July 22-26, 2013

City Center Marriott
Portland, OR

RESTful Services for CIM (CIM-RS)

Andreas Maier (IBM)
STSM, Systems Management Architecture & Design
maiera@de.ibm.com

Disclaimer

- The information in this presentation represents a snapshot of work in progress within the DMTF.
- This information is subject to change. The Standard Specifications remain the normative reference for all information.
- For additional information, see the Distributed Management Task Force (DMTF) Web site.

Why a REST based protocol for CIM ?

- REST enjoys increasing support in the industry
- REST is increasingly used for systems management
- Client application code can talk REST directly
- REST is perceived to be simple
- We can add it as a new protocol to existing CIMOMs without impacting providers

REST in 5 minutes

- REST = "REpresentational State Transfer"
  - Introduced in 2000 in the dissertation of Roy Fielding
- Mostly uses HTTP operations (e.g. GET, PUT, POST, DELETE)
  - HTTP semantics provides for a number of benefits, e.g. transparent usage of HTTP caching
- REST operations deal with "resources"
  - A resource is addressed using a URI
  - Representation of resources can be determined by HTTP content negotiation (e.g. JSON, XML, …)
- Stateless protocol – stateful resources
  - Server does not need to maintain state for client context or session between operations
  - Resources are still "stateful", i.e. make their state accessible
- Differences between RESTful services and "stateful" Web Services:
  - Addressing of resources (REST URI vs. WS-Addressing)
  - Abstraction from transport protocol (REST often tied to HTTP, Web Services try to abstract from t.p.)
  - Usage of operation semantics (REST often tied to HTTP, Web Services have the WS-* stack of specs)
  - Literature: "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision" by Pautasso, Zimmermann, Leymann (http://www.jopera.org/files/www2008-restws-pautasso-zimmermann-leymann.pdf)

CIM-RS Overview

- Supports RESTful access to CIM-modeled resources
  - Any CIM model can be used that conforms to the CIM metamodel
- True REST support
  - Clients can use any REST framework, no need for WBEM client libraries
  - CIM client libraries can also support CIM-RS for compatibility
- Open set of payload encodings
  - HTTP content negotiation used for selecting the payload encoding
  - Supports a JSON encoding
- Separation of model and protocol
  - CIM model (schema and profiles): Defined in a protocol-neutral way
  - CIM-RS protocol: Defined in a model-neutral way

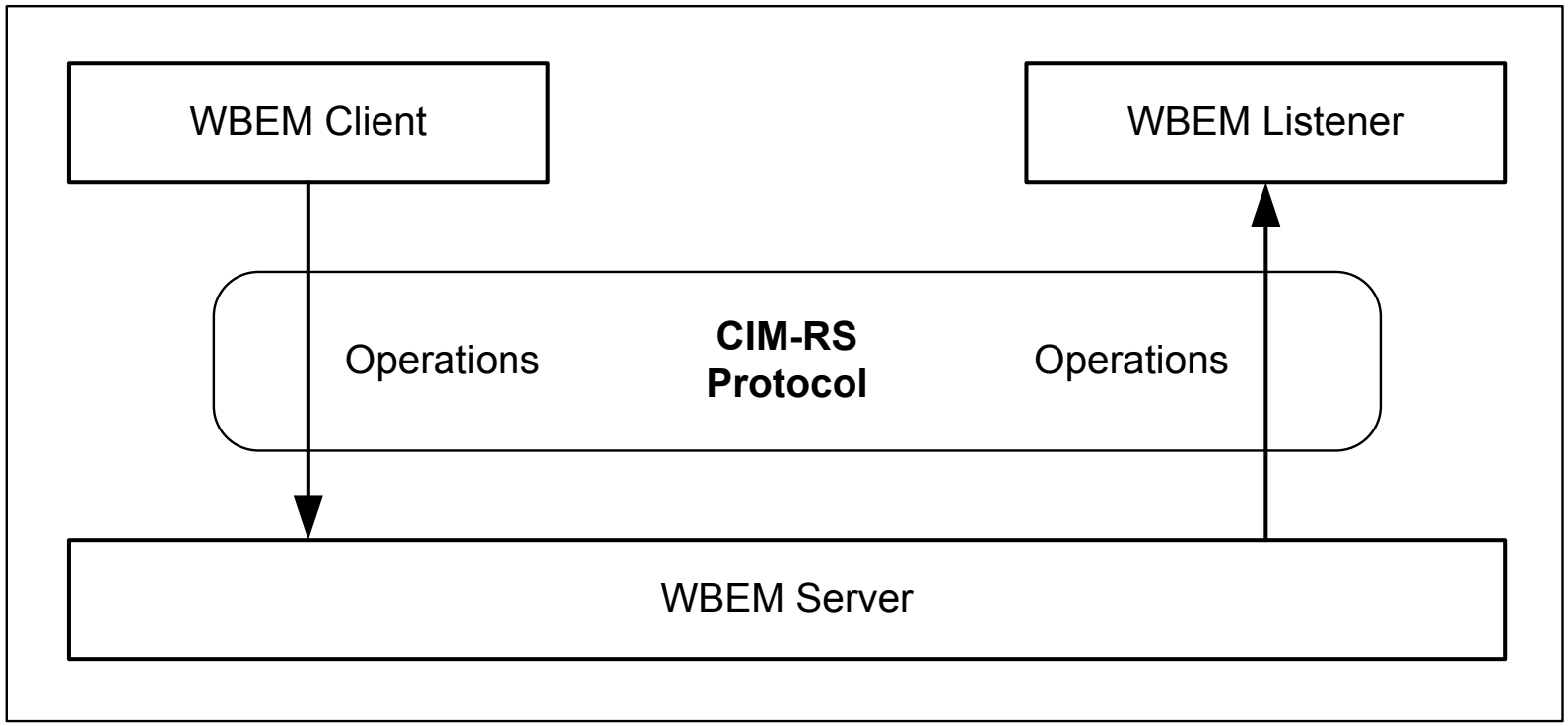
  -> CIM-RS is implementable in a model-neutral way on existing WBEM infrastructures
- No impact on provider APIs or providers
  - Works with JSR-48 and CMPI provider APIs
  - Providers will work with CIM-RS unchanged

