner link (BPEL process definition)
  - identifies the web services mutually used by the partner or process
    - e.g., agent process interacts with customer, airline
  - references a partner link type
  - defines role taken by the process itself (myRole) and role that has to be accepted by the partner (partnerRole)
- Partner link names are used in all service interactions to identify partners
  - see activities for invoking/providing services

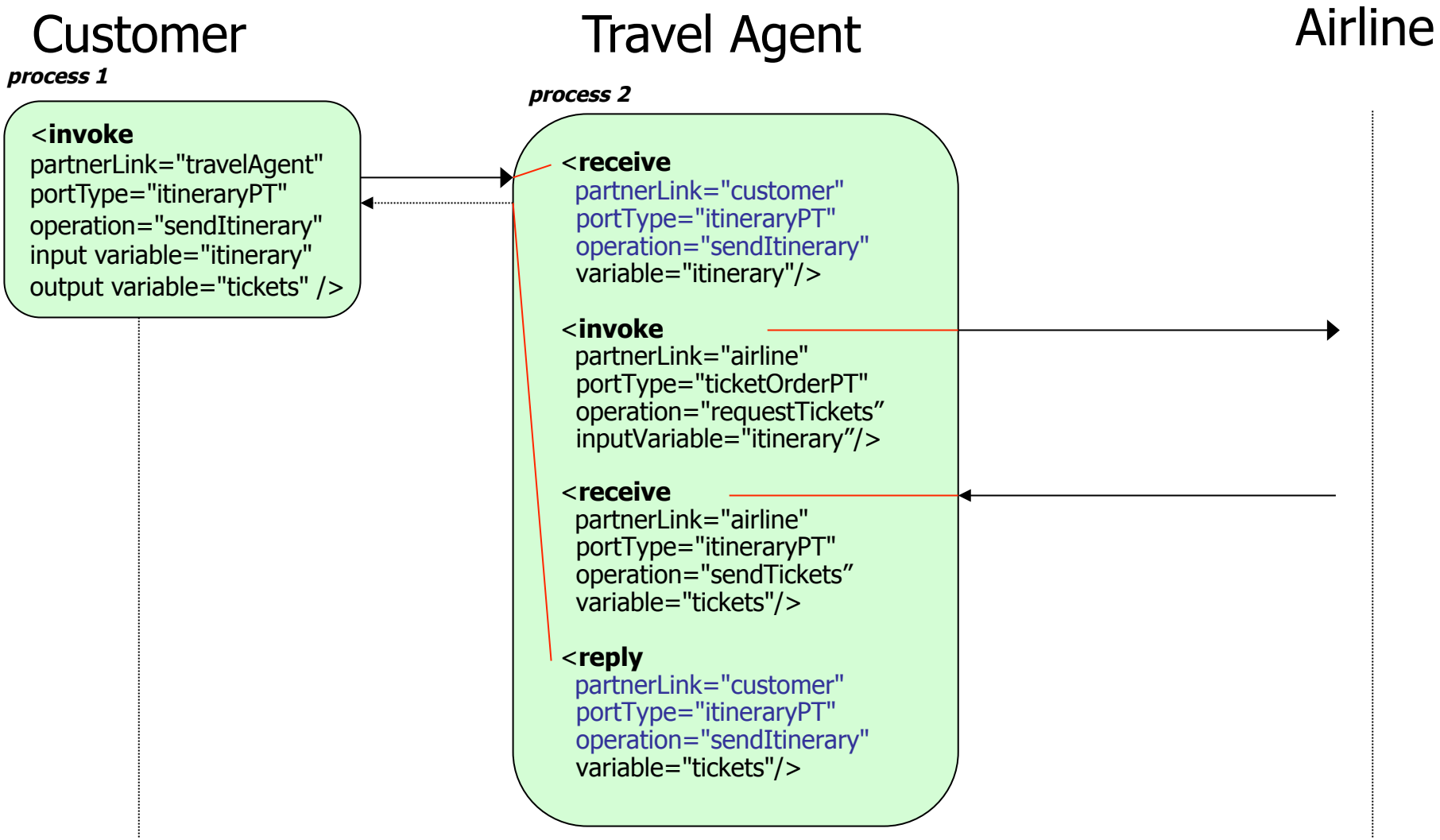
```
1   <process name="ticketOrder">
2   <partnerLinks>
3     <partnerLink name="customer"
4                  partnerLinkType="agentLink"
5                  myRole="agentService"/>
6     <partnerLink name="airline"
7                  partnerLinkType="buyerLink"
8                  myRole="ticketRequester"
9                  partnerRole="ticketService"/>
10  </partnerLinks>
```

- Partner link type (WSDL extension) defines
  - roles played by partners in a conversational relationship
  - web service interfaces that need to be implemented to assume a role
- Assignment of endpoints for partners
  - at deployment time
  - dynamically at run time

Partner link type definition

```
1 <partnerLinkType name="buyerLink">
2   <role name="ticketRequester">
3     <portType name="itineraryPT"/>
4   </role>
5   <role name="ticketService">
6     <portType name="ticketOrderPT"/>
7   </role>
8 </partnerLinkType>
```

# BPEL Activities – Example

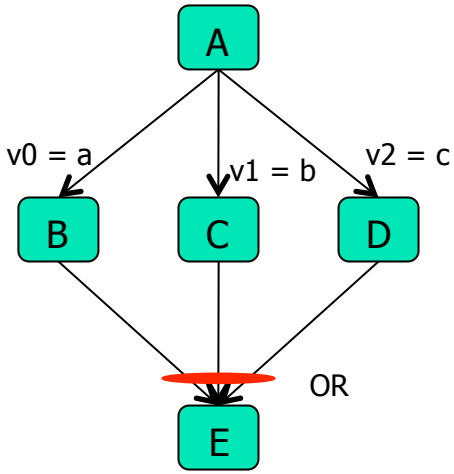


# Orchestration Model - Structured Activities

- Sequence
  - Enclosed activities are carried out in listed order
- If-else (i.e., switch)
  - Selects one of several activities based on selection criteria
- Repetitive Activities
  - While, RepeatUntil,
    - repeatedly carry out enclosed activities while/until specified condition is true
  - ForEach
    - serial: enclosed activity (scope) is carried out repeatedly, based on counter, optional completion condition
    - parallel: (effective copies of) enclosed activity (scope) executed n+1 times in parallel, based on start/end counter values
- Pick
  - Specifies a set of activities with associated events (e.g., receipt of message)
    - messages can be received from the same or different partners
    - activity is completed when one of the events occurs

