Document Number: DSP-IS0203

Date: 2011-10-26

Version: 1.0.0

CIM-RS White Paper

Document Type: White Paper

Document Status: DMTF Informational

Document Language: en-US

CONTENTS

Tables

Abstract

This white paper provides background information for the informational specifications DSP-IS0201, *CIM Operations over RESTful Services*, and DSP-IS0202, *CIM-RS Binding to JSON*. This white paper explains some of the decisions in these specifications and gives the reader insight into when the use of CIM-RS may be appropriate. Some of the considerations in choosing payload encodings such as JSON or XML are also discussed.

This white paper is targeted to potential users of DSP-IS0201 and DSP-IS0202 who are considering developing a server-side interface to a CIM implementation that follows REST principles, or to a client that consumes such an interface.

Foreword

The *CIM-RS White Paper* (DSP-IS0203) was prepared by the CIM-RS Incubator.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about the DMTF, see http://www.dmtf.org.

Acknowledgments

The DMTF acknowledges the following individuals for their contributions to this document:

- Andreas Maier, IBM (editor)
- Marvin Waschke, CA Technologies (editor)
- Cornelia Davis, EMC
- George Ericson, EMC
- Wojtek Kozaczynski, Microsoft
- Lawrence Lamers, VMware
- Bob Tillman, EMC

Document Conventions

Typographical Conventions

The following typographical conventions are used in this document:

- Document titles are marked in *italics*.
- Important terms that are used for the first time are marked in *italics*.
- Terms include a link to the term definition in the "Terminology" clause, enabling easy navigation to the term definition.

CIM-RS White Paper

1 Executive Summary

The DMTF Common Information Model (CIM) is a conceptual information model for describing computing and business entities in Internet, enterprise, and service-provider environments. CIM uses object-oriented techniques to provide a consistent definition of and structure for data. The CIM Schema establishes a common conceptual framework that describes the managed environment.

CIM provides a foundation for IT management software that can be written in one environment and easily converted to operate in a different environment. They also facilitate communication between software managing different aspects of IT infrastructure. In this way, CIM provides a basis for an integrated IT management environment that is more manageable and less complex than environments based on narrower and less consistent information.

CIM is built on object-oriented principles and provides a consistent and cohesive programming model for IT management software. One of the developing trends in enterprise network software architecture in recent years has been Representational State Transfer (REST). REST represents a set of architectural constraints that have risen from the experience of the World Wide Web. Developers have discovered that the architecture of the Web offers some of the same benefits in simplicity and reliability to enterprise software as it has provided for the Internet. IT management is an important application of enterprise software, and there is growing interest in using CIM- based software in an architecture that adheres to REST constraints.

Fortunately, CIM is built on object-oriented principles and follow basic architectural principles that largely fit well into RESTful architectures. Therefore, the CIM-RS Incubator undertook to develop specifications for a RESTful protocol tailored to the needs of CIM.

2 Terminology

In this document, some terms have a specific meaning beyond the normal English meaning. Those terms are defined in this clause.

Some of the terms and abbreviations defined in DSP0004 and DSP0223 are used in this document but are not repeated in this clause.

2.1

application state

The state that indicates where an application is in completing a task. In a RESTful system, the client is solely responsible for application or session state. The server is only responsible for resource state, the state of the resources managed by the service. An example of resource state is the account balance in a banking service, which would be maintained by the server. An example of application state is a specific client that has posted a deposit and is waiting for it to clear. Only the client would track the fact that it has posted a deposit request.

2.2

Atom

The term Atom applies to two related standards. The Atom Syndication Format is an XML-based format for publishing Web content and metadata. The Atom Publishing Protocol (AtomPub or APP) is an HTTP-based protocol for publishing and editing Web resources. See RFC4287 for the Atom Syndication Format. See RFC5023 for the Atom Publishing Protocol.

2.3

CIMOM

CIM Object Manager

2.4

CIM-RS

RESTful Services for CIM

The protocol covered by this white paper.

2.5

HATEOAS

Hypertext As The Engine Of Application State

The practice of using links embedded in resource representations to advertise further possible activities or resources related to the application .For example, an “order” link might be placed in the resource representation for an item offered in a catalog. The presence of the order link indicates that the item can be ordered and represents a path to order the item. In a visual representation, the “order” link would appear as a button on the screen. When the button is clicked, a POST or PUT HTTP method targeting the resource identifier provided in the link would be issued and would cause the item to be ordered. The returned resource represents the next application state, perhaps a form for entering quantity and shipping method.