Status of CIM-RS

- CIM-RS 1.0 has been released as a DMTF Standard in 1/2013:
  - **DSP0210: CIM-RS Protocol**
    - Defines the use of HTTP methods for accessing WBEM servers
  - **DSP0211: CIM-RS Payload Representation in JSON**
    - Defines a CIM-RS payload representation in JSON (Javascript Object Notation)
  - **DSP2032: CIM-RS White Paper**
    - Provides an overview of CIM-RS

- The incubator version of CIM-RS published in 2011 is no longer relevant:
  - DSP-IS0201, DSP-IS0202, DSP-IS0203

CIM-RS protocol participants



- CIM-RS is a WBEM protocol (like CIM-XML, WS-Man)

CIM-RS Usage of HTTP Methods

Core methods of HTTP 1.1 (RFC2616):

- GET
  - Retrieve an instance
  - Retrieve all instances of a class (enumeration)
  - Retrieve associated instances (association traversal)
  - Retrieve server / listener entry point resource
- POST
  - Create an instance
  - Invoke methods (static and non-static)
  - Deliver an indication
- PUT
  - Modify an instance (fully & partial)
- DELETE
  - Delete an instance

- No more use of OPTIONS and PATCH methods used by incubator CIM-RS

"CIMishness" of incubator CIM-RS

- Incubator version of CIM-RS is optimized for implementation as a protocol adapter on existing WBEM infrastructures:
  - Operations for namespaces, classes, instances, qualifier types
    - Following the semantics defined in Generic Operations (DSP0223)
  - Full set of parameters of Generic Operations
    - Directly represented as URI query parameters
  - Canonical discovery hierarchy:
    - WBEM server -> Namespace collection -> Namespace -> Class collection -> Class -> Instance collection -> Instance
  - Single URI on an instance for traversal of all associations from that instance
    - Association selection by means of query parameters
    - Directly optimized for GenOps association traversal operations
  - Single URI on an instance for invocation of all methods on that instance
    - Method selection by means of data in request payload
    - Directly optimized for GenOps method invocation operation

-> Too CIM-ish compared to a RESTful interface designed without CIM in mind

"DeCIMification" in standard CIM-RS

- The standard version of CIM-RS is less CIM-oriented
- Harvested good ideas from CIMI REST work
- Changes in standard CIM-RS:
  - Deal with resources representing managed objects instead of resources representing CIM meta-entities (i.e. namespaces, classes, instances, qualifier types)
  - Drop operations on namespaces, classes, qualifier types
    - Only instance-level operations remain
    - Use upcoming Schema Inspection Model to access namespaces, classes, qualifier types through instances
  - One URI for each implemented method
    - Only for implemented methods, not for all methods defined
  - One URI for each implemented association traversal
    - Only for implemented association traversals, not for all associations defined
  - Clean up query parameters
    - More readable names
    - Less bound to parameters of Generic Operations