# Structured Activities (cont.)

- Flow activity: defines sets of activities plus (optional) control flow
  - all activities can (potentially) execute in parallel
    - flow activity completes when all directly nested concurrent activities complete
    - implicit fork/join behavior
  - activities can be "wired together" via control links
    - link has one source activity, and one target activity
    - transition conditions
      - evaluated after source activity completes
      - determines the link status to be either true or false
      - links status also set to false, if source activity is determined not to be executed (e.g., if-else)
    - join conditions
      - can refer to status of incoming links of a target activity (e.g., AND, OR)
      - are evaluated only after the status of all incoming links is known
      - false join condition results in a join failure
    - dead path elimination
      - failure may be suppressed, status "false" is propagated to outgoing links



# Process life-cycle

- Start activities
  - receive, pick – createInstance attribute
    - creates a new process instance, if it doesn't exist already
  - Example:

```
<receive partner="customer",
        portType="itineraryPT",
        operation="sendItinerary",
        variable="itinerary"
        createInstance="yes"/>
```

  - each process must have at least one start activity as an initial activity
- Process termination
  - process-level activity completes successfully
  - fault "arrives" at the process level (handled or not)
  - terminate activity is invoked

# Data Types and Data Transfer

- Variables can be used to define data containers
  - WSDL messages received from or sent to partners
  - Messages that are persisted by the process
  - XML data defining the process state
- Constitute the "business context" of the process
- Access to variables can be serialized to some extent

```
11 <variables>
12      <variable name="itinerary" messageType="itineraryMessage"/>
13      <variable name="tickets" messageType="ticketsMessage"/>
14 </variables>
```

- Variable assignment
  - Receiving a message (or a reply of an invoke activity) implicitly assigns value
  - Alternative: assign activity (another simple activity)
    - Copies fields from containers into other containers

# Correlation

- Message needs to be delivered not only to the correct port, but to the correct instance of the business process providing the port
  - conversation routing
- Correlation Set
  - one or more properties used for correlating messages
  - example
    - <correlationSets>
      <correlationSet name="Booking"
      properties="orderNumber"/>
      ...
      </correlationSets>
  - correlation properties are like "late-bound constants"
    - binding happens through specially marked message send/receive activities
    - value must not change after the binding happens
- Often, more than one correlation set is used for an entire process
  - example: orderNumber -> invoiceNumber
  - correlated message exchanges may nest, overlap
  - same message may carry multiple correlation sets

# Properties

- Property
  - Globally defined types
  - Primarily used to correlate a message with a specific process instance
    - E.g., order number
    - Usually included in the message
    - Often the same property is used in different messages
  - Can be defined in BPEL as a separate entity:

```
9  <property name="orderNumber" type="xsd:int"/>
```

- Property alias
  - Allows to point to a dedicated field of the message that represents the property
    - Usually different for each message type
    - Can be used in expression and assignments to easily use properties

```
10 <propertyAlias propertyName="orderNumber"
11                messageType="ticketsMessage"
12                part="orderInfo"
13                query="/orderID"/>
```

# Scope

- Defines the behavior context of an activity (primary activity)
  - simple or structured (group of activities)
- Can provide the following for a (regular) activity
  - (Local) data variables
  - Correlation Sets
  - Event handler(s)
  - Fault handler(s)
  - Termination handler
  - Compensation handler
    - Scope acts as a compensation sphere
- Scopes can be arbitrarily nested

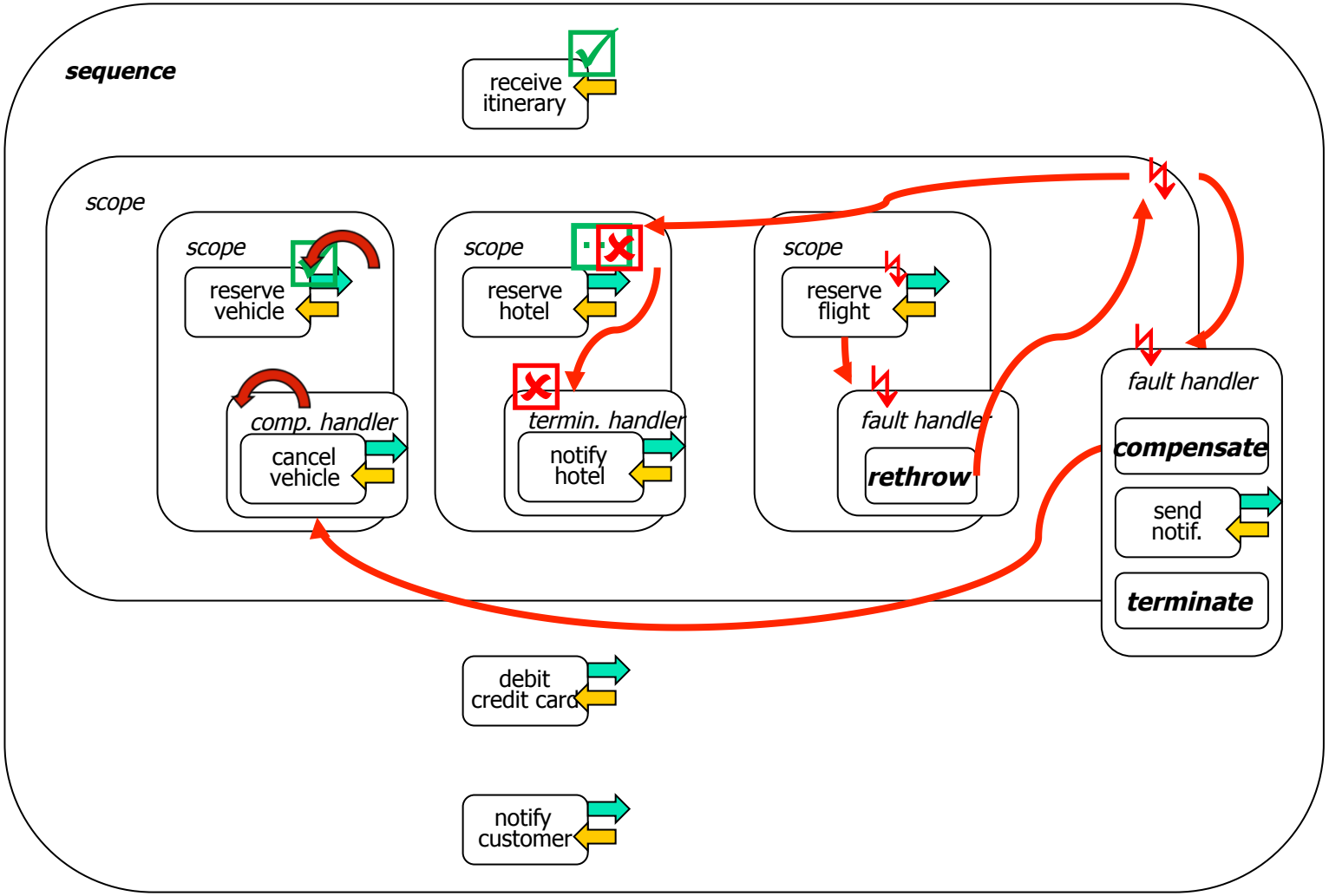
# Fault Handlers and Termination Handler

