Standards and Best Practices on Designing Service Contracts

Prepared by

Enterprise Planning and Architecture Strategies Team

Control Page:

Revision History:			
Version No	**Revised Date**	**Author**	**Comments**
	05/05/2008	Anitha Ramakrishnan	Initial Draft
1.0	05/09/2008	Anitha Ramakrishnan	
1.1	09/10/2008	John Savastio	
1.2	12/08/2008	Anitha Ramakrishnan	Applied new standards template. Added more standards
1.3	01/02/2009	Anitha Ramakrishnan	Added another verb for process and operation naming

Acknowledgements:

None.

Table of Contents:

1 Introduction:

In the DOL P&T SOA development environment, SDLC standards documents are developed collaboratively by all parties impacted by and involved in the software development process. The Architecture Group (EPAS) coordinates the process to develop the standards, and then publishes the standards on their page of the P&T website. It is understood that given the ever-changing environment of software development, these standards are to be viewed as living documents. In addition, as experience evolves our best practices, it is expected that these standards documents will evolve as well.

SDLC staff may initiate a change request to any of these standards documents by contacting (in the form of an email), the manager of the Architecture Group.

1.1 Purpose:

The purpose of the document is to outline the standards and best practices to be followed while designing the service contracts for SOA based applications. The Service Contracts are perhaps the most fundamental and core architectural components of Service Oriented Architecture (SOA). Service contracts include both technical service descriptions designed for runtime consumption like WSDL, XML-Schema, WS-policy and non-technical documents like Service Level Agreements (SLA).

This document is intended for Enterprise Architects, Technical leads and Developers.

2 Standards and Best Practices:

The following are some of the general standards and best practices to be followed in designing service contracts.

- Service contracts should not be coupled to the logic of the parent process from which the service was identified.
- Details on service implementation should not be included in service contracts. This can be avoided by first designing the contract before implementing the logic, instead of generating the contracts from service implementation.
- Do not provide technology specific information in service contracts. For example, programming languages used to write the program, system resources (like oracle DB, legacy APIs) used by the program etc.
- As soon as the services are identified and have been defined conceptually, add it to WSRR. This could even be a draft service specification document. This will improve service discoverability. Add WSDL, XSD-schemas, WS-Policy (optional), Service specification documents and other related documents (If any) to WSRR as soon as they are created. Please refer to the WSRR usage and standards document (WSRR_SetUp_Admin_&Standards.doc) prepared by EPAS for more information on the same.

2.1 WSDL and SOAP Versions:

The following versions are current standards, as our environment does not have support for the latest version yet (WSDL 2.0).

WSDL 1.1 (http://schemas.xmlsoap.org/wsdl/)
SOAP 1.1 (http://schemas.xmlsoap.org/wsdl/soap/)

2.2 Namespaces:

Namespace gives a way to organize related elements into groups. Within a group, each element name should be unique. Although WSDL 1.1 does not mandate namespaces, at DOL we need to create namespaces for all WSDLs and XSDs.

DOL standards for namespaces are as follows:

- The name of a namespace must be a Uniform Resource Identifier.
- The namespace for WSDLs should be of the following format ***http://labor.state.ny.us/wsdl/domain_name/subdomain_name/WSDL_name***
  For example, a service called CalculateTaxRate that is defined in the sub domain of employertax in the UI (Unemployment Insurance) domain should have a namespace as follows: ***http://labor.state.ny.us/wsdl/ui/employertax/CalculateTaxRate***
- The namespace for XSDs should be of the following format ***http://labor.state.ny.us/schema/domain_name/subdomain_name/XSD_name***
  For example, a schema called EmployerTaxRating that is defined in the sub domain of employertax in the UI (Unemployment Insurance) domain should have a namespace as follows: ***http://labor.state.ny.us/schema/ui/employertax/EmployerTaxRating***
- All of the letters in a namespace should be in lowercase, except the last string that names the contract, which would be capitalized in the first letter of each word. Refer to Section 2.6 Naming Conventions for more details.
- No special characters or encoding are allowed except '/' and ':'.
- Namespaces are case sensitive. So do not use two names with just the case being different.
- No relative URL should be used for namespaces.
- Use an abbreviated prefix for namespaces.

