A Very Short Introduction to Web Services

Massimo Mecella

Dipartimento di Informatica e Sistemistica ANTONIO RUBERTI SAPIENZA Università di Roma
mecella@dis.uniroma1.it
BASIC CONCEPTS
An e-Service is often defined as an application accessible via the Web, that provides a set of functionalities to businesses or individuals. What makes the e-Service vision attractive is the ability to automatically discover the e-Services that fulfill the users’ needs, negotiate service contracts, and have the services delivered where and when users need them.

*Guest editorial. In [VLDBJ01]*

**e-Service**: an application component provided by an organization in order to be assembled and reused in a distributed, Internet-based environment; an application component is considered as an e-Service if it is: (i) **open**, that is independent, as much as possible, of specific platforms and computing paradigms; (ii) **developed mainly for inter-organizations applications**, not only for intra-organization applications; (iii) **easily composable**: its assembling and integration in an inter-organizations application does not require the development of complex adapters.

**e-Application**: a distributed application which integrates in a cooperative way the e-Services offered by different organizations.

*M. Mecella, B. Pernici: Designing Wrapper Components for e-Services in Integrating Heterogeneous Systems. In [VLDBJ01]*
A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.

• Services are self-describing, open components that support rapid, low-cost composition of distributed applications. Services are offered by service providers — organizations that procure the service implementations, supply their service descriptions, and provide related technical and business support.

Since services may be offered by different enterprises and communicate over the Internet, they provide a distributed computing infrastructure for both intra and cross-enterprise application integration and collaboration.

Service descriptions are used to advertise the service capabilities, interface, behavior, and quality. Publication of such information about available services provides the necessary means for discovery, selection, binding, and composition of services. In particular, the service capability description states the conceptual purpose and expected results of the service (by using terms or concepts defined in an application-specific taxonomy). The service interface description publishes the service signature (its input/output/error parameters and message types). The (expected) behavior of a service during its execution is described by its service behavior description. Finally, the Quality of Service (QoS) description publishes important functional and nonfunctional service quality attributes [...]. Service clients (end-user organizations that use some service) and service aggregators (organizations that consolidate multiple services into a new, single service offering) utilize service descriptions to achieve their objectives.

• The application on the Web (including several aspects of the SOA) is manifested by Web services

Guest editorial. In [CACMO3]
And Today?

- **e-Service**
  - e-Service is the provision of a service via the Internet (the prefix “e” standing for “electronic”)
  - True Web jargon, meaning just about anything done online
  - Basically whichever Web application usable by a human, through a user interface

- **Web service**
  - software component available on the Web, to be invoked by some other client application/component
  - A way of building Web-scale component-based distributed systems

- For building an e-Service, a designer may need to use/invoke many Web services
Two Architectures (and Middlewares) (1)

Company A (provider)
- Web service interface
  - Logic for accessing to internal systems
    - internal architecture & middleware
      - internal service logic

Company B (provider)

Company C (provider)
- Web service
- Web service
- Web service

Company D (client)

[from ACKM04]
Two Architectures (and Middlewares) (2)

1. publish the service description
2. find
3. interact

the abstraction and infrastructure provided by the registry are part of the external middleware

Company A (service requester)

Web service client

Web services middleware (internal)

other tiers

Company B (service provider)

Web service

Web services middleware (internal)

other tiers

Company C (directory service provider)

service descriptions
Two Architectures (and Middlewares) (3)

Company A (service requester)

- Web service client
- Internal middleware
- Other tiers

Company B (service provider)

- Web service
- Internal middleware
- Other tiers

Company C (directory service provider)

- Service descriptions

[from ACKM04]
Services

- A service is characterized by the set of (atomic) operations that it exports ...

- ... and possibly by constraints on the possible conversations
  - Using a service typically involves performing sequences of operations in a particular order (conversations)
  - During a conversation, the client typically chooses the next operation to invoke (on the basis of previous results, etc.) among the ones that the service allows at that point

\[
\begin{aligned}
\text{Client} & \xrightarrow{\text{requestQuote}} \text{Service} \\
\text{Service} & \xrightarrow{\text{orderGoods}} \text{Client} \\
\text{Service} & \xrightarrow{\text{confirmOrder}} \text{Client} \\
\text{Service} & \xrightarrow{\text{makePayment}} \text{Client} \\
\end{aligned}
\]
**Choreography: Coordination of Conversations of N Services**

- **Global specification of the conversations of N peer services (i.e., multi-party conversations)**
  - Roles
  - Message exchanges
  - Constraints on the order in which such exchanges should occur

Choreography: Coordination of Conversations of N Services

[from ACKM04]
Composition

- Deals with the implementation of an application (in turn offered as a service) whose application logic involves the invocation of operations offered by other services
  - The new service is the *composite service*
  - The invoked services are the *component services*
Through the development environment, a composition schema is synthesized, either manually or (semi-)automatically. A service composition model and a language (maybe characterized by a graphical and a textual representation) are adopted.

