WSDM/WS-Man Reconciliation

An Overview and Migration Guide

Version 1.0

August 2006

IBM

Abstract

As IBM, HP, Intel and Microsoft work on the reconciliation of WSDM and WS-Man, this document will present an overview of the current status of the work. It covers a wide range of topics, starting from very high-level discussions on the impact of this work, all the way down to the specific coding changes that may be seen by WSDM developers with the intention of answering the many questions people may have.

As the reconciliation effort progresses, this document will be updated as needed – providing an update-to-date reference and migration guide for those interesting in this effort.

Contents

1. Executive Summary

On March 15, 2006, HP, Intel, IBM and Microsoft announced[1] the intention to reconcile the WSDM and WS-Man specifications into a single standard for management of system resources using Web services. As the work reaches certain milestones, portions of it will be shared with the Web service community to solicit feedback. This document summarizes the first of these milestones: the publication of a first draft of Web Services Resource Transfer (WS-RT) specification[2], the publication of the Service Modeling Language specification[3] and an update to the WS-Metadata Exchange (WS-MEX) specification[4].

Both WSDM and WS-Man profile the use of lower-level specifications for use in the management domain, so the reconciliation of these two management specifications can not happen without first reconciling the infrastructural specifications each are built upon. One of the key components in this is the ability to access and manipulate resources. WSDM uses the OASIS family of specifications known as WS-Resource Framework, while WS-Man uses WS-Transfer for this purpose. The reconciliation work merged these two to produce a new specification called WS-Resource Transfer. WS-RT introduces features that will leverage the extension points in WS-Transfer to allow for a more optimal use in not only the management domain but general purpose resource manipulation as well.

The key features introduced in WS-Resource Transfer are:

- Partial resource manipulation. WS-RT allows for a subset of a resource's data to be manipulated – allowing for a much more performant message exchange.
- Definition of data access languages that are better suited for low-end devices.
- A tighter relationship between a resource and its metadata.

The Service Modeling Language (SML) specification profiles how resources are to be modeled and how to represent relationships between those resources. By bringing together the "resource manipulation" features of WSDM and WS-Man together into one specification, and agreeing on the modeling language for those resources, this effort will, in the long-run, simplify the Service Oriented Architecture landscape and accelerate the adoption of Web services based solutions.

In conjunction with the publication of the WS-RT specification, an update of WS-MetadataExchange is also being published. This update better aligns it with the reconciliation work by reusing the WS-Transfer Get operation to retrieve the representation of a "metadata" resource, instead of defining its own operation for this purpose.

[1] http://www.ibm.com/developerworks/webservices/library/specification/ws-roadmap/
[2] http://www.ibm.com/developerworks/webservices/library/specification/ws-wsrt/
[3] http://www.ibm.com/autonomic/service_modeling_language_spec.html
[4] http://www.ibm.com/developerworks/webservices/library/specification/ws-mex/

Current WSDM users can be assured that IBM is working on tools to smoothly transition them to the new specifications. Once the reconciliation work is complete, current customers and vendors of WSDM will continue to have access to all of the features that the WSDM family of specifications offers. A commitment to WSDM today will allow users to migrate towards the converged specifications at their own pace through the use of migration tooling – making this an evolutionary change (i.e. a 'version' upgrade) rather than a disruptive one.

2. Overview of new Specifications

This portion of the document is intended to provide a high-level technical overview of the features in the WS-Resource Transfer specification.

As part of the reconciliation work several new specifications were developed, while some existing specifications were modified to more easily allow these new specifications to make use of them. This section will provide an overview of the new specifications as well a highlight the changes to the existing specifications that were made. It is assumed that the reader has at least a basic knowledge of the pre-reconciliation version of the specifications used for the reconciliation work. Figure 2.1 shows the relationship between the two stacks of specifications that warranted the need for this reconciliation work and their relationship to the reconciled stack.

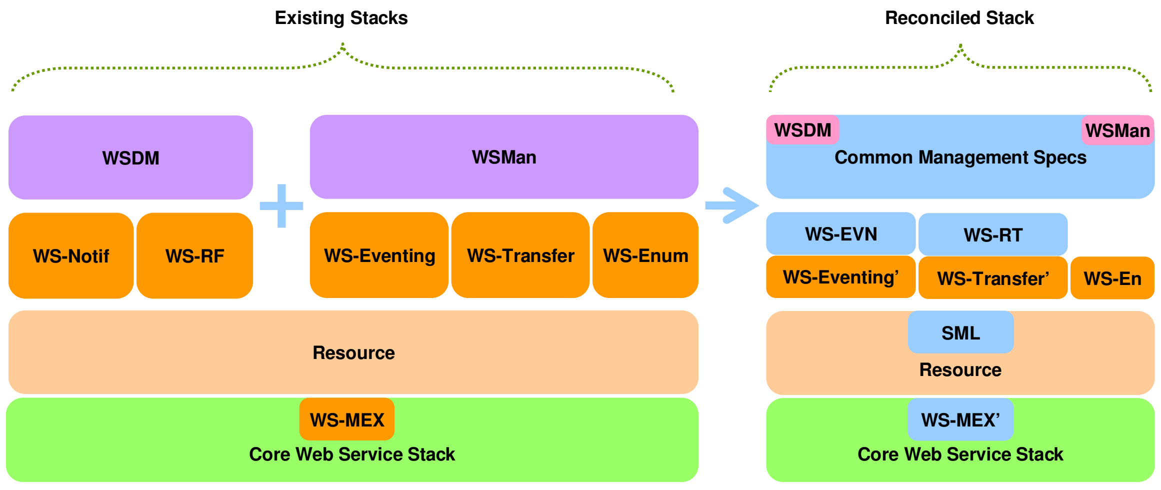

Figure 2.1

Note: WS-RF (ResourceFramework) is a collection of specifications. Most notably, WS-ResourceLife and WS-ResourceProperties (WS-RP) were examined during this effort. While WS-MEX (MetadataExchange) is not technically a management specific specification, it was modified as part of this effort as discussed in section 2.3. While it is a goal of the reconciliation effort to develop a set of Common Management specification that completely encompasses the functionality of both WSDM and WS-Man, it is possible for there to still remain some WSDM or WS-Man specific functionality that is considered out of scope – these are represented by the small pink boxes at the top corners of the Common Management Specs box. However, all parties involved are hoping that in the end there will be no need for them.