2.3 Documentation:

Use a <documentation> tag to add comments at the top of each document to denote the purpose. Also use it to describe messages, ports, message pattern etc. Add a comment followed with a line of separation at the beginning of each section (such as Types, Messages, Ports, Bindings

and Services) of a WSDL document so that it is easy to read. Our standard comment for contracts is as follows and is defined at the top, right after the <xml> tag.

<!-- edited with WID by Anitha Ramakrishnan

WSDL for Employer Service
Version : 1.0
Authors : Anitha Ramakrishnan
Created : 07 October 2008

Revision History:
1.0 10/07/08 New wsdl definition

-->

2.4 Service Data Representation:

XSD schema is used to define the data model for messages. These data models should be defined and implemented separately from the service capabilities (operations) that utilize them to represent the structure and typing of message content. Schemas should be centralized by having a single "official" schema for every information set or entity (for example Employer, Address, Petition, etc). Please refer to the XSD standards document for more information on defining XSDs.

The data-types a service uses should always be placed in separate .xsd files, with their own namespaces, rather than directly defining them in the types section of a WSDL document. Import the xsd files in the ***types section*** of WSDLs using <xsd:import> tag. For example, the following section describes how the types are imported from an xsd file in WSDL.

```
<wsdl:types>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import
      namespace="http://labor.state.ny.us/schema/ui/employertax/tCalcul
      ateTaxRate" schemaLocation="tCalculateTaxRate.xsd"/>
      <xsd:import namespace="http://labor.state.ny.us/exception"
      schemaLocation="ErrorResponse.xsd"/>
</xsd:schema>
</wsdl:types>
```

The xsd:include element requires that the included schema have the same target namespace value as the schema that contains the xsd:include statement. The xsd:import element doesn't have this limitation and hence is preferred in the above scenario.

2.5 Standards on Messages, Ports, Bindings and Services:

Messages:

The messages tag of wsdl is used to define the input, output and faults for each operation in a service.

There can only be one <part> element defined inside each <message> tag in order to be compliant with WSI basic profile 1.0.

The following are the naming standards for message names for input, output and fault messages:

Input message : ***OperationNameRequest***
Output message : ***OperationNameResponse***
Fault message : ***OperationNameFault***

The following are the naming standards for part name in messages for input, output and fault messages:

Part name for Input message : ***OperationNameRequestValue***
Part name for Output message : ***OperationNameResponseValue***
Part name for Fault message : ***OperationNameFaultValue***

Note:

Although the best practice is to use generic names for part names, (for example RequestValue, ResponseValue, FaultValue), there are some issues and inconveniences in using this method in WID. Instead we have decided to add the name of the operation to prefix the generic partnames for inputs, outputs and faults.

A sample message definition in WSDL based on the above standards:

```
<wsdl:message name="CalculateEmployerTaxRateRequest">
    <wsdl:part name="CalculateEmployerTaxRateRequestValue"
element="calcschema:CalculateEmployerTaxRateRequest"/>
  </wsdl:message>

  <wsdl:message name="CalculateEmployerTaxRateResponse">
    <wsdl:part name="CalculateEmployerTaxRateResponseValue"
element="calcschema:CalculateEmployerTaxRateResponse"/>
  </wsdl:message>

  <wsdl:message name="CalculateEmployerTaxRateFault">
    <wsdl:part name="CalculateEmployerTaxRateFaultValue"
element="err:DOLCustomException" />
  </wsdl:message>
```

The <wsdl:part> tag should be defined with name and element attributes. The ***element*** attribute is required when WS style is ‘document/literal’ or ‘document/literal wrapped’.

There should always be fault defined for each operation and the element should be set to ***DOLCustomException***, which is a standard exception format thrown by all services. Please refer to our Exception Handling Standards document for more information on faults.

Ports, Bindings and Services:

All the operations in a service must be defined in the portType section with their inputs, outputs and faults.

The WS style to be used for WSDLs can either be document / literal or document / literal wrapped. RPC/encoded style must not be used.