**Orchestration**: the run-time environment executes the composite service business logic by invoking other services (through appropriate protocols).

**Development environment**
- development environment
- run-time environment (orchestration engine)
- schema definitions
- composite service execution data

**Other Web Services middleware**
- Supplier WS
- Warehouse WS
- Accounting WS

**Component services offered by other providers**
- House hunting service
- Packaging service
- Flight reservation service
- Shipment service
- Internet DSL line installation service
- Phone line installation service

**Web service composition middleware**
- development environment
- run-time environment (orchestration engine)
- schema definitions
- composite service execution data

**Composition schema**
- Warehouse WS
- Supplier WS
- Accounting WS

**Component service provider** [from ACKM04]
Synthesis and Orchestration

- (Composition) Synthesis: building the specification of the composite service (i.e., the composition schema)
  - Manual
  - Automatic
- Orchestration: the run-time management of the composite service (invoking other services, scheduling the different steps, etc.)
  - Composition schema is the "program" to be executed
  - Similarities with WfMSs (Workflow Management Systems)
Composition Schema

- A composition schema specifies the “process” of the composite service
  - The “workflow” of the service
- Different clients, by interacting with the composite service, satisfy their specific needs (reach their goals)
  - A specific execution of the composition schema for a given client is an orchestration instance
**Choreography (Coordination) vs. Composition (Orchestration)**

- **Composition is about implementing new services**
  - From the point of view of the client, a composite service and a basic (i.e., implemented in a traditional programming language) one are indistinguishable.

- **Choreography is about global modeling of N peers, for proving correctness, design-time discovery of possible partners and run-time bindings**

- **N.B.: There is a strong relationship between a service internal composition and the external choreographies it can participate in**
  - If A is a composite service that invokes B, the A's composition schema must reflect the coordination protocol governing A - B interactions.
  - In turn, the composition schema of A determines the coordination protocols that A is able to support (i.e., the choreographies it can participate in).
Services Mash-up (1)

Web application that combines data from one or more sources into a single integrated tool

- easy, fast integration, frequently done by access to open APIs and data sources to produce results that were not the original reason for producing the raw source data.
- E.g., cartographic data from Google Maps to add location information to real estate data, thereby creating a new and distinct e-Service that was not originally provided by either source.
- Bottom-up, developers-driven approach

Services Mash-up (2)

- Based on various technologies
  - Web services
    - SOAP
    - RESTful
    - Atom/RSS
- Basically a lightweight form of composition
RELEVANT TECHNOLOGIES AND ABSTRACTIONS
The "Stacks" of Service Technologies

- Registry/Repository & Discovery
  - WSDL-based

- Multiple Interacting Services
  - RESTful
  - ebXML-based
  - Semantic-based

- Single Service
  - WSDL-based

- Messaging
The WSDL-based "Stack"

Includes 3 specifications:
(i) Web Service Context (WS-CTX)
(ii) Web Service Coordination Framework (WS-CF)
(iii) Web Service Transaction Management (WS-TXM)

repository & discovery

transaction management

choreography

orchestration

conversation description / interaction protocol

non-functional features, QoS

description (interface definition)

advanced messaging

basic messaging

content

transport

Formerly WSEL (Web Service Endpoint Language) by IBM (i)
Also the research/academic proposal
Web Service Offering Language [WSOL]

WSCL (Web Service Conversation Language) (by HP) and
Conversation Support for Web Services (CS-WS, by IBM) proposals are no
more supported

(i) formerly also WSEL (Web Service Endpoint Language) by IBM
(ii) also the research academic proposal Web Service Offering Language [WSOL]
A Minimalist Infrastructure for Web Service

- service requestor
  - application object (client)
  - SOAP-based middleware

- service provider
  - application object (service provider)
  - SOAP-based middleware

SOAP messages exchanged on top of HTTP, SMTP, or other transport.