- Fault handlers catch and deal with faults occurring in **active** scope
  - Can catch internal faults (throw activity), WS fault messages
  - All active work in the scope is stopped!
    - Results in invocation of termination handlers for active enclosed scopes
  - After fault handler completes successfully, processing continues outside the scope
    - Processing of the scope is still considered to have ended abnormally
- Termination handler allows to define scope-specific termination behavior
  - Invoked if an active scope needs to be terminated
    - Example: perform cleanup work, notify business partner, cancel activity
  - For nested scope: TH for inner scope is invoked before the TH of the outer

# Compensation Handlers

- Compensation handlers reverse the work of a **sucessfully completed** scope
  - Compensation handler is "installed" after successful completion of the scope
  - Can be defined for each scope
  - Compensation activity can be any activity
  - Compensation handlers live in a snapshot world
    - When invoked, they see a snapshot of the variables at scope completion time
    - Cannot update "live" data variables
    - Can only affect external entities
    - Input/output parameters for compensation handler are future direction
- Compensate activity
  - Invokes compensation handler for named scope
  - Can be invoked only from the fault handler or compensation handler of the immediately enclosing scope

# Fault-Termination-Compensation - Example



# Default Compensation and Fault Handlers

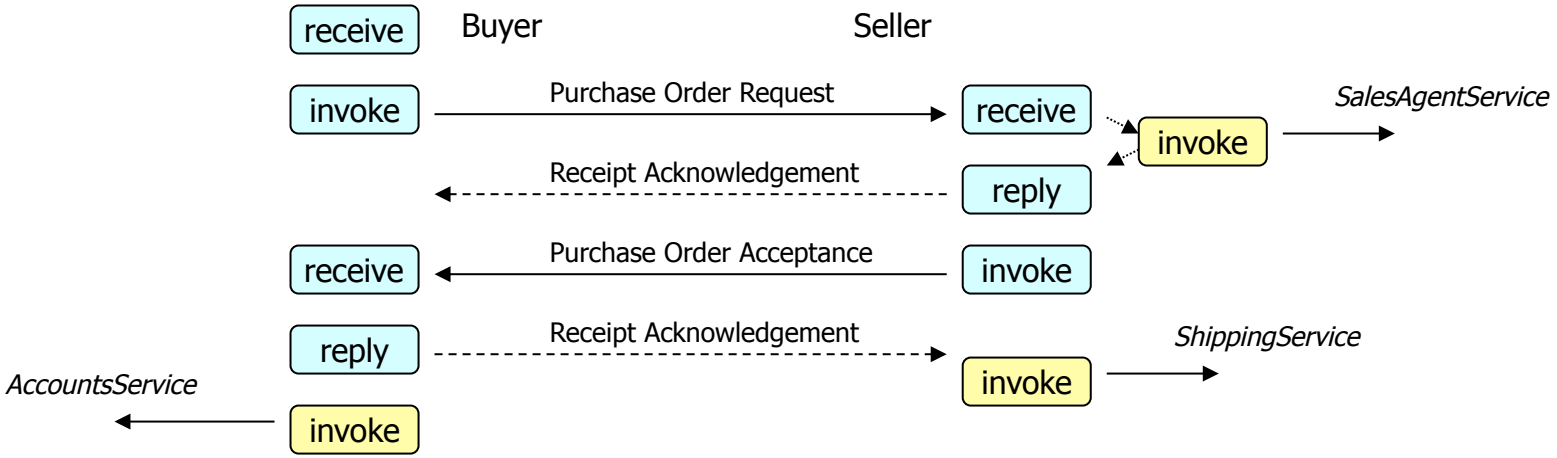
- Default compensation handler
  - Invokes compensation handlers of immediately enclosed scopes in the reverse order of the completion of the scopes
  - Is used if a (enclosing) scope does not explicitly define a compensation handler
  - Can also be invoked explicitly
    - Useful if comp. action = "compensate enclosed scope in reverse order" + "additional activities"
- Default fault handler
  - Invokes compensation handlers of immediately enclosed scopes in the reverse order of the completion of the scopes
  - Rethrows the exception

# BPEL – Abstract Processes

- Abstract Process = Role-specific view of a protocol
  - only public information
  - no private, implementation-specific aspects
    - branching conditions, activity realization, …
  - not executable
  - can be used by a conversation controller to ensure business protocol compliance
- Properties of BPEL abstract processes
  - handle only protocol-relevant data
    - message properties
  - variables
    - do not need to be fully initialized
    - variables for inbound or outbound messages may be omitted from invoke, receive, reply, if the intent is to just constrain the sequence of activities
  - opaque assignments
    - can correspond to creating a unique value for correlation properties
    - hide private behavior for providing the values