As previously mentioned, this section provides an overview of the specifications referenced by the reconciliation architecture – it will cover the high-level aspects of the specifications. For a more through description and discussion of the various features of the specifications please refer to the specifications themselves.

2.1. WS-Transfer Update

WS-Transfer, like WS-ResourceLifetime and WS-ResourceProperties, focuses on how to access and manipulate resources. WS-Transfer leverages the common CRUD (Create, Read (aka Get), Update (aka Put) and Delete) database operations as its main interface to the resource. Use of these four operations as a basis works well, however WS-Transfer in its pre-reconciliation state did not allow more advanced features, such as those defined in WS-ResourceTransfer. To allow for more flexibility, the authors of WS-Transfer updated the specification as specified in the following sections:

2.1.1. WS-Transfer Get

In the original WS-Transfer specification, the Get operation didn't allow for a SOAP Body. This meant that it could only be used for retrieval of the entire XML representation of a resource. While in many cases retrieving the entire resource may be adequate, there are other scenarios where this causes undue implementations burdens. For example, if the querying party is only interested in one small portion of a large resource's data, getting a large XML document places a heavy performance burden on the network.

To accommodate more specialized retrieval mechanisms, an extensible version of Get was introduced – one in which a SOAP Body was allowed to be included in the request message. This allows other specifications to define more optimized versions of Get that are better suited for their particular needs – as we will see in the WS-ResourceTransfer section. Note, that this change impacts both the Get's request and response messages since the response messages now allow for something other than the entire resource's representation to be returned.

2.1.2. WS-Transfer Put

WS-Transfer's Put operation needed to be modified in a similar way for similar reasons. Originally, it only allowed for the entire XML representation of the resource to be included in the Put's request message. To accommodate updating only a small portion of the resource, the Put operation was updated to allow for other specifications to define their own extensions (to be placed in the SOAP Body of the Put request message) again, allowing for more optimized functionality to be utilized.

The original version of Put only required the representation to be returned when the Put's request representation differed from the actual representation of the final state of the resource. However, since this could be quite hard for some implementations to determine, and the request may (through some additional specification) only contain a portion of the resource to update, this requirement was removed.

2.1.3. WS-Tansfer Delete

No substantive change was made to the Delete operation – however, it should be noted that use of the Delete operation completely deletes the resource and subsequent attempts to access it should result in the same response (fault) as if the resource has never existed at all. If only portions of the resource's representation are to be deleted then the Put operation should be used or the fragment support as defined in WS-ResourceTransfer's Put operation.

2.1.4. WS-Transfer Create

During the reconciliation work, it was realized that WS-Transfer's constraint on the Create operation that required the entire resource's representation to be included could be a challenge for some implementations. Following the pattern established by the changes to the Put operation, Create was modified to allow for more specialized SOAP Body content to be included in the request message. For example, through the use of WS-ResourceTransfer's Create operation, only a portion of the resource's representation may be passed in – assuming this is valid per the schema. The Create's response message still returns an EPR to the newly created resource – however, the initial representation is no longer returned. If this is still needed by the initiating endpoint the Get operation may be used to retrieve it.

2.2. WS-ResourceTransfer

With the modifications to WS-Transfer mentioned in the previous section, other specifications can now more easily leverage the extension points of WS-Transfer to define more specialized CRUD type of mechanisms. Through the examination of the various use-cases that guided the development of WS-ResourceProperties, WSDM and WS-Management, the reconciliation effort created one such specification: WS-ResourceTransfer.

It is important to note that since WS-ResourceTransfer extends WS-Transfer, all of the SOAP Action values (and WS-Addressing Action values) for each of the operations will be the same as specified in WS-Transfer. This means that the use of the wsrt:ResourceTransfer SOAP Header (discussed later) is critical, as without it an implementation may choose to ignore the extensions and the client would be unaware that its request was not processed per the WS-ResourceTransfer specification.

2.2.1. Fragment Support

One of the most significant features of WS-ResourceTransfer is the notion of referencing a portion of a resource's representation rather than the entire resource in its entirety. Clearly, from a performance point of view being able to send just one String, as an example, rather than the entire resource's XML representation (which could be quite large) could lead to a significant boost in performance.

To allow for this targeted referencing of a subset of a resource, WS-ResourceTransfer introduces the notion of a "fragment". Put simply, a fragment is nothing more than an expression that denotes which portion of the resource is of interest for the particular operation being invoked. To maximize flexibility, the specification does not mandate one particular form (or language) that must be used when specifying a fragment. Instead, it allows for implementations to specify a "Dialect" (or language) that will be used when formulating the expression. Depending on the application or implementation choices, different Dialects may be better suited then others.

The WS-ResourceTransfer specification specifies a set of common Fragment Dialects that it considers to be most common and, as such, encourages all implementations to support them:

- QName Dialect
  The QName dialect can be used to reference the top-level (immediate children of the root) elements of the resource. By specifying just the QName of the element of interest this fragment dialect can be used for very light-weight implementations that may not be able to support more flexible (or costly) dialects – such as XPath.
- XPath 1.0 Dialect
  This dialect allows for any XPath 1.0 expression to be used to reference any portion of the resource.
- XPath Level 1 Dialect
  This dialect subsets the XPath 1.0 dialect to just some of the most common, and basic, features – such as referencing single nodes or limiting the full set of library functions to just the text() one. This dialect was developed for resource constrained devices that needed some basic level of XPath support but would find it too costly to support XPath 1.0 itself.

In the WS-ResourceProperties specification there is a QueryExpressionDialect property that specifies which Dialects may be used when issuing queries against the resource. WS-ResourceTransfer will provide the same information in its metadata – see section 2.2.6 for more information.

