JAX-WS 2.2 : Web Services Addressing 1.0 - Metadata Proposal

1 Introduction

Web Services Addressing 1.0 - Metadata defines "how the abstract properties defined in Web Services Addressing 1.0 - Core are described using WSDL, how to include WSDL metadata in endpoint references, and how WS-Policy can be used to indicate the support of WS-Addressing by a Web service". By incorporating the metadata specification, JAX-WS 2.2 will have a complete Web Services Addressing - 1.0 support. The benefits are:

- In JAX-WS 2.1, a service and a client need to explicitly use @Addressing/AddressingFeature to enable addressing. That will not be required anymore in JAX-WS 2.2. Similarly, no need to explicitly use @Action/@FaultAction and BindingProvider.SOAPACTION_URI_PROPERTY to specify wsa:Action value.
- A JAX-WS 2.2 service can publish a WSDL with addressing requirements, so that any client could use the addressing metadata in the WSDL to communicate with the service.
- EndpointReference.getPort() , Service.getPort() methods can use WSDL metadata in EPR to create proxies.

1.1 WSDL-->Java : Mapping of wsam:Action to SEI method

This mapping tries to capture wsam:Action information in WSDL on SEI methods. The mapping is carried out irrespective of whether addressing is enabled or not.

1.1.1 Proposal

SEI Methods MUST be annotated with @Action and @FaultAction annotations for the corresponding wsdl:input, wsdl:output and wsdl:fault messages that contain wsam:Action attributes

- If a wsdl:input element contains a wsam:Action attribute, the value of the attribute MUST be set to the input element of @Action
- If a wsdl:output element contains a wsam:Action attribute, the value of the attribute MUST be set to the output element of @Action
- If a wsdl:fault element contains a wsam:Action attribute, the value of the attribute MUST be set to the value element of @FaultAction. The className element of @FaultAction MUST be the exception class name associated with the wsdl:fault

For example:

```
<operation name="getPrice">
  <input message="tns:getPrice" />
  <output message="tns:getPriceResponse" wsam:Action="outAction"/>
  <fault name="InvalidTickerException" message="tns:InvalidTickerException" wsam:Action="faultAction"/>
</operation>

// the mapped java method
@Action(output="outAction",
        fault={ @FaultAction(className=InvalidTickerException.class, value="faultAction") }
)
float getPrice(String ticker) throws InvalidTickerException;
```

Advantages :

- wsdl:portType information is captured in SEI method.
- More and more implementations are adding wsam:Action on all operations by default. So introspection into WSDL is not required to compute wsa:Action.

Disadvantages :

- When wsam:action is not specified, the runtime still needs to introspect WSDL while computing wsa:Action.

1.2 Java-->WSDL

1.2.1 Mapping of @Addressing

The metadata specification supports a mechanism to indicate the use of addressing in WSDL description using WS-Policy framework. We can use @Addressing to map the use of addressing information when a WSDL is generated. @Addressing API need to be modified to support all the use cases as specified in the section3 of metadata. The proposed @Addressing API (note the addition of responses annotation element) is as follows:

```
public @interface Addressing {
    /**
     * Specifies if this feature is enabled or disabled. If enabled, it means the
     * endpoint supports WS-Addressing but does not require its use. Corresponding
     * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicyaddressing">
     * 3.1.1 Addressing Assertion</a> must be generated in the generated WSDL.
     */
    boolean enabled() default true;

    /**
     * If addressing is enabled, this property determines whether the endpoint
     * requires WS-Addressing. If required is true, the endpoint requires
     * WS-Addressing and WS-Addressing headers MUST
     * be present on incoming messages. A corresponding
     * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicyaddressing">
     * 3.1.1 Addressing Assertion</a> must be generated in the WSDL.
     */
    boolean required() default false;

    /**
     * If addressing is enabled, this property determines if endpoint requires
     * the use of anonymous responses, or non-anonymous responses, or all.
     *
     * <p>
     * {@link Responses#ANONYMOUS} requires the use of only anonymous
     * responses. It will result into wsam:AnonymousResponses nested assertion
     * as specified in
     * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicyanonresponses">
     * 3.1.2 AnonymousResponses Assertion</a> in the WSDL.
     *
     * <p>
     * {@link Responses#NON_ANONYMOUS} requires the use of only non-anonymous
     * responses. It will result into
     * wsam:AnonymousResponses nested assertion as specified in
     * <a href="http://www.w3.org/TR/ws-addr-metadata/#wspolicynonanonresponses">
     * 3.1.3 NonAnonymousResponses Assertion</a> in the generated WSDL.
     *
     * <p>The default value supports all response types. Specifying both
     * {@link Responses#ANONYMOUS}, and {@link Responses#NON_ANONYMOUS }
     * would mean that the endpoint supports all response types. Similary,
     * not specifying any one of them would mean that the endpoint supports
     * all response types.
     *
     * @since JAX-WS 2.2
```

```
     */
    Responses[] responses() default { Responses.ANONYMOUS, Responses.NON_ANONYMOUS };

}
```