2.6

HTTP content negotiation

Negotiation between HTTP clients and HTTP servers to determine the format of the content transferred. When a client makes a request, it lists acceptable response formats by specifying MIME types in an `Accept` header. Thus, the server is able to supply different representations of the same resource identified with the same resource identifier. A common example is GIF and PNG images. A browser that cannot display PNGs can be served GIFs based on the `Accept` header. In a RESTful system, the choice is more often between XML and JSON. For details, see RFC2616.

2.7

IANA

Internet Assigned Numbers Authority, http://www.iana.org/

2.8

idempotent HTTP method

An HTTP method with the behavior that (aside from error or expiration issues) the side effects of N consecutive identical requests are the same as for a single one of those requests. RFC2616 requires the HTTP methods GET, HEAD, PUT and DELETE to be idempotent. HTTP methods that have no side effects (that is, safe methods) are inherently idempotent. For details, see RFC2616.

2.9

International Resource Identifier (IRI)

URIs that are expanded to use the Universal Character Set (defined in ISO/IEC 10646, also known as Unicode), including non-alphabetic characters like Arabic and Chinese in addition to ASCII. When appropriate, an IRI can be used instead of a URI. Typically, a REST resource identifier is a URI or an IRI. IRIs are defined in RFC3987.

2.10

JSON

JavaScript Object Notation, defined in chapter 15 of ECMA-262.

2.11

media type, MIME type

File format types originally defined for email attachments but now used in other protocols, including HTTP where they are used to specify the payload format. Media types consist of a type, subtype and optional parameters. Examples are text/plain (plain text), text/html (HTML markup), and application/json (JSON). Types and subtypes are registered with IANA. For details, see RFC2045 and RFC2046.

2.12

MIME

Multipurpose Internet Mail Extension, defined in RFC2045 and RFC2046.

2.13

resource

An entity that can be identified and represented in a RESTful protocol. Example resources are devices, documents, or events.

2.14

resource identifier

An unambiguous reference to (or address of) a resource, in some format. Usually, URIs or IRIs are used as resource identifiers. A resource may have more than one resource identifier.

2.15

resource representation

A representation of a resource or some aspect thereof, in some format. A particular resource may have any number of representations. The format of a resource representation is identified by a media type.

2.16

resource state

The state of a resource managed by a RESTful service. Contrast with application state.

2.17

REST

Representational State Transfer

A style of software architecture for distributed systems that is based on addressable resources, a uniform constrained interface, representation orientation, stateless communication, and state transitions driven by data formats. Usually REST architectures use the HTTP protocol, although other protocols are possible. See *Architectural Styles and the Design of Network-based Software Architectures* for the original description of the REST architectural style.

2.18

RPC

Remote Procedure Call

An implementation of a function in which a call to the function occurs in one process and the function is executed in a different process, often in a remote location linked by a network. RPC-based systems are often contrasted with RESTful systems. In a RESTful system, the interactions between client and server follow the REST constraints and the design focus is on the resources. In an RPC-based system, the design focus is on the functions invoked, and there is not necessarily even the notion of well-defined resources.

2.19

safe HTTP method

An HTTP method that has no side effects. RFC2616 requires the HTTP methods GET and HEAD to be safe. By definition, an HTTP method that is safe is also idempotent.

2.20

Universal Resource Identifier (URI)

Universal Resource Locator (URL)

Universal Resource Name (URN)

International Resource Identifier (IRI)

URLs and URNs are types of URIs. IRIs are URIs that are expanded to use the Universal Character Set (defined in ISO/IEC 10646, also known as Unicode), including non-alphabetic characters like Arabic and Chinese in addition to ASCII. When appropriate, an IRI can be used instead of a URI. Typically, a REST resource identifier is a URI or an IRI. URIs are defined in RFC3986. URNs are defined in RFC2141. IRIs are defined in RFC3987.

3 Rationale for a RESTful interface for CIM

There has been a great deal of interest in constructing RESTful enterprise applications in the last few years and this interest has inspired the specification of CIM-RS. To understand the origins of this interest, the nature of REST and its relationship to IT management must be explored.

Enterprise applications are being built more and more frequently on architectures that involve remote network connections to some part of the implementation of the application. These connections are often via the Internet. This is especially true with the rise of cloud computing.

REST is a set of architectural constraints that were designed around the features of the Internet. For example, REST constraints are designed to assure that applications that follow constraints will have maximum benefit from typical Internet features such as caches, proxies, and load balancers.