In the services section of the WSDL, provide the endpoint of the service in the address location attribute using a qualified http url (if it's soap/http), and do not use the IP addresses of the service providers. In case of JMS, provide the queue information.

Note:

The naming standards for all names used in a WSDL should follow the standards defined in Section 2.6 Naming Conventions of this document.

2.6 Naming Conventions:

Following are the general naming standards for services, types (xsd elements), service operations, porttypes etc. In other words, these basic naming conventions should be followed for all elements defined in a service contract:

- The first letter should be capitalized.
- If the name includes two words, capitalize the second word's first letter as well without any special characters between the first and second words. For example LegalName, MailingAddress etc.
- Do not use any special characters or spaces in the name.
- If using acronyms or abbreviations in the name, capitalize all letters. For example, employer identifier should be defined as EmployerID and Federal identification number should be defined as FEIN.

2.6.1 Standards for Naming Services:

Service names should convey the purpose and requirements of individual capabilities belonging to the service. A service name should clearly establish a meaning and a context in the business. Service names should not use redundant words like 'service', or the name of the logical layer that

it belongs to such as 'task', 'entity' or 'process' (this is because these layers are used mainly by IT in service models, and are purely logical and don't have any business meaning or value).

Utility Service:

The utility services involve operations that encapsulate common functionality used across applications / services in an enterprise (For example logging, notification etc). Our standard is to name utility services with a **verb.** For example "Notify" or "Log."

In the event that clarification or differentiation is needed between two or more services in the utility layer that perform the same type of action, and therefore need to use the same verb name, adding a noun to the front of the verb to form a **NounVerb** pairing is allowable. An example would be "EmployerLog."

Entity Service:

Entity services represent a specific business entity and their names are often predetermined by the entity name. Our standard is to name the entity services with a **noun.** For example, "Customer", "Employer", "Invoice" etc.

Task Service / Sub-process:

Task service operations encapsulate process logic (which could be just a few steps in the process and may vary in granularity). In a task service, the thread that ties together the grouped operations is a specific activity being automated by the service logic and should be named after that activity. For example, SearchEmployer is a task service that could group operations like SearchbyFEIN, SearchbyName etc. Another example would be if a task / process service just groups the CRUD (create,read,update,delete) operations on an entity, the service could be named using the verb 'Manage' i.e. if a task service groups CRUD operations on the entity Employer, the service could be named 'ManageEmployer'. Accordingly, our standard to name a task service is the **VerbNoun** format.

Note: An entity (e.g Employer) could have multiple task services, therefore naming the task services with '**entitynameTask**' format (e.g EmployerTask) should be avoided. The problem would be that following this convention; the second task service on the same entity will have to named EmployerTask2, which is too vague.

Orchestrated Task / Process Service:

While similar in their coordination nature to task services, orchestration/process services function at a higher level and as such are coarser than their lower level counterparts. However, due to their common niche, they will share the same naming convention, using the **VerbNoun** format.

2.6.2 Standards for Naming Service Operations across all service layers:

- Operation names should not duplicate words from their parent service name (Since a properly defined service name clearly establishes a meaning and a context). For example, if the Service name is ExperienceRatingAccount, the operations could just be named like GetAccount, AddAccount etc.

- Each operation name will consist of a two-word **VerbNoun** pair, with the verb always coming first.

- Standard operation verbs for interfacing with databases:

  Add - Inserts a new row into the database table(s).
  Search - Processes a request from the user that contains search keys that are not expected to return one exact match, but instead several matches that are intended to be displayed to the user as a list of choices.
  Check - Checks a database table(s) to find if a record exists or not. A Boolean indicating either a found or not found value is returned.
  Get - Retrieves one record (an exact match) from the database table(s) based on specific search criteria.
  Get….List - Retrieves one or more records from the database table(s) based on criteria (where List is a suffix following the noun)
  Modify - Makes changes to existing database row(s).
  Delete - Removes row(s) from the database table(s) or changes the row's status to 'delete'.

- Standard operation verbs for non-database actions:

  Calculate –
  Verify -

- Note: The above are "living" lists and as such it is understood that new verbs may be added or the meanings of existing verbs may be modified over time.

- Operation verbs will be standard across all types of services (i.e. utility, entity etc)..