2.2.2. WS-ResourceTransfer Get

Per the WS-Transfer specification, the Get operation can be used to retrieve the XML representation of a resource. The WS-ResourceTransfer specification makes use of the extensibility points defined in WS-Transfer by defining a new XML element to be used in the SOAP Body of a Get request message to allow for retrieval of just a fragment of the resource's representation. The syntax of this new element is:

```
(01) <wsrt:Get Dialect="xs:anyURI" ...>
(02)   <wsrt:Expression ...> xs:any </wsrt:Expression> *
(03) </wsrt:Get>
```

As described in section 2.2.1 (Fragment Support) the Dialect attribute specifies the language that will be used in the Expression element. It is worth pointing out that only one type of Dialect is permitted per Get request (and as we'll see on the Put as well). While on the surface it would seem more flexible to allow each Expression to have it own Dialect attribute – it was decided that it was better to allow an implementation of Get to be able to determine quickly whether or not it supported the requested Dialect – thus allowing it to reject any incoming request message immediately without incurring the overhead of processing some of the Expressions just to find out that it might need to return a Fault if is encounters a Dialect that it does not support.

The Expression element is a wrapper for the Dialect specific query string that will specify the portion of the XML representation of the resource that should be returned in the response.

Since there may be multiple Expression elements within the Get element, it is possible to retrieve several different sections of the resource at the same time. However, as noted above, all of the Expression elements must use the same Dialect language. If an implementation find it too costly to support more than a certain number of Expression elements within on Get request, then it may generate a Fault indicating how many it does support.

When a Get request message is processed there is no way for a client to know for sure whether or not the server will actually support the WS-ResourceTransfer extension. If it doesn't support it then it may simply choose to ignore the Body of the Get[5]. In order to ensure that the resource implementation understands and will adhere to the semantics defined in the WS-ResourceTransfer specification a new SOAP Header must be included in the request message:

```
<wsrt:ResourceTransfer soap:mustUnderstand="true"?>
```

When this Header is included, and marked with the mustUnderstand attribute set to 'true' then the client will be assured that if the server doesn't know about WS-ResourceTransfer then it will generate a MustUnderstand Fault when it encounters this Header and no further processing will take place.

If the Get succeeds then the response message will look like:

```
<wsrt:GetResponse>
  <wsrt:Result...> xs:any </wsrt:Result> +
</wsrt:GetResponse>
```

There will be one Result element in the response for each, and every, Expression element that was present in the request – even if the response for that particular Expression was empty. Also note that the order of the Result elements must match the order of the Expression elements.

While the base WS-Transfer Get is still supported for the retrieval of the entire resource, it is worth pointing out that the same functionality can be achieved two ways, first by specifying an Expression that refers to the root element of the resource, and second by not including any Expression elements at all – thus asking for the entire resource's representation.

The wsrt:ResourceTransfer Header will also be included on the GetResponse message so that the receiver knows how to properly interpret the results.

[5] While the SOAP specification itself makes it very clear that a SOAP processor must understand the Body element (or fault), the updated WS-Transfer specification seems to imply that an implementation may choose to ignore it – thus it mentions the need for a SOAP mustUnderstand Header to ensure the Body element is processed. Technically, this is redundant but since the initial version of WS-Transfer didn't allow for a SOAP Body, existing implementation may not even look at the Body element at all. By using a mustUnderstand SOAP Header we're guaranteed all implementations will examine and understand the SOAP Body.

When comparing the functionality of this extension with the current retrieval operations that are specified in WS-ResourceProoperties (e.g. GetResourceProperty), the WS-ResourceTransfer Get, plus the use of the Fragment support, provides equivalent functionality.

2.2.3. WS-ResourceTransfer Put

As with the WS-ResourceTransfer Get operation, the WS-ResourceTransfer Put operation adds Fragment support to allow for more fine-grained data access and manipulation. In the Put case, the syntax of the new SOAP Body looks like:

```
<wsrt:Put Dialect="xs:anyURI"?>
  <wsrt:Fragment Mode="Modify|Insert|Remove">
    <wsrt:Expression> xs:any </wsrt:Expression> ?
    <wsrt:Value ...> xs:any </wsrt:Value> ?
  </wsrt:Fragment> +
</wsrt:Put>
```

Like we saw in the Get element, the Put element has a Dialect attribute that specifies the language that will be used in the Expression elements.

Since there may be multiple portions (Fragments) of the resource being updated, the Put element allows for multiple Fragment elements to be specified. Each one contains an Expression and a Value element. The Expression element is used to specify which portion of the XML representation of the resource is being targeted by this Fragment. And, the Value element specifies the new XML value to be inserted at that location.

The Fragment element has a Mode attribute that allows for a wider level of control than simply insertion of new data. There are four possible values:

- Modify
  This mode means that the fragment pointed to by the Expression must be replaced by the XML in the Value element. If the Expression element is not present then the Mode attribute must be "Modify" and the entire resource's representation will be replaced.
- Insert
  This mode means that the new value must be added to the resource at the fragment pointed to by the Expression. This is typically used to add more value to array-type of structures. If the Expression references a repeated element then the fragment will be added to the end, if it references an existing item of a repeated element then it will be added before it.
- Remove
  This mode is used as a fragment-level delete. The portion of the resource that is referenced by the Expression element will be removed from the XML representation of the resource. Note, in this case the Value element must not be present.

The WS-ResourceTransfer Put operation requires the use of the wsrt:ResourceTransfer SOAP Header as described in the WS-ResourceTansfer Get operation. This ensures compliance with this specification.

If the resource successfully processes the Put operation then the response message will be as it is described in the WS-Transfer specification – meaning the SOAP Body will either contain an updated EPR to the resource or nothing at all. And, like we say in the GetResponse message, it must include the wsrt:ResourceTransfer SOAP Header.

The functionality defined in WS-ResourceTransfer Put provides equivalent mechanisms that are defined in WS-ResourceProperties Set/Insert/Update/Delete-ResourceProperties operations.