@Addressing API defines the rules to generate the corresponding policy to indicate the use of WS-Addressing in a generated WSDL. A JAX-WS implementation MUST generate policies in the WSDL for @Addressing annotated endpoints.

Some of the examples are :

@Adressing annotation	Possible Policy Assertion in the generated WSDL
`@Addressing(enabled=false)`	None
`@Addressing(enabled=false,required=true)`	Error

`@Addressing`	`<wsam:Addressing wsp:Optional="true">`
	`  <wsp:Policy/>`
	`</wsam:Addressing>`

`@Addressing(required=true)`	`<wsam:Addressing>`
	`  <wsp:Policy/>`
	`</wsam:Addressing>`

`@Addressing(`
`  required=true,`
`  responses=Responses.ANONYMOUS)`
	`<wsam:Addressing>`
	`  <wsp:Policy>`
	`  <wsam:AnonymousResponses/>`
	`  </wsp:Policy>`
	`</wsam:Addressing>`

`@Addressing(`
`  responses=Responses.NON_ANONYMOUS)`
	`<wsam:Addressing wsp:Optional="true">`
	`  <wsp:Policy>`
	`  <wsam:NonAnonymousResponses/>`
	`  </wsp:Policy>`
	`</wsam:Addressing>`

1.2.2 Mapping of SEI method to wsdl:operation

A SEI method is mapped to a wsdl:operation in the generated WSDL. Clients use this WSDL to infer wsa:Action for a wire-level message when addressing is enabled. The metadata specification defines explicit and defaulting mechanisms to associate a value of the wsa:Action with input, output and fault elements within a wsdl:operation.

The challenge is to define a mapping so that all the JAX-WS implementations create a wsdl:operation such that the clients infer the same wsa:Action values. Otherwise, a service is not portable across JAX-WS implementations. The mapping can be achieved as follows:

- Mapping of wsam:Action using @Action and @FaultAction. Then wsa:Action can be derived using explicit mechanism of metadata specification.
- Use the default mechanism of metadata specification to compute wsa:Action. JAX-WS/JSR 181 didn't define values for the name attributes on input, output and fault elements within a wsdl:operation. But these names are used in the computation of the default action pattern algorithm. This means the computed wsa:Action is different across different implementations. Hence, defining some ways to restrict name attribute values on these elements.
- A combination of the above two.

This mapping needs to be carried out whether the addressing is enabled or not.

1.2.2.1 Proposal

The algorithm is as follows:

- A SEI method annotated with a @Action(input=...) or @WebMethod(action=...) MUST result into wsdl:input[@wsam:Action] attribute in the corresponding wsdl:operation. The attribute value MUST be same as the input annotation element value. Also, @Action(input=...) and @WebMethod(action=...) annotation element values MUST be same, if present.
- A SEI method annotated with a @Action(output=...) MUST result into wsdl:output[@wsam:Action] attribute in the corresponding wsdl:operation. The attribute value MUST be same as the output annotation element value.
- A SEI method annotated with @Action(@FaultAction=...) MUST result into wsdl:fault[@wsam:Action] attribute in the corresponding wsdl:operation. The wsdl:fault element MUST correspond to the exception specified by className annotated element value and its wsam:Action attribute value MUST be same as the value annotation element value.
- If wsdl:input[@wsam:Action] cannot be mapped from above steps, then wsam:Action is generated using the metadata defaulting algorithm as if wsdl:input[@name] is not present in WSDL.
- If wsdl:output[@wsam:Action] cannot be mapped from above steps, then wsam:Action is generated using the metadata defaulting algorithm as if wsdl:output[@name] is not present in WSDL.
- If wsdl:fault[@wsam:Action] cannot be mapped from above steps, then wsam:Action is generated using the metadata defaulting algorithm as if wsdl:fault[@name] is the corresponding exception class name.

For example:

```
@Action(input="inAction")
public float getPrice(String ticker) throws InvalidTickerException;

// the mapped wsdl:operation if targetnamespace="http://example.com" and porttype="StockQuoteProvider"
<operation name="getPrice">
  <input name="foo" message="tns:getPrice" wsam:Action="inAction"/>
  <output name="bar" message="tns:getPriceResponse" wsam:Action="http://example.com/StockQuoteProvider/
getPriceResponse" />
  <fault name="FooTickerException" message="tns:InvalidTickerException"
wsam:Action="http://example.com/StockQuoteProvider/getPrice/Fault/InvalidTickerException"/>

</operation>
```

Some of the disadvantages are:

- wsam:Action is always present on all input, output, fault elements within wsdl:operation

1.3 WSDL Metadata in EPR

JAX-WS implementation MUST understand WSDL Metadata in EPR as specified by the Section 2 of the metadata specification.

- When BindingProvider#getEndpointReference() returns W3CEndpointReference EPR, the EPR MUST contain wsam:ServiceName and wsam:ServiceName[@EndpointName]. The wsam:InterfaceName MAY be present in the EPR. If there is an associated WSDL, then the WSDL location MUST be referenced using wsdli:wsdlLocation in the EPR's wsa:Metadata.
- When WebServiceContext#getEndpointReference() methods return W3CEndpointReference

EPR, the EPR MUST contain wsam:ServiceName and wsam:ServiceName[@EndpointName]. The wsam:InterfaceName MAY be present in the EPR. If there is an associated WSDL, then the WSDL location MUST be referenced using wsdli:wsdlLocation in the EPR's wsa:Metadata.

- Service#getPort(EndpointReference, ..), Service.createDispatch(EndpointReference, ...) methods return a proxy for the endpoint specified by the EPR. The EPR's wsam:ServiceName MUST match the Service instance's ServiceName. EPR's EndpointName MUST be used as the PortName of the SEI. If the Service instance has an associated WSDL, its WSDL MUST be used to determine any binding information, any WSDL in the EPR will be ignored. If the Service instance does not have a WSDL, then any WSDL referenced in the EPR will be used to determine binding information.
- EndpointReference#getPort(EndpointReference, ..) method returns a proxy for the endpoint specified by the EPR. The EPR's wsam:ServiceName, and wsam:EndpointName MUST be used to create the proxy. EPR's referenced WSDL location MUST be used to determine binding information.
- W3CEndPointReferenceBuilder methods MUST also reflect wsam:ServiceName, wsam:ServiceName[@EndpointName], wsdli:wsdlLocation in the constructed EPR.

1.4 Runtime Behaviour

The runtime requirements are mentioned below:

- wsa:Action value MUST be used from @Action annotation elements in SEI, if present. But if client sets BindingProvider.SOAPACTION_URI_PROPERTY then that MUST be used for wsa:Action header.
- The use of Addressing is determined using this precedence order: deployment descriptors, @Addressing/AddressingFeature , and WSDL.
- BindingProvider will have TO_PROPERTY, FAULTTO_PROPERTY, REPLYTO_PROPERTY so that clients can populate wsa:To, wsa:ReplyTo, wsa:FaultTo headers.