# Implementing Business Protocols

- Suggested path
  - protocol specification as a starting point
  - derive role-specific views of the protocol
    - includes all the message exchanges that involve a certain role
  - define abstract process for role-specific view
    - model interactions using receive, invoke, reply
    - represent additional public information, such as branching situations, parallelism
  - turn abstract process into an executable process to implement it



# RosettaNet

- Goal: Develop standard e-commerce interfaces to align the processes between IT supply chain partners
  - consortium founded in 1998
  - "vertical" coordination protocols
  - more than 3000 documented production implementations by 2004
- Main standardization areas
  - (Public) Business processes
    - coordination protocols for trading partners
    - Partner Interface Processes (PIPs)
      - business documents, vocabulary, choreography of message exchanges
  - Data format
    - establishment of a common vocabulary
      - business directory
      - technical dictionary
  - Message services
    - RosettaNet Implementation Framework
      - reliable, secure execution of the protocol specifications
      - transfer, routing, packaging of encrypted and authenticated messages between business partners

# PIP Definitions

- Standardized PIP definitions are arranged into clusters, further broken down into segments
- Clusters:
  1. RosettaNet Support
     - administrative functionality
  2. Partner Product and Service Review
     - collect, maintain, distribute product or service information
       - account setup, product info subscription, …
  3. Product Information
     - distribute, update product information
       - query technical product info, …
  4. Order Management
       - request quote, request purchase order, query order status, …
  5. Inventory Management
       - distribute inventory report, …
  6. Marketing Information Management
     - exchange of marketing information
  7. Service and Support
       - request warranty claim, …
  8. Manufacturing
     - "virtual manufacturing"
       - notify of manufacturing work order, …

# Implementing RosettaNet PIPs

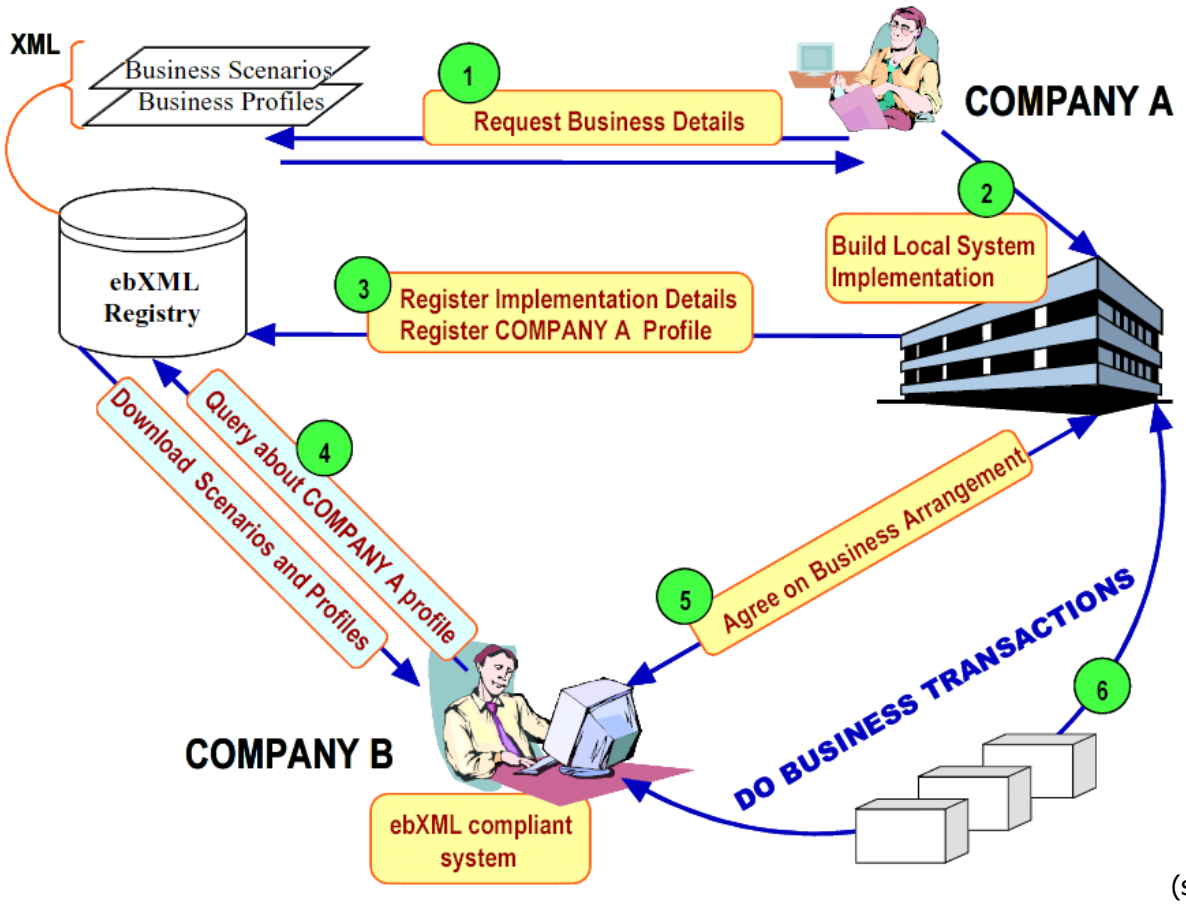
- Involves mapping PIP to WSDL, BPEL
  - types in message definitions -> types in WSDL
    - DTDs to XML Schema
  - message definitions -> WSDL message definitions
  - PIP actions -> operations in WSDL
  - PIP partner roles -> BPEL partners
  - PIP choreography: follow the "suggested path" on previous chart
- Additional aspects
  - realize time-outs, etc. using BPEL events and fault handlers
  - additional requirements regarding security need to be resolved
    - WS-Security support, not integrated in BPEL