2.2.4. WS-ResourceTransfer Create

Following the pattern we've seen in the Get and Put operations, the WS-ResourceTransfer Create operation allows for multiple Fragment elements to be included in the request message. This allows for the client to specify only portion of the resource in the create message. This is useful in cases where the remaining portion of the resource has default values that do not need to be specified by the initiating endpoint. The syntax for the WS-ResourceTransfer Create is:

```
<wsrt:Create Dialect="xs:anyURI">
  <mex:Metadata>resource metadata</mex:Metadata> ?
  <wsrt:Fragment>
    <wsrt:Expression> xs:any </wsrt:Expression> ?
    <wsrt:Value ...> xs:any </wsrt:Value>
  </wsrt:Fragment> *
</wsrt:Create>
```

The Dialect attribute, like the other operations, specifies the language of the Expression element. The Fragment element, like in the Put, contains an Expression and Value, allowing the client to specify where to place the specified data within the resource's representation. This operation also requires the use of the wsrt:ResourceTransfer SOAP Header, as described in the WS-ResourceTransfer Get operation, to ensure compliance with this specification. For ease of understanding, the wsrt:Create operation can be thought of as creating a new resource followed by a series of wsrt:Put Fragments that need to be applied to this new resource.

One very important difference between the Create and the Put, however, is the Metadata element. This element allows for any number of metadata representations to be passed in. This can be crucial in cases where the metadata for the resource must be created and associated with the resource at the same time that the resource itself comes into existence. For example, the metadata may contain security policies that would need to be in-place immediately, preventing unauthorized access to the resource. Note, this Metadata element can also contain the WS-ResourceTransfer-specific metadata – see section 2.2.6. This Metadata element must conform to the WS-MetadataExchange Metadata element.

While the base WS-Transfer Create operation can be used to pass in the entire representation of the resource, it is possible to use the WS-ResourceTransfer Create as well by simply specifying an Expression that refers to the root of the resource and placing the entire resource's representation in the Value element, or like we saw in Put, not specifying an Expression element at all implies the entire representation.

WS-ResourceTransfer does not change, or extend, the WS-Transfer Create response message – the response will contain an EPR to the newly create resource. While the specification does show a wsrt:ResourceTransfer header on the response message, since the rest of it looks exactly like a normal WS-Transfer CreateResponse message it really isn't as necessary as it is on the other operations.

2.2.5. WS-ResourceTransfer Delete

The WS-ResourceTransfer specification does not change, or extend, the WS-Transfer delete operation. It provides equivalent functionality to the WS-ResourceLifetime Destory operation.

2.2.6. Resource Metadata

In the WS-ResourceFramework specifications there was a requirement for the resources to contain certain properties based on the types of operations it chose to support – for example, the TerminationTime property would be required to be on a resource that wanted to support the SetTerminationTime operation. In the reconciled specifications any WS-ResourceTransfer 'required' data was moved into the resource's metadata – thus allowing the resources to have full control over its own data and representation. WS-ResourceTransfer defines a new URI that can be used when using WS-MetadataExchange's GetMetadata operation:

```
http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer
```

Since the WS-MetadataExchange specification allows for the result of a GetMetadata operation to return the metadata three different ways (by value, by EPR reference or by URI reference), the WS-ResourceTransfer specification recommends that when the metadata is mutable, the GetMetadata operation should always return an EPR to the metadata resource so that WS-Transfer (and WS-ResourceTransfer) operations can be used to modify the metadata. There is more discussion of this in section 2.3.

The metadata referenced by this URI contains all of the WS-ResourceTransfer specific data that may be needed by a resource. The form of the metadata will look like:

```
<wsrt:Metadata>
  <wsrt:LifeTime> lifetime metadata </wsrt:Lifetime> ?
  <wsrt:SupportedDialect>
    dialect metadata
  </wsrt:SupportedDialect> *
</wsrt:Metadata>
```

The Lifetime element contains similar information to what was defined by the WS-ResourceLifetime specification, it is information related to the lifecycle of the resource. The Lifetime element's syntax looks like:

```
<wsrt:Lifetime>
```

```
  ( <wsrt:TerminateAt>
      <wsrt:TerminationTime> xs:dateTime </wsrt:TerminationTime>
      <wsrt:CurrentTime> xs:dateTime </wsrt:CurrentTime>
    </wsrt:TerminateAt> |
    <wsrt:TerminateAfter> xs:duration </wsrt:TerminateAfter> |
    <wsrt:TerminateAfterIdle>
       xs:duration
    </wsrt:TerminateAfterIdle>] )
</wsrt:Lifetime>
```

The Terminate timings relate to when the resource will be automatically destroyed. The TerminateAt value is the time, as determined by the resource's clock, at which the resource will be destroyed. Notice, that the resource's current time is also part of this metadata – thus allowing the requester of this information the ability to know the value of the resource's clock without having to worry about clock synchronization issues. The TerminateAfter value is the amount of time, as related to the resource's current time, after which the resource will be destroyed. The TerminateAfterIdle value is the amount of time after the receipt of any message, by the resource, that the resource will be destroyed.

The wsrt:SupportedDialect element can be used to express how a dialect, that can be used in the Expression element in the Get and Put operation, can be used. The SupportedDialect element looks like:

```
(01) <wsrt:SupportedDialect DialectName="xs:anyURI">
(02)   <wsrt:SupportedOperation OperationName="xs:anyURI">
(03)     <wsrt:SupportedPutMode> ModeType </wsrt:SupportedPutMode> *
(04)     <wsrt:MultipartLimit>
(05)       xs:positiveInteger
(06)     </wsrt:MultipartLimit> ?
(07)   </wsrt:SupportedOperation> *
(08) </wsrt:SupportedDialect>
```

As seen on line (01) the SupportedDialect element has a DialectName attribute that contains URI of the dialect that is supported. For each dialect there will be an array of SupportedOperation elements, line (02), for each WS-Transfer operation that supports this dialect in an Expression element. Line (03), an array of SupportedPutMode elements, will specify which Modes may be used when using the Put operation. The MultipartLimit element specifies the maximum number of Fragment elements that may be specified for the given operation. If this element is not specified then there is no limit.