Converts procedure calls to/from XML messages sent through HTTP or other protocols.
<operation name="orderGoods">
  <input message = "OrderMsg"/>
</operation>

From Interfaces to Stub/Skeleton
**SOAP messages**

- SOAP is based on **message exchanges**.
- Messages are seen as **envelopes** where the application encloses the data to be sent.
- A SOAP message consists of a SOAP of an `<Envelope>` element containing an optional `<Header>` and a mandatory `<Body>` element.
- The contents of these elements are application defined and not a part of the SOAP specifications.
- A SOAP `<Header>` contains blocks of information relevant to how the message is to be processed. This helps pass information in SOAP messages that is not application payload.
- The SOAP `<Body>` is where the main end-to-end information conveyed in a SOAP message must be carried.
SOAP envelope and header

Example of SOAP envelope

Example of SOAP header
The SOAP Body

- The SOAP body is the area of the SOAP message, where the application specific XML data (payload) being exchanged in the message is placed.

- The `<Body>` element must be present and is an immediate child of the envelope. It may contain a number of child elements, called body entries, but it may also be empty. The `<Body>` element contains either of the following:
  - **Application-specific data** is the information that is exchanged with a Web service. The SOAP `<Body>` is where the method call information and its related arguments are encoded. It is where the response to a method call is placed, and where error information can be stored.
  - A **fault message** is used only when an error occurs.

- A SOAP message may carry either application-specific data or a fault, but not both.
<?xml version='1.0' ?>

<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope" >
  <env:Header>
    <t:transactionID
      xmlns:t="http://intermediary.example.com/procurement"
      env:role="http://www.w3.org/2002/06/soap-envelope/role/next"
      env:mustUnderstand="true" >
      57539
    </t:transactionID>
  </env:Header>
  <env:Body>
    <m:orderGoods
      env:encodingStyle="http://www.w3.org/2002/06/soap-encoding"
      xmlns:m="http://example.com/procurement">
      <m:productItem>
        <name>ACME Softener</name>
      </m:productItem>
      <m:quantity>
        35
      </m:quantity>
    </m:orderGoods>
  </env:Body>
</env:Envelope>
The SOAP Communication Model

- SOAP supports two possible communication styles:
  - remote procedure call (RPC) and
  - document (or message).
RPC-style SOAP Services

- A remote procedure call (RPC)-style Web service appears as a remote object to a client application. The interaction between a client and an RPC-style Web service centers around a service-specific interface. Clients express their request as a method call with a set of arguments, which returns a response containing a return value.

SOAP envelope

<table>
<thead>
<tr>
<th>SOAP body</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method name</td>
</tr>
<tr>
<td>orderGoods</td>
</tr>
</tbody>
</table>

Input parameter 1

| input parameter 1 |
| product item |

Input parameter 2

| input parameter 2 |
| quantity |

Method return

| Return value |
| order id |
RPC-style web services

Example of RPC-style SOAP body

Example of RPC-style SOAP response message
Document (Message)-style SOAP Services

- In the document-style of messaging, the SOAP <Body> contains an XML document fragment. The <Body> element reflects no explicit XML structure.
- The SOAP run-time environment accepts the SOAP <Body> element as it stands and hands it over to the application it is destined for unchanged. There may or may not be a response associated with this message.
Example of document-style SOAP body

```xml
<env:Envelope
xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <tx:Transaction-id
      xmlns:t="http://www.transaction.com/transactions"
      env:mustUnderstand='1'>512</env:Transaction-id>
  </env:Header>
  <env:Body>
    <po:PurchaseOrder
      oderDate="2004-12-02"
      xmlns:m="http://www.plastics_supply.com/POs">
      <po:from>
        <po:accountName>RightPlastics</po:accountName>
        <po:accountNumber>PSC-0343-02</po:accountNumber>
      </po:from>
      <po:to>
        <po:supplierName>Plastic Supplies Inc.</po:supplierName>
        <po:supplierAddress>Yara Valley Melbourne</po:supplierAddress>
      </po:to>
      <po:product>
        <po:product-name>injection molder</po:product-name>
        <po:product-model>G-100T</po:product-model>
        <po:quantity>2</po:quantity>
      </po:product>
    </po:PurchaseOrder>
  </env:Body>
</env:Envelope>
```
SOAP Fault element

