MTOM Support on the WSDL Level

MTOM Support on the WSDL Level

To enable MTOM, the slight modification of WSDL file is needed.

```
<schema targetNamespace="http://pictures.com"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <element name="Picture">
    <complexType>
      <sequence>
        <element name="Title" type="xsd:string"/>
        <element name="ImageData" type="xsd:base64Binary"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

...is going to change to:

```
<schema targetNamespace="http://pictures.com"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xmime="http://www.w3.org/2005/05/xmlmime">
  <element name="Picture">
    <complexType>
      <sequence>
        <element name="Title" type="xsd:string"/>
        <element name="ImageData" type="xsd:base64Binary"
           xmime:expectedContentTypes="application/octet-stream"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

(Source: http://cxf.apache.org/docs/mtom-attachments-with-jaxb.html)

In other words, only new xml attribute expectedContentTypes from namespace with URI http://www.w3.org/2005/05/xmlmime is set to the value application/octet-stream.

Affected API-M Methods

- ingest
- getObjectXML
- export (response)
- modifyDatastreamByValue

There is little discrepancy between *API-M* WSDL definition of method *modifyDatastreamByValue* and it's web definition on (https://wiki.duraspace.org/display/FCR30/API-M#API-M-modifyDatastreamByValue). WSDL says attribute *dsContent* is of type **byte[]**, but web says String. I suppose, the **byte[]** as the correct type. *(You're correct, Jiri. I've fixed the documentation. - Chris)*

Affected API-A Methods

- getDatastreamDissemination

Methods marked as red has an input parameter of type **byte[]** and blue methods has output value of type **byte[]**.

Changes to WSDL

Since *API-A* and *API-M* wsdl files both use the include directive inside their **<types>** element, the changes will be made only to *fedora-types.xsd*.

Two sides of MTOM

MTOM can be enabled on both the client and the server side. If it is done on the client side, then the SOAP requests with binary data are "MTOMized" and if it is done on the server side, then the responses are. So far, it seems that either client or server (generated by CXF) can handle both MTOM (multipart /related) and non-MTOM (text/xml) SOAP messages in runtime without knowing in advance what type should be received. I am affraid that some old WS client won't be able to provide such a functionality, therefore new set of methods with MTOM support has to be created in order not to break backward compatibility wtih MTOM clients not understanding the MTOM/XOP coding. Another backward compatible solution could be for instance to return MTOMed message if and only if there is some special attribute in HTTP header of SOAP request, however, I haven't found how to change this behaviour of WS programmatically in runtime.

Also there might be some complication with this issue http://stackoverflow.com/questions/2808967/mtom-request-non-mtom-response.

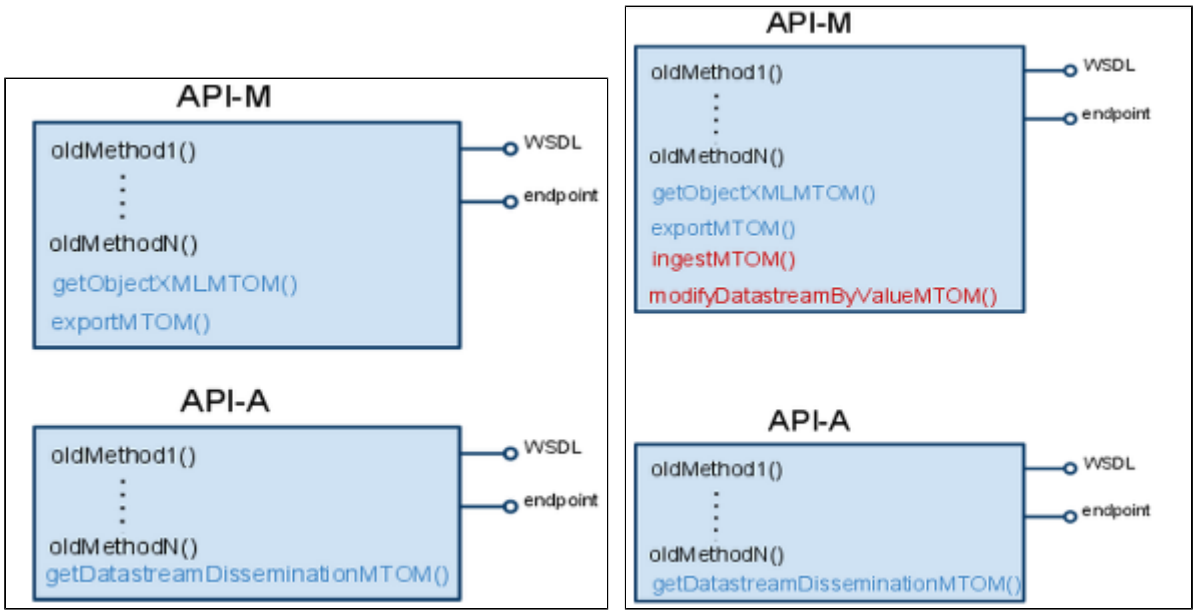
Possible Solutions

All the solutions are backward compatible.