In addition, REST constraints are closely tied to the design of HTTP, the primary application level protocol of the Internet. In fact, the prime formulator of REST, Roy Fielding, was also an author of the HTTP standard. Consequently, REST was designed to take full advantage of HTTP and HTTP meets the needs of REST.

Some of the specific benefits that have been experienced in RESTful applications are as follows:

- Simplicity. REST limits itself to the methods implemented in HTTP and runs directly on the HTTP stack. Note, however, that this simplicity can be deceptive. The design effort to comply with REST may engender its own complexity.
- Resilience in the face of network disturbance. One of the hallmarks of a RESTful application is a stateless relationship between the server and the client. Each request from the client contains all the history the server needs to respond to the client. Therefore, when requests are self-contained and independent, if a server becomes inaccessible recovery does not require unwinding a stack and complex recovery logic..
- Upgradability. The operations available in a RESTful application are discovered by the client as the processes occur. Consequently, in some cases, the server implementation often may be upgraded transparently to the client. In some cases, a well-designed client may be able to take advantage of new features automatically.

Although these are important benefits, it is important to note that REST is not a panacea. Some of the limitations of REST are as follows:

- Not all activities are easily compatible with its constraints.
- Not every operation fits easily into the stateless paradigm.
- The discoverability of RESTful applications may break down as applications become more complex and transactions become more elaborate.

Nevertheless, as a result of the benefits, a substantial number of developers of IT management applications that use CIM have turned to REST. Therefore, there is a need for a specification for a uniform protocol that will promote interoperability between RESTful CIM based applications.

4 Goals of the CIM-RS informational specifications

Unlike the usual informational documents produced by DMTF incubators, the CIM-RS Incubator produced two informational specifications that rather precisely describe the RESTful protocol for CIM:

- DSP-IS0201 (*CIM Operations Over RESTful Services*) defines a RESTful protocol that follows the semantics of generic operations (DSP0223). The format of the payload can be negotiated between client and server and is not defined in this document.
- DSP-IS0202 (*CIM-RS Binding to JSON*) defines a payload representation in JSON.

A payload representation in XML has not been defined by the CIM-RS Incubator, because one of the existing ones was envisioned to be used.

The purpose of these two informational specifications is to explore the a RESTful protocol for CIM in more detail, also outside of the DMTF.

Specific aspects explored in these informational specifications are as follows:

- Does the generic-operations semantic lead to a reasonably RESTful protocol?
- Does it work to generically provide access to CIM modeled resources merely by providing a new protocol and without redefining the CIM model?
- Does it work to implement the RESTful protocol as a protocol adapter on top of existing WBEM infrastructure components, without changing the lower levels of the server-side instrumentation (for example, providers), and how expensive is such an implementation?
- Is JSON sufficiently capable for representing the protocol payload for accessing CIM modeled resources?

It is important to understand that these two specifications are informational, that is, they are not normative DMTF standards. Any implementation of these two documents should be treated as experimental or prototypical, in order to provide feedback on the aspects described above.

5 Characteristics of a RESTful protocol and CIM-RS

The characteristics of a RESTful protocol are not standardized or otherwise defined normatively. The principles and constraints of the REST architectural style were originally described by Roy Fielding in chapter 5 of *Architectural Styles and the Design of Network-based Software Architectures*. Roy Fielding's blog entry *REST APIs must be hypertext driven* provides further insight into REST principles. While these descriptions of the REST architectural style are not limited to the use of HTTP, the HTTP protocol comes close to supporting that style and obviously has a very broad use.

The CIM-RS protocol is based on HTTP and supports the REST architectural style to a large degree. The following list describes to what extent the typical REST constraints are satisfied by the CIM-RS protocol:

- **Client-Server:** The participants in the CIM-RS protocol are the WBEM client, the WBEM server, and the WBEM listener. There is a client-server relationship between the WBEM client and WBEM server, and one between the WBEM server and WBEM listener, where the WBEM server acts as a client to the WBEM listener. Thus, the WBEM server has two roles: To act as a server in the interactions with the WBEM client, and to act as a client in the interactions with the WBEM listener.

  This REST constraint is fully satisfied in CIM-RS.

- **Stateless:** Interactions in CIM-RS are self-describing and stateless in that the servers (that is, the WBEM server in its server role, and the WBEM listener) do not maintain any application state or session state.

  This REST constraint is fully satisfied in CIM-RS.