- SOAP provides a model for handling faults arise.
- It distinguishes between the conditions that result in a fault, and the ability to signal that fault to the originator of the faulty message or another node. The SOAP <Body> is the place where fault information is placed.

```xml
<env:Envelope
    xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://www.plastics_supply.com/product-prices">
  <env:Header>
    <tx:Transaction
        xmlns:t="http://www.transaction.com/transactions"
        env:mustUnderstand='1'>
      512
    </tx:Transaction>
  </env:Header>
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode>
          <env:Value> m:InvalidPurchaseOrder </env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-UK"> Specified product did not exist </env:Text>
      </env:Reason>
      <env:Detail>
        <err:myFaultDetails
            xmlns:err="http://www.plastics_supply.com/faults">
          <err:message> Product number contains invalid characters </err:message>
          <err:errorcode> 129 </err:errorcode>
        </err:myFaultDetails>
      </env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```
Registry

Service requestor

- Application object (client)
- Stub
- SOAP-based middleware

Service provider

- Application object (service provider)
- Skeleton
- SOAP-based middleware

SOAP messages

- SOAP messages (to look for services)
- SOAP messages (to publish service description)

SOAP-based middleware

Service descriptions

UDDI registry
UDDI and WSDL

- Service provider
  - WSDL service description
- Service requestor
  - Inquiry URL
  - SOAP-HTTP
  - Publishing URL
  - SOAP-HTTPS

UDDI service registry
- Service description
  - Inquiry URL
  - SOAP-HTTP
  - Publishing URL
  - SOAP-HTTPS

- UDDI entry
  - White pages info.
  - Yellow pages info.
  - Green pages info.
  - Technical info.
  - Pointer to service description

- Technical information:
  - UDDI entry
  - Service provider
  - Service requestor
  - WSDL service description
Web Service Definition Language (WS-DL)

- WS-DL provides a framework for defining:
  - Interface: operations and input/output formal parameters
  - Access specification: protocol bindings (e.g., SOAP)
  - Endpoint: the location of service
Message Exchange Patterns
(1)

**in-only (no faults)**

Client

Service

**robust in-only (message triggers fault)**

Client

Service

**out-only (no faults)**

Client

Service

**robust out-only (message triggers fault)**

Client

Service
Message Exchange Patterns (2)

- **in-out (fault replaces message)**
  - (1) input
  - (2) output
  - (2') fault

- **out-in (fault replaces message)**
  - (1) output
  - (2) input

- **in-optional-out (message triggers fault)**
  - (?) output
  - (?') input

- **out-optional-in (message triggers fault)**
  - output
  - (?) input
An Example (1)

<definitions ... >
  <types>
    <element name="ListOfSong_Type">
      <complexType>
        <sequence>
          <element minOccurs="0" maxOccurs="unbound" name="SongTitle" type="xs:string"/>
        </sequence>
      </complexType>
    </element>
    <element name="SearchByTitleRequest">
      <complexType>
        <all>
          <element name="containedInTitle" type="xs:string"/>
        </all>
      </complexType>
    </element>
    <element name="SearchByTitleResponse">
      <complexType>
        <all>
          <element name="matchingSongs" xsi:type="ListOfSong_Type"/>
        </all>
      </complexType>
    </element>
  </types>
</definitions>
An Example (2)

```xml
<element name="SearchByAuthorRequest">
  <complexType><all>
    <element name="authorName" type="xs:string"/>
  </all></complexType>
</element>
<element name="SearchByAuthorResponse">
  <complexType><all>
    <element name="matchingSongs" xsi:type="ListOfSong_Type"/>
  </all></complexType>
</element>
<element name="ListenRequest">
  <complexType><all>
    <element name="selectedSong" type="xs:string"/>
  </all></complexType>
</element>
```
An Example (3)

```xml
<element name="ListenResponse">
    <complexType><all>
        <element name="MP3fileURL" type="xs:string"/>
    </all></complexType>
</element>

<element name="ErrorMessage">
    <complexType><all>
        <element name="cause" type="xs:string"/>
    </all></complexType>
</element>