# ebXML

- Supported by UN/CEFACT, OASIS
- Vision
  - single global electronic marketplace
  - based on exchange of XML messages
- ebXML architecture covers:
  - definition of business processes and their associated messages and content
  - registry and discovery of business process sequences with related message exchanges
  - definition of company profiles
  - definition of trading partner agreements
  - uniform message transport layer
- ebXML advantages
  - goes beyond generic protocols and specifications
    - e.g., ebXML registry is much more detailed than UDDI
  - captures the logic behind e-commerce exchanges
    - e.g., business arrangements
  - specifies how e-commerce exchanges should be specified, documented, conducted

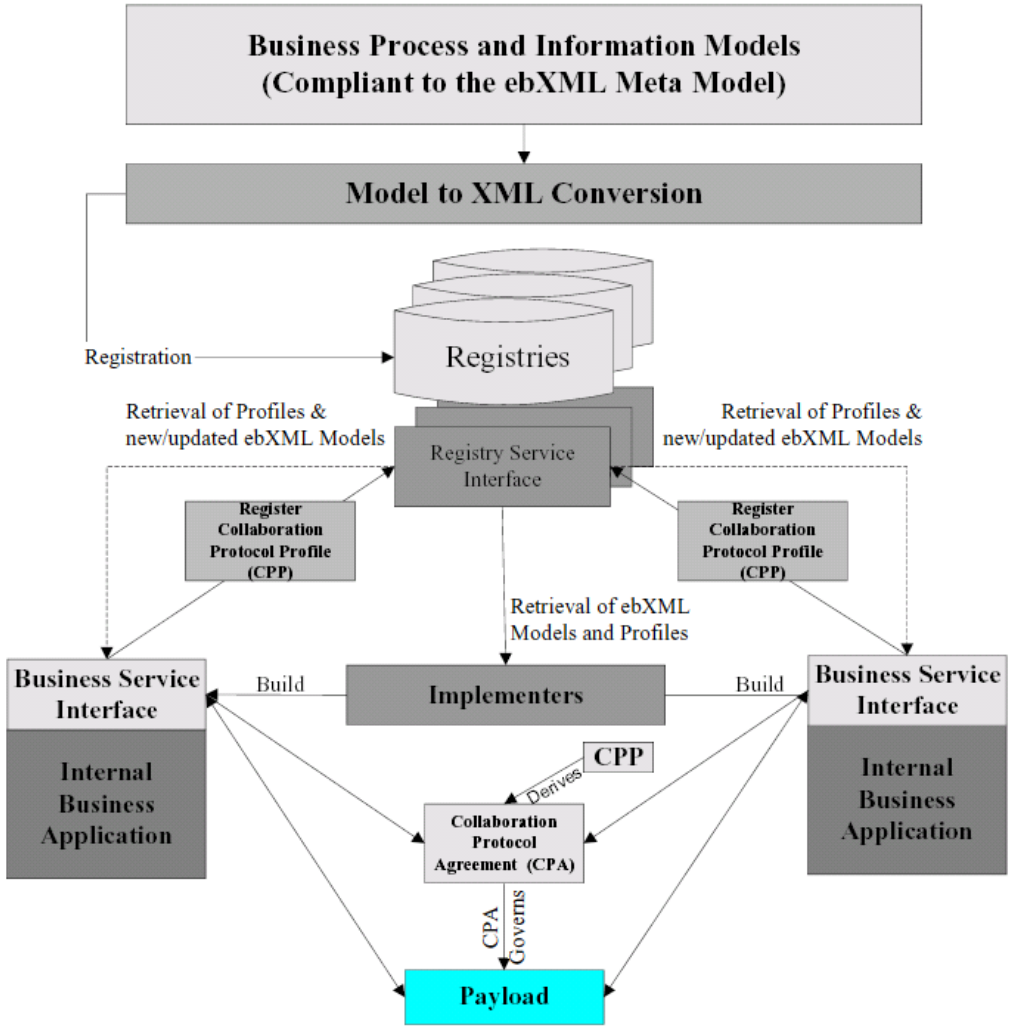
# Collaboration with ebXML

- Example



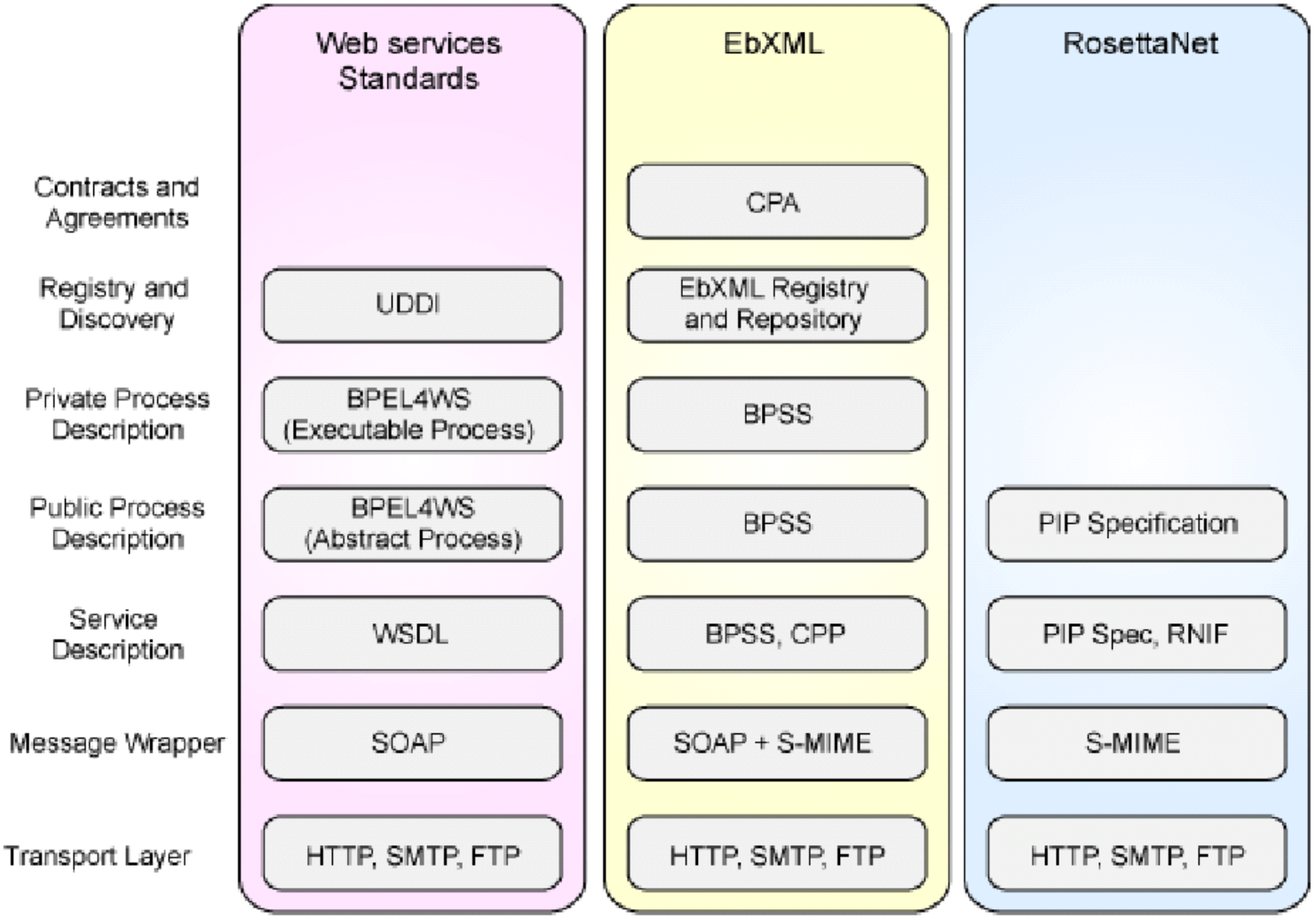

(source: ebXML Technical Architecture Specification)

# Technical Architecture




(source: ebXML Technical Architecture Specification)