REST Resources in CIM-RS

Types of resources in a WBEM server:

Resource type	Meaning
Server entry point resource	The entry point resource of a server; representing capabilities of the server, and the target resources below for each namespace
Instance creation resource	Target resource for creating instance resources (and thus, managed objects)
Method invocation resource	Target resource for invoking static methods
Instance enumeration resource	Target resource for enumerating instance resources by class
Instance resource	A modeled object in the managed environment
Instance collection resource	A collection of instance resources
Reference collection resource	A collection of references (to instance resources)

Types of resources in a WBEM listener:

Resource type	Meaning
Listener entry point resource	The entry point resource of a listener, representing capabilities of the listener
Listener destination resource	Target resource for delivering indications

CIM-RS Operations – WBEM server

Operations targeting a WBEM server:

Function	CIM-RS operation (HTTP method & target resource)
Get capabilities of server and target resources	GET to server entry point resource (well-known URI)
Create instance of a class (managed object)	POST to instance creation resource of namespace
Invocation of non-static method on an instance	POST to that method invocation resource (on instance)
Invocation of static method on a class	POST to method invocation resource of namespace
Enumerate instances of a class	GET to instance enumeration resource of namespace
Retrieve instance	GET to instance resource
Delete instance	DELETE to instance resource
Modification of instance (fully)	PUT to instance resource
Modification of instance (partial)	PUT to instance resource, with subsetted property set
Association traversal	GET to source instance with refer & expand query parameters
Fetching results: Retrieve instance collection	GET on instance collection resource
Fetching results: Retrieve reference collection	GET on reference collection resource

CIM-RS Operations – WBEM listener

Operations targeting a WBEM listener:

Function	CIM-RS operation (HTTP method & target URI)
Get capabilities of listener and listener destinations	GET to listener URI (top-level resource)
Deliver indication	POST to listener destination URI

CIM-RS Resource URIs

- Only the top-level resources have URIs with a defined format :
  Server: `http://{host+port}/cimrs`
  Listener: `http://{host+port}/cimrs`

- All other resources have URIs with an implementation-defined format :

  – URI format is opaque to client, except for:
    - URI normalization (as defined in RFC3986)
    - Query parameters in the URI

  – URIs are never parsed, constructed or modified by clients
    - Exceptions see above

  – URIs are discovered through operations
    - Discovery starts with the top-level resources (server, listener)
    - Responses contain URIs for further operations

URI query parameters in CIM-RS (1)

- Query parameter design has changed quite a bit since incubator version of CIM-RS
- Query parameters on GET instance resource:

`$properties={prop},…,{prop}`	limit returned instances to only these properties
`$methods={meth},…,{meth}`	limit returned instances to only these (non-static) method links
`$expand={nav-path},…,{nav-path}`	include instance collections for the specified navigation paths
`$refer={nav-path},…,{nav-path}`	include reference collections for the specified navigation paths
`$max={maxsize}`	limit size of returned collections in paged retrieval
`$continueonerror=true/false`	control continuation on errors within paged retrieval
`$pagingtimeout={time}`	specify inactivity timeout for paged retrieval

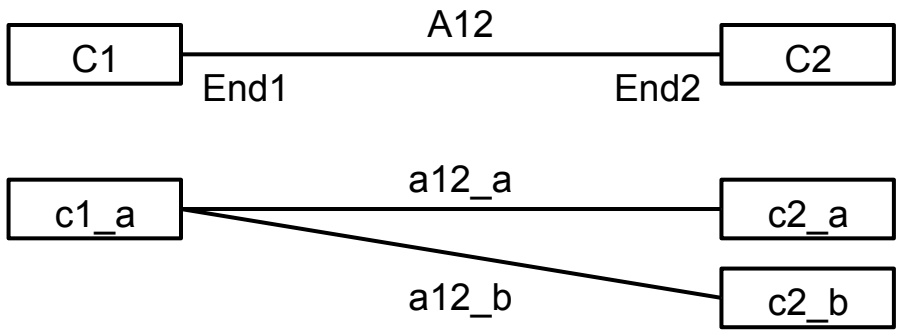
- Query parameters on GET instance enumeration resource:

  all query parameters of GET instance resource

`$class={classname}`	specify enumeration class
`$filter={filter-query}`	limit returned instances to match filter

Navigation properties

- *Navigation properties* are "artificial" properties that are included in returned instances upon request of the client.
- They represent associated or referencing instances or references thereof, possibly across multiple hops (equivalent of traditional Associators & References operations).
- Client specifies the inclusion of navigation properties by specifying *navigation paths*, using the **`$expand`** and **`$refer`** query parameters.
- Example:



Class Diagram

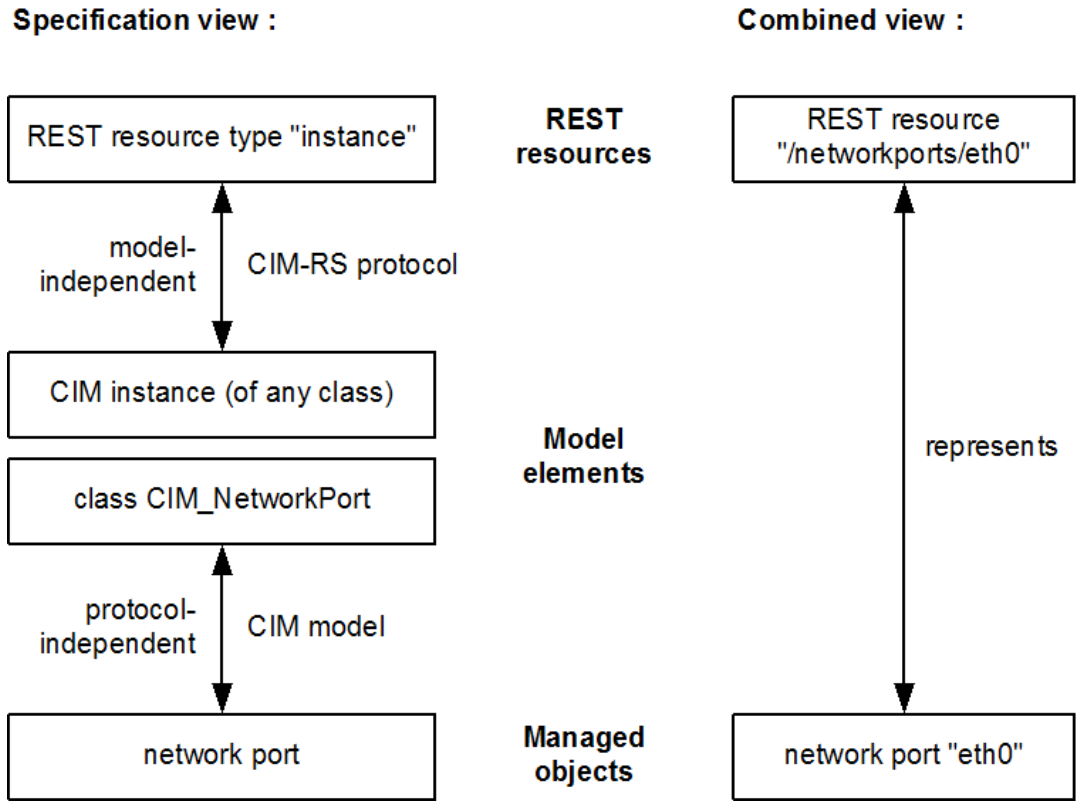
Instance Diagram

```
GET {c1_a}?$expand=A12.End2

Instance c1_a:
    A12.End2: InstanceCollection:
        Instance c2_a: ...
        Instance c2_b: ...
```

```
GET {c1_a}?$refer=A12.End2

Instance c1_a:
    A12.End2: ReferenceCollection:
        Ref c2_a: ...
        Ref c2_b: ...
```

Two-staged resource mapping

- CIM-RS protocol definition is model-neutral
- As a result, mapping of managed objects to CIM-RS REST resources is two-staged:



- **CIM-RS protocol definition talks about "instances" and "instance collections"**
  - and leaves mapping of instances to managed objects ("resources") to a CIM model

Why JSON as a payload encoding ?

- Perceived to be simple, easy to use and understand
  - JavaScript standard (ECMA-262, 5th edition) defines JSON on only a few pages
- Results in a compact payload representation
- JSON and RESTful services are often used together

JSON in 2 minutes

- Empty Object :

```
{ }
```

- Empty Array :

```
[ ]
```

Array elements can be of different types

- Object with a property :

```
{ "Caption": "Hello World" }
```

- Value types are Array, Object, String, Number, Boolean :

```
{ "AnArray": [ 1,2,3 ],
  "AnObject": { … },
  "AString": "Hello World",
  "ANumber": 42,
  "ABoolean": true }
```

- Objects and Arrays can be arbitrarily nested

  e.g. an array with two object-typed elements :

```
[ { "One": 1 }, { "Two": 2 } ]
```