</types>
```
<interface name="MP3ServiceType">
  <operation name="search_by_title" pattern="in-out">
    <input message="SearchByTitleRequest"/>
    <output message="SearchByTitleResponse"/>
    <outfault message="ErrorMessage"/>
  </operation>
  <operation name="search_by_author" pattern="in-out">
    <input message="SearchByAuthorRequest"/>
    <output message="SearchByAuthorResponse"/>
    <outfault message="ErrorMessage"/>
  </operation>
  <operation name="listen" pattern="in-out">
    <input message="ListenRequest"/>
    <output message="ListenResponse"/>
    <outfault message="ErrorMessage"/>
  </operation>
</interface>
</definitions>

Definition of an operation and its message exchange pattern

Definition of a service interface
Transition Systems

- A transition system (TS) is a tuple \( T = \langle A, S, S^0, \delta, F \rangle \) where:
  - \( A \) is the set of actions
  - \( S \) is the set of states
  - \( S^0 \subseteq S \) is the set of initial states
  - \( \delta \subseteq S \times A \times S \) is the transition relation
  - \( F \subseteq S \) is the set of final states
Automata vs. Transition Systems

- Automata
  - define sets of runs (or traces or strings): (finite) length sequences of actions

- TSs
  - ... but I can be interested also in the alternatives “encountered” during runs, as they represent client’s “choice points”

As automata they recognize the same language: abc* + ade*

Different as TSs
WS-DL is the Set of Actions

• A message exchange pattern (and the related operation) represents an interaction with the service client
  - an action that the service can perform by interacting with its client
• Abstracting from formal parameters, we can associate a different symbol to each operation ...
• ... thus obtaining the alphabet of actions
An Example

- The MP3ServiceInterface defines 3 actions:
  - `search_by_title / st`
  - `search_by_author / sa`
  - `listen / l`

- Formally $A = \{st, sa, l\}$
Putting All Together

service provider

- service implementation
- server stub
- SOAP router
- HTTP engine

1. WSDL generator
2. WSDL compiler
3. UDDI publisher

- businessEntity
  - businessService
    - bindingTemplate
      - tModel

Inquiry API
Publishers API

UDDI registry
Business Process Execution Language for Web Services (WS-BPEL)

• Allows specification of composition schemas of Web Services
  - Business processes as coordinated interactions of Web Services
  - Business processes as Web Services
• Allows abstract and executable processes
• Influenced from
  - Traditional flow models
  - Structured programming
  - Successor of WSFL and XLANG
• Component Web Services described in WS-DL (v1.1)
WS-BPEL Specification

An XML document specifying
• Roles exchanging messages with the composite service/process
• The (WSDL) interfaces supported by such roles

• The orchestration of the process
  – Variables and data transfer
  – Exception handling
  – Correlation information
Process Model
(Activities)

- **Primitive**
  - **invoke**: to invoke a Web Service (in-out) operation
  - **receive**: to wait for a message from an external source
  - **reply**: to reply to an external source message
  - **wait**: to remain idle for a given time period
  - **assign**: to copy data from one variable to another
  - **throw**: to raise exception errors
  - **empty**: to do nothing

- **Structured**
  - **sequence**: sequential order
  - **switch**: conditional routing
  - **while**: loop iteration
  - **pick**: choices based on events
  - **flow**: concurrent execution (synchronized by links)
  - **scope**: to group activities to be treated “transactionally” (managed by the same fault handler, within the same transactional context)

A link connects exactly one source activity S to exactly one target activity T; T starts only after S ends. An activity can have multiple incoming (possibly with join conditions) and outgoing links. Links can be guarded.
Process Model
(Data Manipulation and Exception Handling)

• Blackboard approach
  - a blackboard of variables is associated to each orchestration instance (i.e., a shared memory within an orchestration instance)
  - variables are not initialized at the beginning; they are modified (read/write) by assignments and messages
  - manipulation through XPath

• Try-catch-throw approach
  - definition of fault handlers
  - ... but also event handlers and compensation handlers (for managing transactionality as in the SAGA model)
From a TS to WS-BPEL (1)

Transition System

Mapping transitions

Transition Skeletons

Mapping states

State Skeletons

Connecting state skeletons on the basis of the graph

WS-BPEL Specification Skeleton