# How Do These Standards Relate?

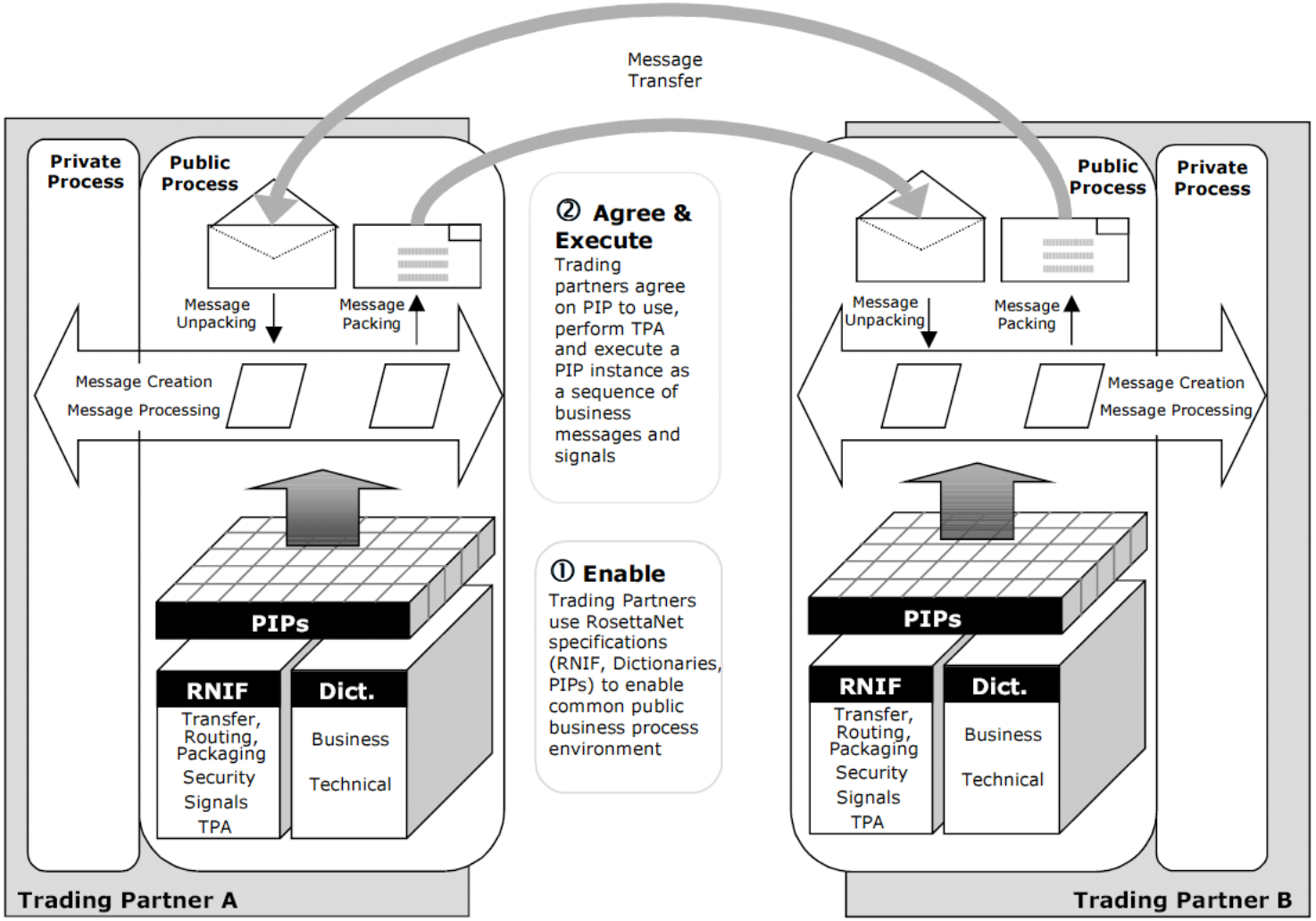


# Summary

- Web service composition
  - means to implement web service by reusing/combining existing services
  - can be supported by WS composition middleware
    - borrowing concepts from WFMS
- BPEL
  - de-facto and de-jure (OASIS) web service composition standard
  - allows definition of composition and coordination aspects
    - abstract vs. executable processes
  - main concepts
    - basic activities for web service operations
    - structured activities for defining service composition, control flow
    - blackboard approach for data flow based on variables
    - service selection based on partner link types, partner links, endpoints
    - elaborate model for transactions and exception handling
      - fault handler, termination handler, compensation handler
- More BPEL extensions are on the way
  - people WF (BPEL4People, WS-HumanTask), Java/SQL snippets (BPELJ, BPEL/SQL)