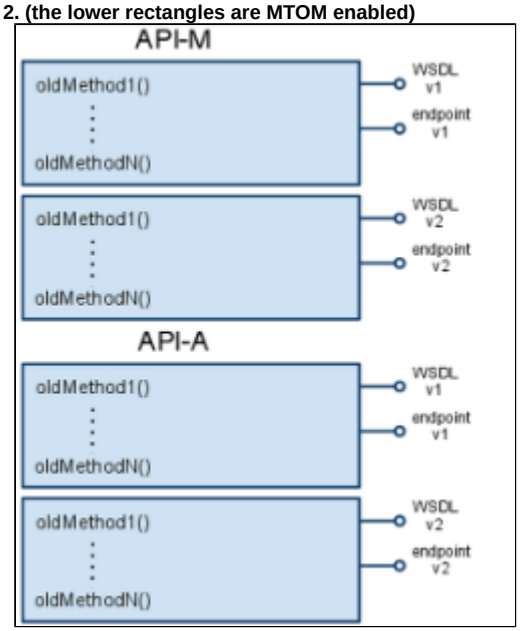
1. **One WSDL rule them all**
    a. MTOMize only "blue" operations
    b. MTOMize all MTOMizable operations
2. **New WSDLs and endpoints for MTOMized version of API**
3. **Routing (Mediator Pattern)**

Visualization in pseudo UML

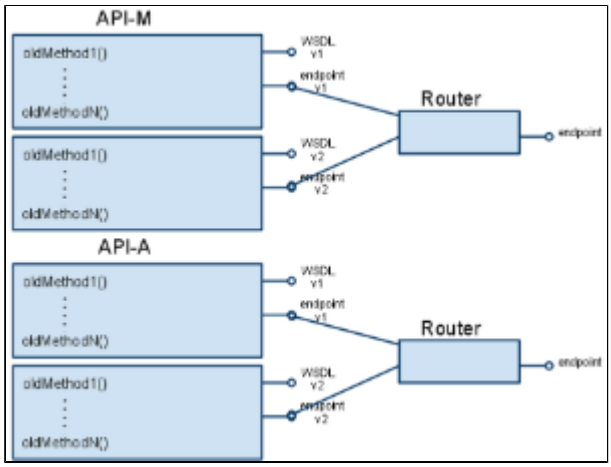
1.a & **1.b**



2. (the lower rectangles are MTOM enabled)



3.



Pros:

- 1.a
  - The easiest solution
  - The smallest set of operations in the API.
  - No need for new endpoints/wsdls
- 1.b
  - Consistent (all MTOM operation has "MTOM" in name)
  - Still easy
  - No need for new endpoints/wsdls
- 2.
  - 100% guarantee of backward compatibility
  - This method is the best practise, when doing backward compatible changes to WS API
- 3.
  - The WS endpoints remain the same and can handle all versions of API based on the particular namespace (now the namespace inside the SOAP messages is http://www.fedora.info/definitions/1/0/types/. I suggest something like http://www.fedora.info/2011/7/[definitions/types/|http://www.fedora.info/definitions/2011/7/29/types/] (W3C convetion)).
  - will not break "low lvl clients" i.e. clients that are not generated from wsdl like curl, because the endpoint is still the same and if such "low lvl client" wants to start use new version of WS the only change for him is to change the structure of SOAP by changing the namespace. In the Java/.NET world this is a minor advantage, but if the client is written in, say, bash, it may help.
  - probably most robust solution with keeping in mind the possible future changes to API (once the infrastructure around is implemented, the addition of new version of API is easier)

Cons:

- 1.a

    - It is not transparent for the client when calling, say, ingest() whether the request is MTOMized or not, since the server-side can handle both. This will hold also for method ingestMTOM() from solution 1b).
    - Not consistent in that sense that not all methods which use MTOM have "MTOM" suffix in their name.
- 1.b
    - More operations
- 2.
    - 4x wsdl (2x API-A + 2x API-M) + 4x endpoint
    - More complicated
    -
    - There is need to maintain more methods which will be probably the object of deprecation sometimes in the future
- 3.
    - 4x wsdl (2x API-A + 2x API-M) + 6x endpoint
    - More complicated, the know how of CXF is needed to implement the router as an interceptor

The Winner

To be honest, I don't know. I would suggest the solution number 3. i.e. the router and new WSDLs, because it seems to me the best choice for possible future changes to Fedora's API. I had also some discussion with my friends which support this proposal.

Namespaces

suggestion:

prefix	old	new
fedora-types	http://www.fedora.info/definitions/1/0/types/	http://fedora-commons.org/2011/07/definitions/types/
fedora-api	http://www.fedora.info/definitions/1/0/api/	http://fedora-commons.org/]2011/07/definitions/api/

References

- http://www.ibm.com/developerworks/webservices/library/ws-version/
- http://en.wikipedia.org/wiki/Mediator_pattern
- http://cxf.apache.org/docs/service-routing.html
- http://marek.potociar.net/2009/10/19/custom-metro-tube-interceptor (Metro)