  NOTE: Pulled enumeration operations as defined in DSP0223 maintain the enumeration state either on the server side or on the client side. In both approaches, the client needs to hand back and forth an opaque data item called enumeration context, which is the actual enumeration state in the case of a client-maintained enumeration state, or a handle to the enumeration state in the case of a server-maintained enumeration state. CIM-RS supports both of these approaches. It is possible for a server to remain stateless, as far as the enumeration state goes, by implementing the client-based approach. The approach implemented by a server is not visible to a client, because the enumeration context handed back and forth is opaque to the client in both approaches.

- **Cache:** The HTTP methods used in CIM-RS are defined in RFC2616 and RFC5789 (for PATCH). As a result, they are cacheable as defined in RFC2616.

  This REST constraint is fully satisfied in CIM-RS.

  NOTE: RFC2616 defines only the result of HTTP GET methods to be cacheable.

- **Uniform interface:** The resources represented in CIM-RS are CIM namespaces, CIM classes, CIM qualifier types, and CIM instances. CIM-RS defines a uniform interface for creating, deleting, retrieving, replacing, and modifying these resources, based on HTTP methods.

  This REST constraint is satisfied in CIM-RS, with the following deviation:

  CIM methods can be invoked in CIM-RS through the use of HTTP POST. This may be seen as a deviation from the REST architectural style, which suggests that any "method" be represented as a modification of a resource. However, that is not practical in CIM, because significant effort has been put into the definition of CIM method semantics in the CIM Schema and in management profiles, and into existing implementations of these methods. Therefore, it is necessary to support CIM method invocation as an interaction in CIM-RS.

- **Layered system:** Layering is an inherent part of CIM, because it defines a CIM model of managed objects in a managed environment and thus restricts knowledge of a client to only the modeled representation of the managed environment. CIM-RS represents the entities modeled in CIM, separating the concerns of the RESTful protocol from resource modeling concerns. In addition, CIM-RS supports the use of HTTP intermediaries (for example, caches and proxy servers).

  This REST constraint is fully satisfied in CIM-RS.

- **Code-On-Demand:** CIM-RS does not provide for sending any code back to the client.

  This REST constraint is not satisfied, but such functionality is not needed in CIM.

Beyond that, CIM-RS has the following other characteristics:

- **Model independence:** CIM-RS does not define or prescribe the use of a particular CIM model. However, it does require the use of a CIM model defined using the CIM infrastructure/architecture. This allows reusing the traditional DMTF technology stack and its implementations, with only minimal impact to existing implementations. For details on CIM-RS resources, see clause 6.
- **Opaqueness of resource identifiers:** CIM-RS uses URIs as resource identifiers and defines all but a top-level URI to be opaque to clients. That allows reuse of the URIs supported by existing WBEM protocols without any remapping, as well as the use of new URI formats in the future. It encourages a client style of programming that is more RESTful than when clients parse resource URIs. For details on CIM-RS resource identifiers, see clause 7.
- **Consistency of operations:** Beyond following the REST constraints, the CIM-RS operations are consistent with the generic operations defined in DSP0223. This allows implementing CIM-RS as an additional protocol in existing WBEM infrastructures, causing impact only where it is necessary (that is, at the protocol level), leveraging existing investments. For details on CIM-RS operations, see clause 8.
- **Supports use of new RESTful frameworks:** Because CIM-RS is a RESTful protocol, it supports the use of new RESTful frameworks both on the client side and on the server side, without tying client application development to the use of traditional WBEM clients or CIM client APIs, and without tying server instrumentation development to the use of traditional WBEM servers, such as CIMOMs and providers.

6 Resources in CIM-RS

The REST architectural style allows for the representation of rather static entities, such as disk drives, or entities with highly varying states, such as a metric measuring the amount of available disk space at a specific point in time, or even entities that dynamically come into existence or cease to exist, such as file system mounts.

CIM-RS represents CIM modeling entities as resources, and thus it can represent all these different kinds of entities.

The resources supported by CIM-RS are defined at the level of CIM modeling entities:

- **CIM instances:** They represent certain aspects of managed objects in the managed environment, as defined by their CIM class.
- **CIM classes:** They represent the definition of the structure of properties for CIM instances, methods that can be invoked, and qualifier values (that is, metadata).
- **CIM qualifier types:** They represent the definition of the name, data type and other characteristics of qualifiers (that is, the qualifier type specification).

- **CIM namespaces:** They represent a container and naming space for CIM instances, CIM classes and CIM qualifier types.