```xml
<process name = "...">
  <partnerLinks>
    ...
  </partnerLinks>
  <variables>
    ...
  </variables>
  <flow>
    <links>
      ...
    </links>
    <!-- state skel. -->
    ...
    <!-- state skel. -->
  </flow>
</process>
```
From a TS to WS-BPEL

(2)

Intuition [Baina et al. CAISE04, Berardi et al. VLDB-TES04]

1. Each transition corresponds to a WS-BPEL pattern consisting of (i) an `<onMessage>` operation (in order to wait for the input from the client of the composite service), (ii) followed by the effective logic of the transition, and then (iii) a final operation for returning the result to the client. Of course both before the effective logic and before returning the result, messages should be copied forth and back in appropriate variables.

2. All the transitions originating from the same state are collected in a `<pick>` operation, having as many `<onMessage>` clauses as transitions originating from the state.

3. The WS-BPEL file is built visiting all the nodes of the graph, starting from the initial state and applying the previous rules.

N.B.: (1) and (2) works for in-out interactions (the ones shown in the following). Simple modifications are needed for in-only, robust-in-only and in-optional-out. The other kinds of interactions implies a proactive behaviour of the composite service, possibly guarded by `<onAlarm>` blocks.
<onMessage ... >
  <sequence>
    <assign>
      <copy>
        <from variable="input" ... />  
        <to variable="transitionData" ... /> 
      </copy>
    </assign>
    <!-- logic of the transition -->
    <assign>
      <copy>
        <from variable="transitionData" ... /> 
        <to variable="output" ... /> 
      </copy>
    </assign>
    <reply ... />
  </sequence>
</onMessage>
State Skeletons

• N transitions from state $S_i$ are mapped onto:

```xml
<pick name = "$S_i$">
  <!-- transition #1 -->
  <onMessage ...
    <!-- transition skeleton -->
  </onMessage>

  ... ...

  <!-- transition #N -->
  <onMessage ...
    <!-- transition skeleton -->
  </onMessage>

</pick>
```
Mapping the TS

• All the `<pick>` blocks are enclosed in a surrounding `<flow>`; the dependencies are modeled as `<link>`s
  - `<link>`s are controlled by specific variables $S_i$-to-$S_j$ that are set to TRUE iff the transition $S_i \rightarrow S_j$ is executed
  - Each state skeleton has many outgoing `<link>`s as states connected in output, each going to the appropriate `<pick>` block
  - Transitions going back into the initial state should not be considered, as they can be represented as the start of a new instance
An Example (1)

```xml
<partnerLinks>
  <!-- The "client" role represents the requester of this composite service -->
  <partnerLink name="client"
    partnerLinkType="tns:Transition"
    myRole="MP3ServiceTypeProvider"
    partnerRole="MP3ServiceTypeRequester"/>
  <partnerLink name="service"
    partnerLinkType="nws:MP3CompositeService"
    myRole="MP3ServiceTypeRequester"
    partnerRole="MP3ServiceTypeProvider"/>
</partnerLinks>
```
An Example (2)

```xml
<variables>
  <variable name="input" messageType="tns:listen_request"/>
  <variable name="output" messageType="tns:listen_response"/>
  <variable name="dataIn" messageType="nws:listen_request"/>
  <variable name="dataOut" messageType="nws:listen_response"/>
</variables>

<pick>
  <onMessage partnerLink="client"
    portType="tns:MP3ServiceType"
    operation="listen"
    variable="input">
    <sequence>
      <assign>
        <copy>
          <from variable="input" part="selectedSong"/>
          <to variable="dataIn" part="selectedSong"/>
        </copy>
      </assign>
      ... ...
      <assign>
        <copy>
          <from variable="dataOut" part="MP3FileURL"/>
          <to variable="output" part="MP3FileURL"/>
        </copy>
      </assign>
      <reply name="replyOutput"
        partnerLink="client"
        portType="tns:MP3ServiceType"
        operation="listen"
        variable="output"/>
    </sequence>
  </onMessage>
  ... ...
</pick>
```
An Example (3)

```xml
<process suppressJoinFailure = "no">
  <flow>
    <links>
      <link name="start-to-1"/>
      <link name="start-to-2"/>
    </links>
    <pick createInstance = "yes">
      <onMessage="sa">
        <sequence>
          <copy>...</copy>
          ... ...
          <copy>...
          <reply ... />
        </sequence>
      </onMessage>
      <onMessage="st">
        <sequence>
          <copy>...
          ... ...
          <copy>...
          <reply ... />
        </sequence>
      </onMessage>
      <source linkName="start-to-1" transitionCondition = "bpws:getVariableData('start-to-1') = 'TRUE' " />
      <source linkName="start-to-2" transitionCondition = "bpws:getVariableData('start-to-2') = 'TRUE' " />
    </pick>
  </flow>
</process>
```

A new instance is created in the initial state. This resolve also the presence of the cycles without the need of enclosing `<while>`.

The `<sa>` transition skeleton should set variables:
- `start-to-1 = TRUE`
- `start-to-2 = FALSE`

The `<st>` transition skeleton should set variables:
- `start-to-1 = FALSE`
- `start-to-2 = TRUE`
An Example (4)