2.7 Service Contract Versioning:

Versioning can be used to distinguish between changes to a service contract. The service versions should be represented as **<major>.<minor>**. The initial version should always be 1.0, until the service is released in production environment. In an xsd schema, use the xsd:schema element's version attribute to set the version. For example, <xsd:schema version="1.0">

In the case of a WSDL, the WSDL definitions element does not provide a built-in version attribute and hence use a standard comment at the top of the document to specify the version. For example:

<!-- edited with WID by Anitha Ramakrishnan

WSDL for Employer Service
Version : 1.0
Authors : Anitha Ramakrishnan
Created : 07 October 2008

Revision History:
1.0 10/07/08 New wsdl definition

-->

2.7.1 Minor Version Increments:

A compatible change in contracts that does not affect existing service consumers just increases the minor number. For example, a compatible change will be represented using ***<major>.<minor + 1>***.

Examples of minor version change in WSDL:

- Adding a new operation
- Adding another binding

Examples of minor version change in XSD Schema:

- Adding an optional element.

2.7.2 Major Version Increments:

An incompatible change in contracts that breaks existing consumers' backward compatibility increases the major number. For example, an incompatible change will be represented using ***<major + 1>.<0>***.

Examples of minor version change in WSDL:

- Changing the message exchange pattern of an existing operation. For example, changing a one-way operation to request-response.
- Changing existing bindings.

Examples of major version change in XSD Schema:

- Removing an existing schema component
- Changing an optional element to 'Required'

Note:

In the case of a major version change, the target namespace changes by reflecting the version. For example, the target namespace could be changed to "http://labor.state.ny.us/wsdl/ui/employertax/TaxRatingTransaction/v2". By encoding only the major release number in the namespace, successive minor releases share the same namespace and so are compatible. The exception to this rule is when the first version of a service contract is released, in which case no version number is added to the namespace.

2.8 WSI-compliance:

Although 100% WSI-compliance may be sometimes impossible (based on the requirement), we need to try to be as compliant as possible with WS-I Basic Profile 1.0 standards. There are tools like SOAPUI, or even Web services toolkit in RAD, that could be used to set and test the

compliance while creating the WSDL. Our above-mentioned standards also enforce the compliance to WSI.

2.9 WS-Policy:

Policy assertions can be defined and attached to various parts of a WSDL. This is an optional document for service contracts.

Policies should be centralized and a base policy containing broad, generalized assertions must be created. More specialized assertions can be placed into separate policy definitions that can then be attached to the same WSDL definition.

Ws-Policy will be covered more in detail in our SOA security document.

2.10 Non-technical Service Contracts:

Service Level Agreements (SLAs) are non-technical documents of service contracts that define the additional quality-of-service features, availability, accessibility, performance, behaviors, and limitations etc of a service. Although an SLA could be created for internal services, our standard is to create SLAs only for services that we expose to external agencies.

A format / template for SLAs can be found in EPAS's SOA document templates directory.

2.11 Procedures for Centralizing schemas and Contracts:

Service contracts should be reviewed by the Enterprise Architecture Team, especially for centralizing schema definitions and to analyze the reuse potential of existing schemas defined for various projects across the enterprise.

Service contracts can be created using WID / RAD / RSA. If the business and technical implementation model are done in WBM, importing the model into WID/RSA will automatically create WSDL and XSDs if the services are already defined and the business items for each service are also defined. These artifacts can be used as a starting point and should be reviewed and made to comply with our standards by an enterprise architect.

Also refer to the XSD standards document on the ratification process of XSD schemas using candidate schemas for the base vocabularies in the enterprise.

2.12 Service Granularity:

3 Tips and Techniques:

Use WS-I compliance tester in RAD/WID/RSA to test the level of compliance of WSDLs to Basic Profile 1.0.