# APPENDIX

# RosettaNet Trading Partner Implementation



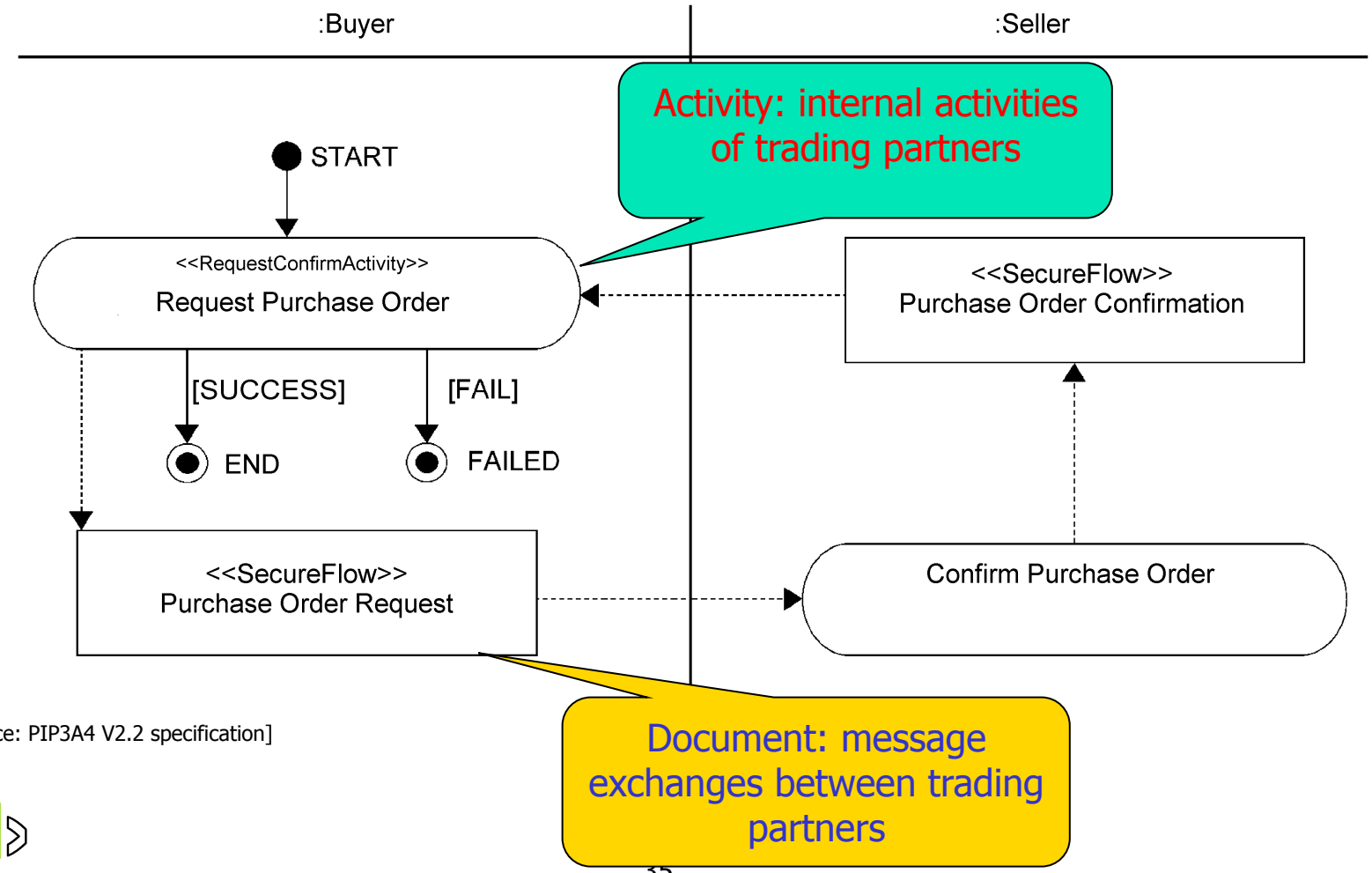

[source: RosettaNet Implementation Framework Core Specification]

# Partner Interface Process (PIP) Specifications

- Describes how to implement a collaborative coordination protocol
  - technical dictionary describes components that are exchanged
  - message guideline document
    - business actions, business signals (ack receipt of action message)
- Major PIP specification sections
  - Business Operational View (aka Action Layer)
    - flow of business interactions, based on
      - partner roles
      - partner role interactions
  - Functional Service View (aka Transaction Layer)
    - derived from the business operational view
    - business transactions between entities in the form of message exchanges
      - ccordination protocols
      - message control information
        - time limits for acknowledgements
        - security requirements
  - Implementation Framework View (aka Service Layer)
    - based on functional service and business operational views
    - defines communication protocol and message format requirements
      - e.g., SSL, encryption, XML DTDs for messages, …