It is worth noting that this metadata may not be static and, if the resource chooses to allow it, may be updated through the use of a Put operation. See the WS-MetaExchange section for more information about how to get access to this data.

2.3. WS-MetadataExchange

Those who are familiar with the work that has been done in the past with WS-MetadataExchange will remember that there were three key components to the specification:

1. Definition of the mex:Metadata element
   This defined a standard format for wrappering groups of metadata into one XML document.
2. Definition of the mex:GetMetadata operation
   This defined as standard mechanism by which a requester could ask for an endpoint's metadata. The metadata could be returned in one of three ways:
   a. The metadata could be returned inside the response message.
   b. A URL reference to the metadata could be returned – and an HTTP GET could be issued against this URL to retrieve the actual metadata.
   c. An EPR reference to the metadata could be returned – and a mex:Get operation could be used to retrieve the actual metadata.
3. Definition of the mex:Get operation
   As mentioned in 2.c, this operation defined a standard operation for retrieving the metadata when that metadata happened to also be the XML representation stored in a resource (also known as a metadata resource). In other words, when an EPR referred to a resource who's data was metadata about another resource, then mex:Get could be used to retrieve it.

As part of the reconciliation work the overlap between the WS-MetadataExchange Get operation and the WS-Transfer Get operation was removed. In both instances the operations were used to retrieve the XML representation of a resource – the only thing special about the WS-MetadataExchange case is that the resource just happened to be a "metadata" resource. However, there is no conceptual difference between a "metadata" resource and any other kind of resource. As such, there is really no need to have two different operations performing the same action.

It is worth pointing out that this convergence is only possible because of the agreement to use WS-Transfer as the basis for resource access in the future. For users of the current set of specifications, the EPR returned by the WS-MetadataExchange GetMetadata operation can also be used with a WS-ResourceFramework's GetResourcePropertyDocument operation – as noted in the WS-MetadataExchange specification.

While it may seem a bit odd that an infrastructural-level specification, like WS-MetadataExchange, that can be used to retrieve metadata about any Web service (not just management resource) now has a dependency on a higher-level specification – the duplication of function was viewed as something that should be avoided. This means that in Figure 2.1 where it shows WS-MetadataExchange being part of (and only requiring) the "core" Web service specifications – there is an implicit assumption that the WS-Transfer Get operation will be supported by any endpoint that wishes to support retrieval of metadata through an EPR reference. This would move just this one operation (WS-

Transfer Get) into this "core" set of features that are expected to be supported by most SOAP endpoints.

An advantage of relating WS-MetadataExchange with the WS-Transfer specification is its implicit relationship with WS_ResourceTransfer. In the old version of WS-MetadataExchange, much like the old version of WS-Transfer, a client was force to retrieve the entire document representing the metadata. However, now a client can use the WS-ResourceTransfer specification to do a partial retrieval if only a portion of the data is needed – thus allowing for more optimized message exchanged.

There is one slight problem with this though – WS-MetadataExchange does not indicate when the result of a GetMetadata operation should return the data itself, a URL to the data, or when it should return an EPR to the data. The Web service endpoint has no idea what the client's intentions are – as such the service should return as many different forms of the metadata (value, URL and EPR) as possible, thus giving the client the option to choose how to use this information in whatever method is most appropriate for its needs.

2.4. Service Modeling Language

The Service Modeling Language, SML, specification is a set of profiles to be used to define the shape of a set of resource's information and relationships. It defines a profile of XML Schema 1.0 for content, Schematron for constraints, and Xpointer for relationships and references. A set of related SML documents are an SML model. Mapping to XML schema from existing UML type models, like CIM, has been inconsistent because the resulting XML instance documents are often not as simple or clean as they might have been if the resource had been described using XML from the beginning. Various UML profiles and models add information to the XML document to enable round trip transformation. Using the SML profiles we can model IT resources and topologies natively in XML.

Basically, we think of a manageable resource as having a 'resource properties document', which implies an XML schema document describing the shape and constraints of the resource as well as an XML instance document containing its current values. In WS-ResourceTransfer these are called the resource document. We envision that for management, and possibly other domains, these documents describing resources and models of resources will be SML compliant. WS-CIM also creates XML schema from MOF, but it contains a lot of information for round-tripping the mapping. Over time, the WS-CIM Mapping should evolve to be SML compliant.

2.5. Common Model Library

While agreeing on the profiles in SML is important for interoperability, the true interoperability between resources and managers will be achieved when we have interoperable semantics as well. The Common Model Library (CML) will provide a set of modeling building blocks as well as very basic resource types. These common schemas will be reused and composed together to create models for complex and/or vendor specific resources.

The capabilities currently in WSDM were created to provide common semantics when representing management information from different models. We envision that the scenarios addressed by the WSDM capabilities, including Resource ID, Corralatable Names, State, Status, Configuration, etc. will be addressed by the CML model building blocks.

3. Migration and Compatibility

This portion of the document will discuss the various technical details of migrating from an existing WSDM implementation to the new reconciliation specifications.

3.1. Overview

From a functional point of view, the reconciliation specifications are a superset of their OASIS/WSDM counterparts. Therefore, moving from the WSDM specifications to reconciliation specifications can be viewed as both an upgrade and a migration; it's an upgrade because the reconciliation specifications offer additional set of functions, and it's a migration because existent engines and tooling will create produce versions to cover the reconciliation functionality. This provides a clean strategy for compatibility and co-existence for the developer.

What is implied by a compatible version or release update to a WS-Resource is that existing client applications using or managing the WS-Resource should continue to work without disruption. This requires that the changes introduced to the interfaces (WSDL 1.1 portTypes) implemented by the resource be additive. New resource properties, metadata, and operations can be added, however existing properties metadata and operations must not change syntactically or semantically.