4 Appendix:

4.1 Examples / Sample Diagrams:

The following is a sample WSDL following the above mentioned standards:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with WID by Anitha Ramakrishnan
_____________________________________________________
WSDL for CalculateTaxRate Service
Version : 1.0  Authors : Anitha Ramakrishnan    Created : 18 July 2008

      _____________________________________________

Revision History:
1.0  7/18/08 New wsdl definition
_____________________________________________________                -->
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:calc="http://labor.state.ny.us/wsdl/ui/employertax/CalculateTaxRa
te"
xmlns:calcschema="http://labor.state.ny.us/schema/ui/employertax/tCalcu
lateTaxRate" xmlns:err="http://labor.state.ny.us/exception"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://labor.state.ny.us/wsdl/ui/employertax/Calculate
TaxRate">

<!-- ***************************   Types ****************************
-->

  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import
namespace="http://labor.state.ny.us/schema/ui/employertax/tCalculateTax
Rate" schemaLocation="tCalculateTaxRate.xsd"/>
            <xsd:import namespace="http://labor.state.ny.us/exception"
schemaLocation="ErrorResponse.xsd"/>
    </xsd:schema>
  </wsdl:types>


 <!-- ***************************   Messges ************************
-->
```

```
  <!-- ******** CalculateEmployerTaxRate *********** -->

  <wsdl:message name="CalculateEmployerTaxRateRequest">
    <wsdl:part name="CalculateEmployerTaxRateRequestValue"
element="calcschema:CalculateEmployerTaxRateRequest"/>
  </wsdl:message>

  <wsdl:message name="CalculateEmployerTaxRateResponse">
    <wsdl:part name="CalculateEmployerTaxRateResponseValue"
element="calcschema:CalculateEmployerTaxRateResponse"/>
  </wsdl:message>

  <wsdl:message name="CalculateEmployerTaxRateFault">
    <wsdl:part element="err:DOLCustomException"
name="CalculateEmployerTaxRateFaultValue"/>
  </wsdl:message>


  <!-- ***************************   Ports ***************************
-->

   <!-- ******** CalculateEmployerTaxRate *********** -->
  <wsdl:portType name="CalculateTaxRate">
    <wsdl:operation name="CalculateEmployerTaxRate">
           <wsdl:input message="calc:CalculateEmployerTaxRateRequest"
name="CalculateEmployerTaxRateRequest"/>
           <wsdl:output
message="calc:CalculateEmployerTaxRateResponse"
name="CalculateEmployerTaxRateResponse"/>
           <wsdl:fault message="calc:CalculateEmployerTaxRateFault"
name="CalculateEmployerTaxRateFault"/>
    </wsdl:operation>

  </wsdl:portType>

  <!-- ***************************   Bindings ************************
-->

  <wsdl:binding name="CalculateTaxRateSOAP"
type="calc:CalculateTaxRate">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

   <!-- ******** CalculateEmployerTaxRate *********** -->
    <wsdl:operation name="CalculateEmployerTaxRate">
      <soap:operation soapAction="CalculateEmployerTaxRate"/>
      <wsdl:input name="CalculateEmployerTaxRateRequest">
        <soap:body parts="CalculateEmployerTaxRateRequestValue "
use="literal"/>
      </wsdl:input>
      <wsdl:output name="CalculateEmployerTaxRateResponse">
        <soap:body parts="CalculateEmployerTaxRateResponseValue"
use="literal"/>
      </wsdl:output>
      <wsdl:fault name="CalculateEmployerTaxRateFault">
           <soap:fault parts="CalculateEmployerTaxRateFaultValue"
use="literal"/>
```

```
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

 <!-- ****************************   Services ************************
-->

  <wsdl:service name="CalculateTaxRate">
    <wsdl:port binding="calc:CalculateTaxRateSOAP"
name="CalculateTaxRateSOAP">
      <soap:address
location="http://uitaxrate.stage.labor.state.nyenet/RC_WS4/services/Cal
culateTaxRateSOAP"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

4.2 Abbreviations Used:

None.

4.3 Acronyms Used:

WSDL	Web Services Definition Language
XML	Extensible Markup Language
WSI	Web Services Interoperability
SOAP	Simple Object Access Protocol
DOL	Department of Labor
RSA	Rational Software Architect
RAD	Rational Application Developer
WID	Websphere Integration Developer

5 References:

Erl, Thomas. Web Service Contract Design and Versioning. Upper Saddle River, NJ: Prentice Hall, 2009.

5.1 Bibliography:

None.