```xml
<pick>
  <onMessage="l">
    <sequence>
      <copy>...</copy>
      ... ... ...
      <copy>...</copy>
      <reply ... />
    </sequence>
  </onMessage>
  <target linkName="start-to-1" />
</pick>
<pick>
  <onMessage="l">
    <sequence>
      <copy>...</copy>
      ... ... ...
      <copy>...</copy>
      <reply ... />
    </sequence>
  </onMessage>
  <target linkName="start-to-2" />
</pick>
</process>
```
Consider a dance with more than one dancer

- Each dancer has a set of steps that they will perform. They orchestrate their own steps because they are in complete control of their domain (their body)
- A choreographer ensures that the steps all of the dancers make is according to some overall, pre-defined scheme. This is a choreography
- The dancers have no control over the steps they make: their steps must conform to the choreography
- The dancers have a single view-point of the dance
- The choreographer has a multi-party or global view-point of the dance
• Consider a jazz band with many players
  - There is a rhythm and a main theme. This is the choreography
  - Each player executes his piece by improvising variations over the main theme and following the given rhythm
  - The players still have a single view-point of the music; in addition they have full control over the music they play
  - There is a multi-party or global view-point of the music, but this is only a set of “sketchy” guidelines
**WS-BPEL vs. WS-CDL**

- Orchestration/WS-BPEL is about describing and executing a single peer.
- Choreography/WS-CDL is about describing and guiding a global model (N peers).
- You should derive the single peer from the global model by projecting based on participant.
The “Stacks” of Service Technologies

Registry/Repository & Discovery

Multiple Interacting Services

Single Service

Messaging

WSDL-based

RESTful

ebXML-based

Semantic-based
RESTful Services (1)

- REST refers to simple application interfaces transmitting data over HTTP without additional layers as SOAP
  - Web page meant to be consumed by program as opposed to a Web browser or similar UI tool
  - require an architectural style to make sense of them (the REST one), because there’s no smart human being on the client end to keep track
RESTful Services (2)

• Metaphor based on nouns and verbs
  - URIs ~ nouns
  - Verbs describe actions that are applicable to nouns
    • GET -- retrieve information / READ, SELECT
    • POST (PUT) – add/update new information / CREATE, INSERT, UPDATE
    • DELETE -- discard information / DELETE

• State means the application/session state, maintained as part of the content transferred (in XML) from client to server back to client
RESTful Services (3)

- REST is, in a sense, a kind of RPC, except the methods have been defined in advance
  - Consider the stock example of a remote procedure called "getStockPrice"
  - It's not clear what it means to GET, PUT, and POST to something called "getStockPrice"
  - But if we change the name from "getStockPrice" to "CurrentStockPrice" all is well!!
RESTful Services

OrderManagement
+ ID submitOrder(c_id: ID, o:Order)
+ void cancelOrder(id:ID)
+ Order getOrderDetail(id:ID)
+ Order[] getOrders(c_id: ID)

CustomerManagement
+ ID createCustomer(c:Customer)
+ Customer getCustomerDetail(id:ID)
+ Customer[] getCustomers()
RESTful Services (4)

- REST is incompatible with "end-point" RPC -- Either you address data objects or you address "software components"
  - REST does the former
  - End-point RPC does the latter

```plaintext
POST /purchase_orders HTTP/1.1
Host: accounting.mycompany.com
content-type: application/purchase-order+xml
....
<pо>...</pо>

POST /generic_message_handler
content-type: application/SOAP+xml
<soap:envelope>
  <soap:body>
    <submit-purchase-order>
      <destination>accounting.mycompany.com</destination>
      <pо>...</pо>
    </submit-purchase-order>
  </soap:body>
</soap:envelope>
```
# Example (1)

<table>
<thead>
<tr>
<th>Operation</th>
<th>HTTP Request</th>
<th>HTTP Response</th>
<th>Java Technology Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>POST /restfulwebservice-var/postservice/ HTTP/1.0</td>
<td>HTTP/1.1 200 OK</td>
<td>public PurchaseOrderStatus acceptFO(PurchaseOrder order)</td>
</tr>
<tr>
<td></td>
<td>Accept: <em>/</em></td>
<td>X-Powered-By: Servlet/2.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connection: close</td>
<td>Content-Type: text/xml</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Content-Type: text/xml</td>
<td>Date: Fri, 21 Jul 2006 17:07:15 GMT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Praga: no-cache</td>
<td>Connection: close</td>
<td></td>
</tr>
</tbody>
</table>