An instance in JSON

CIM_ComputerSystem instance :

```
{
  "kind": "instance",
  "self": "/cimrs/smash/computersystems/843752",
  "class": "CIM_ComputerSystem",
  "properties": {
    "Name": "MyComputer1",
    "Description": "My very first computer",
    ...
  },
  "methods": {
    "RequestStateChange": "/cimrs/smash/computersystems/843752/RequestStateChange",
    "SetPowerState": "/cimrs/smash/computersystems/843752/SetPowerState"
  }
}
```

First-level properties:

- "kind": format of the payload element (to make payload elements self-describing)
- "self": URI of the represented instance (to make it self-describing)
- "class": name of the CIM creation class of the represented instance
- "properties": properties of the instance (including navigation properties)
- "methods": invocation links for invoking implemented non-static CIM methods on the instance

An instance collection in JSON

Collection of CIM_ComputerSystem instances :

```
{
  "kind": "instancecollection",
  "self": "/cimrs/smash/computersystems",
  "class": "CIM_ComputerSystem",
  "instances": [
    { ... },
    { ... },
    { ... }
  ]
}
```

First-level properties:

- "kind": format of the payload element (to make payload elements self-describing)
- "self": URI of the represented instance collection (to make it self-describing)
- "class": name of a common superclass of the instances in the collection (if one exists)
- "instances": array of instances in the collection (each instance as shown on previous page)

Retrieving an instance

```
GET /cimrs/smash/computersystems/843752&$refer=CIM_SystemDevice.PartComponent HTTP/1.1
Host: example.com
Accept: application/json;version=1.0
X-CIMRS-Version: 1.0.0
```

```
HTTP/1.1 200 OK
Date: Wed, 14 Sep 2011 08:47:22 GMT
Content-Length: XXX
Content-Type: application/json;version=1.0.1
X-CIMRS-Version: 1.0.1

{ "kind": "instance",
  "self": "/cimrs/smash/computersystems/843752",
  "class": "CIM_ComputerSystem",
  "properties": {
    "Name": "MyComputer1",
    "Description": "My very first computer",
    "CIM_SystemDevice.PartComponent": {
      "kind": "referencecollection",
      "self": "/cimrs/smash/computersystems/843752/systemdevices_ref",
      "class": "CIM_LogicalDevice",
      "references": [
        "/cimrs/smash/devices/100042",
        "/cimrs/smash/devices/100043" ] } },
  "methods": {
    "RequestStateChange": "/cimrs/smash/computersystems/843752/RequestStateChange",
    "SetPowerState": "/cimrs/smash/computersystems/843752/SetPowerState" }
}
```

no "next" element
-> collection completely contained

Invoking a method

```
POST /cimrs/smash/computersystems/843752/RequestStateChange HTTP/1.1
Host: example.com
Accept: application/vnd.dmtf.cimrs+json;version=1.0
Content-Length: XXX
Content-Type: application/vnd.dmtf.cimrs+json;version=1.0.0
X-CIMRS-Version: 1.0.0

{
  "kind": "methodrequest",
  "self": "/cimrs/smash/computersystems/843752/RequestStateChange",
  "method": "RequestStateChange",
  "parameters": {
    "RequestedState": 2,
    "TimeoutPeriod": 0
} }
```

Acceptable payload encodings in response

Encoding used in this request payload

CIM-RS protocol version used for this request

```
HTTP/1.1 200 OK
Date: Wed, 14 Sep 2011 08:47:22 GMT
Content-Length: XXX
Content-Type: application/vnd.dmtf.cimrs+json;version=1.0.1
X-CIMRS-Version: 1.0.1

{
  "kind": "methodresponse",
  "self": "/cimrs/smash/computersystems/843752/RequestStateChange",
  "method": "RequestStateChange",
  "returnvalue": 0,
  "parameters": {
    "Job": "/cimrs/smash/jobs/653751"
} }
```

Encoding used in this response payload

CIM-RS protocol version used for this response

Summary & Take-aways

- CIM-RS is a RESTful protocol to CIM modeled resources
- CIM-RS protocol is model neutral
  - Works with every CIM model (not just for DMTF CIM Schema)
  - No further model-specific "mapping specs" required
- CIM-RS supports an open set of payload encodings
  - Currently defined: JSON (DSP0211)
- JSON payload encoding for CIM-RS is model neutral
  - No need to publish a payload specific representation of the CIM Schema
- In CIMOM/provider based infrastructures:
  - CIM-RS implementation can be done once as a protocol handler
  - Providers can be used unchanged