As described in earlier sections, the WS-RT portTypes are not syntactically the same as those defined by WS-RF. Existing WS-RF based WS-Resources that wish to provide backward compatibility to WS-RF client applications should implement both WS-RF and WS-RT portTypes for some period of time. As WS-RT becomes ubiquitous, the WS-Resource would then be able to deprecate the WS-RF portTypes and no longer provide them to client applications (a.k.a. resource managers).

What makes this approach somewhat painless to the WS-Resource developer is that while the WS-RT portTypes are syntactically different, they are semantically very similar. This allows the implementation of the WS-Resource to provide both a WS-RF and WS-RT projections over the same underlying resource state. The burden of providing co-existence of what might be considered redundant interfaces should be relatively trivial. In fact, many runtimes, such as Apache Muse, will provide help to the developer by providing implementations of WS-Resources with this approach.

To further elaborate, a simple example should help. In this example, let's assume some resource "Server" has implemented a WS-RF based WSDM 1.1 manageability interface and is being used by existing management applications and tools. In this example, the Server portType aggregates the properties and operations for the applicable WS-RF portTypes and WSDM Capabilities as shown below:

- wsrp:QueryResourcePropertyDialect
- muws:ResourceId
- muws:ManageabilityCharacteristics
- server:ServerState

- wsrl:Destory()
- wsrp:GetResourceProperty()
- wsrp:QueryResourceProperties()

As management applications and tools become available that support the WS-RT only, it will become important for the Server resource to implement the WS-RT portTypes. It is also assumed that the Server resource will need to provide backward compatibility for the existing management applications and tools that are written to the WS-RF portTypes for some period of time. For this period of time, the Server portType would aggregate the applicable WS-RF and WS-RT portTypes. The resulting compatible portType would be as follows:

- wsrp:QueryResourcePropertyDialect
- muws:ResourceId
- muws:ManageabilityCharacteristics
- server:ServerState
- wsrl:Destory()
- **ws-t:Delete()**
- wsrp:GetResourceProperty()
- **wsrt:Get()**
- wsrp:QueryResourceProperties()

The Server resource would also now need to provide access to a metadata resource that minimally would included WS-RT dialects property (wsrt:Dialects). Semantically this provides the same information to the client as the QueryResourceProperyDialect property defined in WS-RF.

When the backward compatibility was no longer required, the Server resource would no longer implement the WS-RF portTypes. In this case, the Server portType would now only include those operations and properties defined by WS-RT as follows:

- muws:ResourceId
- muws:ManageabilityCharacteristics
- server:ServerState
- ws-t:Delete()
- wsrt:Get()

The Server resource would also continue to provide wsrt:Dialects metadata, however would no longer include the QueryResourcePropertyDialect property.

What makes such a strategy possible is the semantic equivalence of WS-RF and WS-RT. In this simple example, the implementation of the WS-RF resource property operations (GetResourcePropery and QueryResourcePropery) and the WS-RT operations (Get) can be delegated to the same underlying resource state in a fairly trivial fashion.

3.2. Migration Strategies

The suggested strategy for the migration of WS-Resources from WS-RF to WS-RT is to provide client applications compatibility and co-existence, with the eventual deprecation of the WS-RF portTypes. This should be viewed like any other client compatible version or release update to a WS-Resource.

3.2.1. When is migration necessary?

Currently, migration is only necessary for users who are interested in the unique functions of the reconciliation specifications, or who have interoperability scenarios with products, clients/managers and servers/resources that may support the new specifications. Users of WSDM who do not have these requirements do not need to immediately migrate; however, they should closely monitor their interoperability scenarios to ensure a viable migration path when, over time, the WS-RT APIs become more prevalent and the WS-RP APIs are deprecated.

3.2.2. Migration Strategy to WS-RT for WS-RP Users

Most WS-RT operations map directly to WS-RP operations. Engines that support both specifications can simply delegate mappable WS-RT requests to WS-RP services. Therefore, compatibility and coexistence is achieved on engines that support this kind of delegation model because WS-RP users will not need to change their WS-RP web services implementations to leverage WS-ResourceTransfer. Apache Muse is one such engine that implements this delegation model.

A key advantage of the delegation model is it enables engines to provide multilingual (WS-RT/WS-RP) client side APIs that talk to either type of resources on the server side without having to worry about the actual implementation type of the Web service. Internally, the engine will forward WS-RP requests to existent WS-RP resources and WS-RT requests to a delegation layer that will map WS-RT requests to WS-RP operation(s) that can be invoked on existent WS-RP resources. For example, an incoming WS-RT Put request with property dialect will be mapped to a WS-RP SetMultipleResourceProperties operation on the WS-RP resource.

Some WS-RT operations do not have direct equivalents in the WS-RP specification. One such operation is a WS-RT Put that contains XPath fragments. To handle such an operation, the delegation layer can execute the XPath fragments on a DOM representation of the WS-RP document and then invoke a WS-RP PutResourcePropertyDocument operation on the resource. There are no parts of the WS-RT spec that are so unique that they cannot fit side-by-side with the WS-RP specification.

Apache Muse 2.X plan

Apache Muse 2.X has been designed to accommodate WS-RT migration and will implement the delegation strategy described above.

The figure below outlines the Muse 2.X plan.

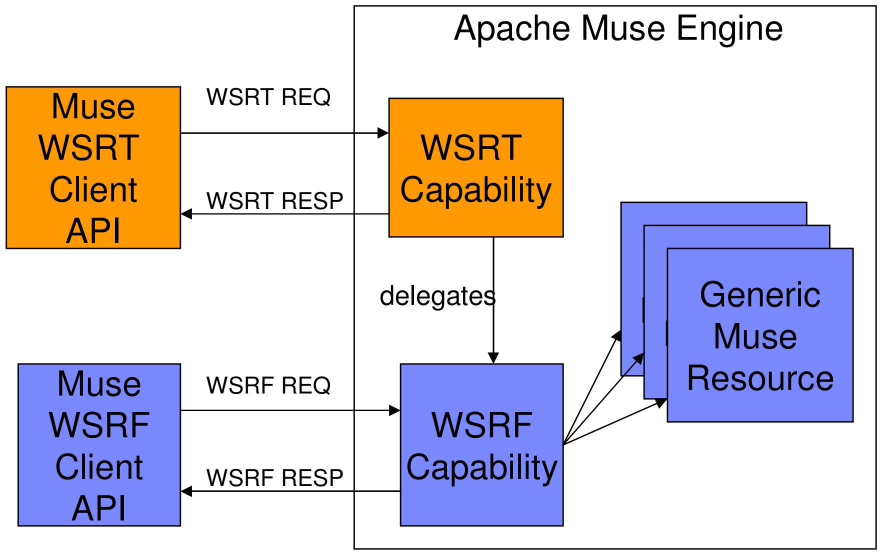

Figure 3.1

Server Side notes:

- Resources hosted on Muse 2.X do not have to be bound to WS-RF or WS-RT specifications; this is represented by the Generic Muse Resource above.
- WS-RT and WS-RF are implemented as "capabilities" on the server-side that can be plugged into a resource via deployment descriptors; therefore, no code changes are required to expose the original generic resource via either specification.
- The notion of generic resource types in the engine; i.e. resources that are bound to neither specification, makes binding a deployment time decision instead of a development time decision.
- The runtime extends the notion of resource properties to types that may not be declared in the WS-RP Web service's WSDL. WS-RP provides a resource property document hint via a wsrf-rp:ResourceProperties attribute in the WS-RP port type whereas no such concept currently exists in WS-RT. Therefore the runtime doesn't assume that a resource schema will always be available in the resource's WSDL

Client Side notes:

- Client-side APIs enable Muse clients to talk to the generic Muse resource in either language.
- The client-side APIs will expose methods to:
    - Send WS-RT or WS-T messages to WSDM/OASIS resources deployed on Muse: This will allow WS-RT or WS-T clients to talk to WSDM/OASIS resources.
    - Hide all XML processing associated with sending WS-RT or WS-T requests to a Muse resource: Factory methods will be provided for creating WS-RT specific objects such as fragments or metadata.
    - By offering a WSDL document that includes both the WS-RP and RT interfaces, as described in the strategies above, users of WSDM will

continue to be able to use the WS-RP client side APIs that is provided by Muse today.

<u>Rationale for delegation strategy</u>

For Muse 2.X, a delegation strategy was chosen over a grounds-up WS-RT implementation to provide an easy migration step for WS-RF users and to allow for the incremental development of a full fledged WS-RT implementation over time.

<u>Limitations of the delegation strategy</u>

The delegation layer has to be coded into the engine; if the WS-RF resource is not deployed on Muse, or is deployed on an engine that does not support the delegation model, an on-the-wire transformation of messages (XSLT stylesheets) can be used to support both dialects. The XSLT transform can occur on the client side before network dispatch or on the server side before Web services dispatch.

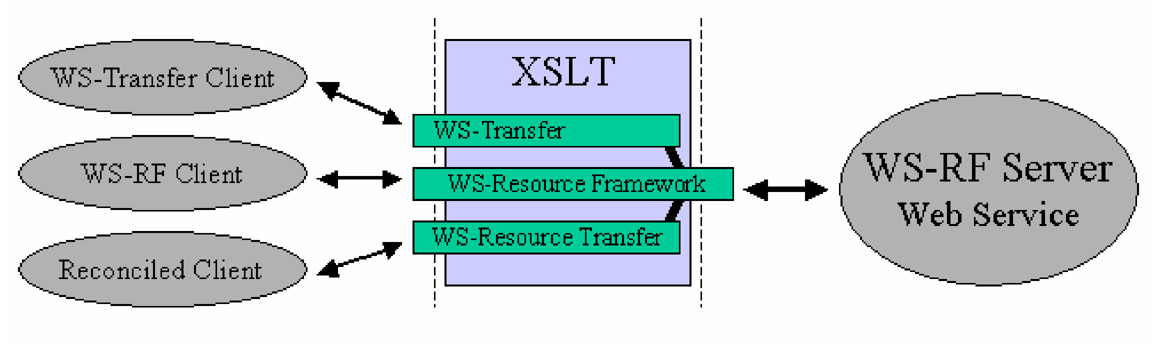

Figure 3.2

XSLT stylesheets perform XML to XML transformations on SOAP request and response messages from one dialect to another. The above figure shows how XSLT can be used by clients of either dialect to talk to any WS-RP Web service hosted on any WS-RF engine.

The following interoperability matrix outlines what scenarios are supported by planned releases of Muse and where an XSLT solution should be used

Client\Resource	**Muse 2.X Resource**	**Immutable Muse* or Non-Muse RT Resources****	**Immutable Muse* or Non-Muse RF Resource****
Muse 2.X RF Client	Muse 2.X: Yes Muse 3.X: Yes	Muse 2.X: XSLT Muse 3.X: Yes	Muse 2.X: Yes Muse 3.X: Yes
Muse 2.X RT Client	Muse 2.X: Yes Muse 3.X: Yes	Muse 2.X: Yes Muse 3.X: Yes	Muse 2.X: XSLT Muse 3.X: Yes

Non-Muse RT Client	Muse 2.X: Yes Muse 3.X: Yes	Implicit	XSLT Always *As client and server are immutable*
Non-Muse RF Client	Muse 2.X: Yes Muse 3.X: Yes	XSLT Always *As client and server are immutable*	Implicit

* A resource that can not be transferred or deployed to another server.
** Assuming Non-Muse resources are deployed on engines that don't support the delegation strategy described above.

The matrix shows that two scenarios will always mandate the use of XSLT transformations in situations where delegation strategies can not be applied to servers in which resources are hosted. While, XSLT can be useful in such scenarios, it should be kept in mind it is a pure XML-XML transformation of dialects and therefore only operations that directly map on to one another in both specifications can be supported by XSLT.

3.3. Apache Muse

The delegation functionality required to handle WS-RT will be delivered in Muse 2.X as a technical preview. Samples and articles for demonstration will also be made available either in Muse directly or on developerWorks or alphaWorks. The tech preview will be introduced into Muse releases in the second half of 2006, with the first release in September.