```xml
<tns:PurchaseOrderDocument xmlns:tns="urn:PurchaseOrderDocument">
  <billTo>
    <street>1 Main Street</street>
    <city>Beverly Hills</city>
    <state>CA</state>
    <zipCode>90210</zipCode>
  </billTo>
  <createDate>2004-03-27T12:21:02.056-05:00</createDate>
  <poID>AEC-CO-19282</poID>
  <items>
    <itemname>Copier Paper</itemname>
    <price>10</price>
    <quantity>2</quantity>
  </items>
  <items>
    <itemname>Toner</itemname>
    <price>$20</price>
    <quantity>1</quantity>
  </items>
  <shipTo>
    <street>1 Main Street</street>
    <city>Beverly Hills</city>
    <state>CA</state>
    <zipCode>90210</zipCode>
  </shipTo>
</tns:PurchaseOrderDocument>
```
<table>
<thead>
<tr>
<th>Request</th>
<th>Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /restfulwebservice-var/poService/ABC1153501634787</td>
<td>HTTP/1.1 200 OK</td>
<td>Retrieve PO (String orderID)</td>
</tr>
<tr>
<td><strong>Example (2)</strong></td>
<td><strong>Example (2)</strong></td>
<td><strong>Example (2)</strong></td>
</tr>
<tr>
<td>GET /restfulwebservice-var/poService/</td>
<td>HTTP/1.1 400 Bad Request</td>
<td>Indicates a problem finding the order</td>
</tr>
<tr>
<td>HTTP/1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection: close</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Type: text/xml</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```xml
<?xml version="1.0" encoding="UTF-8"?><ns4:POProcessingFault xmlns:ns4="urn:POProcessingFault" xmlns:ns2="urn:Status" xmlns:ns1="urn:POProcessingDocument">
<message>Unable to retrieve the order associated with the orderID you specified</message>
</ns4:POProcessingFault>
```
Example (3)

```
PUT /restfulwebservice-war/purchaseService/ HTTP/1.0
Connection: close
Content-Type: text/xml
Content-Length: 620
Pragma: no-cache

<?xml version="1.0" encoding="UTF-8"?>
<ns3:PurchaseOrderDocument
 xmlns:ns3="urn:PurchaseOrderDocument"
 xmlns:ns2="urn:Status"
 xmlns:ns4="urn:POProcessingFault">
 <billTo>
  <street>1 Main Street</street>
  <city>Beverly Hills</city>
  <state>CA</state>
  <zipCode>90210</zipCode>
 </billTo>
 <createDate>2004-03-27T12:21:02.055-05:00</createDate>
 <poID>ABC-CO-19282</poID>
 <items>
  <itemName>Copy Paper</itemName>
  <price>10</price>
  <quantity>2</quantity>
 </items>
 <items>
  <itemName>Toner</itemName>
  <price>920</price>
  <quantity>1</quantity>
 </items>
 <shipTo>
  <street>1 Main Street</street>
  <city>Beverly Hills</city>
  <state>CA</state>
  <zipCode>90210</zipCode>
 </shipTo>
</ns3:PurchaseOrderDocument>

DELETE /restfulwebservice-war/purchaseService/ABC-CO-19282 HTTP/1.0
Connection: close
Content-Type: text/xml
Content-Length: 0
Pragma: no-cache

<?xml version="1.0" encoding="UTF-8"?>

public void updatePO(PurchaseOrder order)
```

```java
public void updatePO(PurchaseOrder order) {
    // Update PurchaseOrder
}
```
Why so trendy?

- Easy and lightweight
- Amazon, Yahoo, Google offer their Web services as RESTful

... but nothing really new for us, basically the same abstractions apply, you can consider the operations as a whole or you can start modeling the data flowing through the service
References

Based on the 1st International Workshop on Technologies for e-Services (VLDB-TES 2001)
An early version of this paper was published in Proc. of CAiSE'03, LNCS 2681, pp. 468-484, 2003


References


References