# Business Operational View - Example

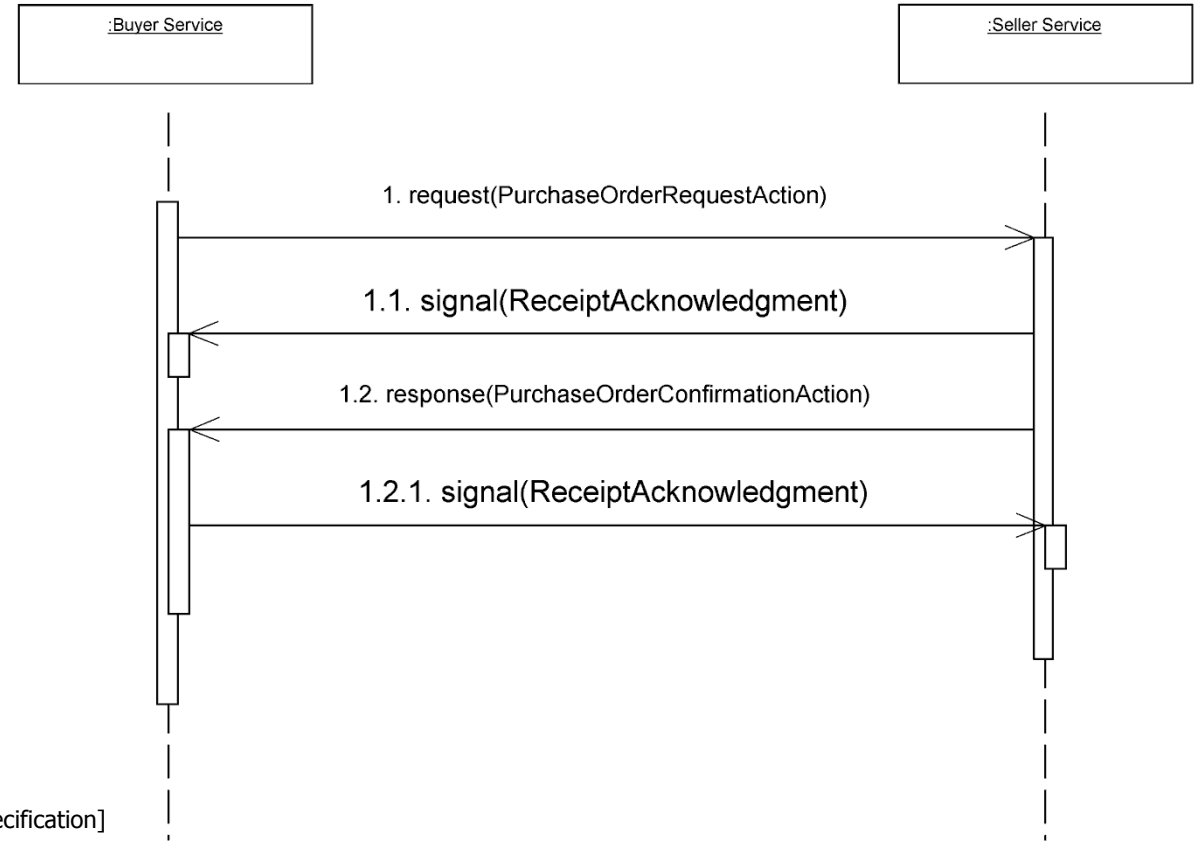
- Business Process Diagram for PIP3A4: Request Purchase Order




[source: PIP3A4 V2.2 specification]

# Functional Service View – Example

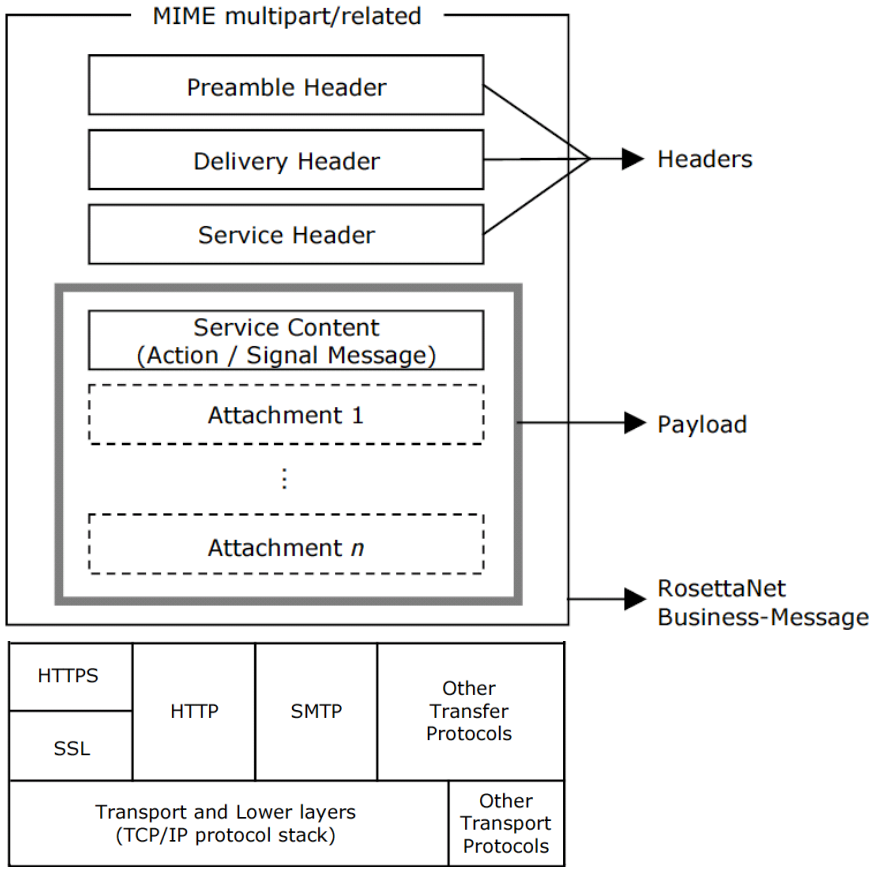
- Business Transaction Dialog Specification for PIP3A4: Request Purchase Order




[source: PIP3A4 V2.2 specification]

# RosettaNet Implementation Framework

- Defines
  - Business Message
    - packaging payload (incl. attachments), headers, …
    - uses MIME, S/MIME
  - Protocol Stack
    - transport-independent
    - reliable messaging
      - support for HTTP, SMTP, …
  - Security Mechanism
    - based on encryption, digital signatures
    - supports authentication, authorization, encryption, non-repudiation
- Designed before the time of SOAP
  - May likely be replaced by SOAP-based web service infrastructure in the future




[source: RosettaNet Implementation Framework Core Specification]