Because it is a tech preview, formal support for the code support for WS-RT will be worked into the 2.X plan. .As the reconciliation specifications go through the standardization process, Muse will continue to add implementation support for them. The current plan is for Muse 3.X to begin deprecating the old implementation and become primarily a WS-RT-based engine. However, the goal of trying to maintain a consistent programming model between 2.X and 3.X will remain the same, in order to facilitate backward compatibility for 2.X resources in the Muse 3.0 code stream.

Muse 3.X will display warning messages during compilation when WS-RF client APIs and deployment descriptors are used.

3.4. Migration Issues

Three parties will be impacted by migration:

- WS-RP Clients: existent clients that need to talk to resources that only support WS-RT clients.
- WS-RP Resources: existent resources that need to serve WS-RT clients.
- WS-RP engine implementers.

3.4.1. Technical Perspective

WS-RP Client migration

1. Mutable Clients: these are WS-RP clients to which code modifications can be made to support new WS-RT dialects. Such clients are encouraged to migrate to use Muse 2.X WS-RT client side APIs to invoke WS-RT resources hosted on any engine. Muse WS-RT client APIs will work just like WS-RP client APIs, i.e. they will provide a programmatic model to invoke WS-RT resources, except in WS-RT dialect instead of WS-RP dialects.

2. Immutable Clients: these are WS-RP clients that can not be modified to use new Muse WS-RT APIs. Such clients can employ XSLT transformations to convert outgoing WS-RF SOAP messages to WS-RT messages, and vice-versa for WS-RT responses, on the wire. The method by which XSLT is applied to messages will be scenario dependent; for example, XSLT transformations may be configured on an enterprise service bus on which the client resides.

WS-RP Resource migration

1. Mutable Resources: these are WS-RP resources to which code changes can be made for adding WS-RT support. Such WS-RP resources can be deployed on an engine that supports the delegation model described above. On Apache Muse 2.X, a resource can be exposed to WS-RT clients via a mere change to a deployment descriptor; therefore the strategy here would be to:
   a. First port existent WS-RP resources to Apache Muse 2.X
   b. Then ensure that ported web service includes the WS-RT "capability" in its Muse deployment descriptor file.

2. Immutable Resources: these are WS-RP resources that can not be modified to support WS-RT. Such resources can employ XSLT transformations to convert incoming WS-RT SOAP messages to WS-RP messages, and vice-versa for WS-RP responses, on the wire. The method by which XSLT is applied to messages will be scenario dependent; for example, XSLT transformations may be configured on an enterprise service bus on which the resource resides.

WS-RP engine migration

Our recommendation to WS-RP engine implementers is to implement the delegation model described above as it has the following advantages:

- Allows for co-existence of WS-RP and WS-RT resources.
- Existent users of the engine will only have to upgrade to new release of engine to use WS-RT; users' WS-RP Web services will not have to be re-coded to support WS-RT.
- Delegation layer is easier to write as opposed to a ground up WS-RT engine; this will be a safe guard for changes that might be made to WS-RT specifications as they go through the standardization process.
- Migration can take place at the application writer's pace – allowing for a more controlled and smooth transition.

Then as the specifications are standardized, WS-RP providers can build a full fledged WS-RT engine, while deprecating the old functionality and providing backward compatibility for previous releases.

3.4.2. Guidance for products

Products can fall into one or more of the categories that have already been described: a) WS-RP engines, 2) WS-RP clients, 3) WS-RP resources. They should use the strategies described above to create a migration roadmap for themselves and their clients.

3.5. Tooling

The Eclipse TPTP Monitoring project contains tools to help instrument, test, debug, and deploy resources enabled for WSDM. The tooling works by allowing the developer to focus on describing the manageability interfaces that will the underlying resource exposes. The descriptions of these resources are then used to generate the code that conforms to the Apache Muse programming model.

Because the manageability interfaces are described externally, the tooling easily supports migration from one release to another. Therefore, the tooling naturally supports the co-existence and migration of WS-RP and WS-RT.

Appendix A Contributors

This document was produced by the following individuals:
Jaipaul Antony, Mike Baskey, John Chang, Jim Crosskey, Doug Davis, Mohammad Fakhar, Ron Goering, Dan Jemiolo, Heather Kreger, Dave Russell, Mark Weitzel, Mike William, Leigh Williamson, Mary Yost

Appendix B Resources

OASIS(Organization for the Advancement of Structured Information Standards) is a not-for-profit, international consortium that drives the development, convergence, and adoption of e-business standards. The consortium produces more Web services standards than any other organization along with standards for security, e-business, and standardization efforts in the public sector and for application-specific markets. Founded in 1993, OASIS has more than 5,000 participants representing over 600 organizations and individual members in 100 countries.
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm

The Eclipse Test & Performance Tools Platform (TPTP) Project is an open source Top Level Project of the Eclipse Foundation. Tools are available to expedite the implementation of WSDM will be available in TPTP 4.3 in November 2006. It uses the Apache Muse runtime.
http://www.eclipse.org/tptp/

The Apache Muse Project is a Java implementation of web services standards published by OASIS; specifically, it includes an implementation of WS-ResourceFramework (WS-RF), WS-BaseNotification (WSN), and the Management Using Web Services (MUWS) portion of WS-DistributedManagement (WSDM). It is a framework upon which users can build web service interfaces for manageable resources without having to implement all of the basic "plumbing" described by the aforementioned specifications.
http://ws.apache.org/muse/

IBM Build-to-Manage Toolkits: Toolkits to support important system management standards including WSDM. Available on DeveloperWorks at
http://www.ibm.com/developerworks/eclipse/btm/

Articles and whitepapers:
Kreager, Heather "A Little Wisdom about WSDM"
http://www.ibm.com/developerworks/webservices/library/ws-wisdom/

Brown, Martin C. "Understanding Web Services Distributed Management"
http://www.ibm.com/developerworks/edu/ws-dw-ws-wsdm.html