In most cases, a client application using CIM-RS will need to interact only with CIM instance-level resources. The knowledge about the data structure (that is, properties) of these instances, about the methods that can be invoked, and about the semantics indicated by qualifier values on their classes, properties, methods and method parameters is typically built into the client application at design time. Only in cases where the client application has a need to dynamically discover such knowledge does it need to interact with CIM class-level resources. CIM qualifier type-level resources are supported by CIM-RS mainly to be consistent with other WBEM protocols and are typically used by client applications to support the loading of qualifier types into a WBEM server as part of its installation. Consistent with other WBEM protocols, the enumeration interactions in CIM-RS are scoped to a single CIM namespace, creating the need to represent CIM namespaces as resources.

Because these kinds of resources are defined at the level of CIM modeling entities and not at the level of a particular CIM model (for example, “CIM instance” instead of “network interface”), the definition of CIM-RS is independent of any particular CIM model and thus applies to any current or future CIM model defined by DMTF or by others.

This model independence allows CIM-RS to be implemented in an existing WBEM server as an additional protocol, or as a gateway in front of an existing unchanged WBEM server, leveraging the investment in that implementation. Specifically, in WBEM servers supporting a separation of CIMOM and providers, adding support for CIM-RS typically drives change only to the CIMOM; it does not drive change to the providers. On the client side, existing WBEM client infrastructures that provide client applications with a reasonably abstracted API can implement CIM-RS as an additional protocol, shielding existing client applications from the new protocol.

In order to make this work, CIM-RS must support the same operation semantics as the operations supported at client APIs, provider APIs and existing WBEM protocols. As a central definition for these operation semantics, the generic operations defined in DSP0223 were used by CIM-RS. For more details about the operations supported by CIM-RS, see clause 8.

Because CIM-RS is a RESTful protocol, it supports the use of new RESTful frameworks both on the client side and on the server side, without tying client application development to the use of traditional WBEM clients or CIM client APIs, and without tying server instrumentation development to the use of traditional WBEM servers, such as CIMOMs and providers.

Of course, combinations of new RESTful frameworks and traditional WBEM infrastructure are also possible. A typical scenario would be the use of a new RESTful framework in a client application, with a traditional WBEM server whose CIMOM portion is extended with CIM-RS protocol support.

It is key to understand that the model independence of CIM-RS and the resulting benefits are its main motivation and are a key differentiator with other DMTF approaches for using REST. The model independence is what positions CIM-RS to be a first-class member of the traditional DMTF technology stack, leveraging a large number of standards defined by DMTF and others (most notably, the CIM architecture/infrastructure, the CIM Schema, and management profiles defined by DMTF and others).

On the downside, the model independence of CIM-RS causes a certain indirection in dealing with the managed objects: CIM-RS resources representing CIM instances and CIM classes can be understood only after understanding the CIM model they implement. The CIM model is defined by a CIM schema and typically by a number of management profiles that scope and refine the use of the CIM Schema to a particular management domain. So the number of documents to read before a client application could reasonably be developed against a CIM instrumentation supporting CIM-RS might be quite significant. On the other hand, developing a client application in this context would be no more complex than developing a client application against a CIM instrumentation supporting other existing WBEM protocols.

Following the REST architectural style, any entity targeted by an operation in the CIM-RS protocol is considered a resource, and the operations are simple operations such as the HTTP methods GET, POST, PUT, PATCH, and DELETE.

The simplicity of these operations requires that details, such as the difference between retrieving a single resource versus a collection of resources or retrieving a resource versus navigating to a related resource, be "encoded" into the resource definitions. This leads to a number of resource variations.

Note that the real-world entities are not called "resources" in this document. Rather, the standard DMTF terminology is used, where such real-world entities are called "managed objects," and the real world itself is called the "managed environment." This terminology allows distinguishing resources as represented in the RESTful protocol from the managed objects they sometimes correspond to, in part or in whole.

CIM-RS defines the following resources, as listed in Table 1.

Table 1 – CIM-RS resources and what they represent

CIM-RS resource	Represents
WBEM server	Top-level CIM-RS resource of a WBEM server
Namespace collection	A collection of CIM namespaces in a WBEM server
Namespace	A single CIM namespace in a WBEM server
Class collection	A collection of CIM classes in a CIM namespace
Class	A single CIM class in a CIM namespace
Class associator collection	A collection of CIM classes associated to a given CIM class
Class reference collection	A collection of CIM association classes referencing a given CIM class
Class method invocation	A particular CIM method that can be invoked on a given CIM class
Instance collection	A collection of CIM instances in a CIM namespace
Instance	A single CIM instance in a CIM namespace
Instance associator collection	A collection of CIM instances associated to a given CIM instance
Instance reference collection	A collection of CIM association instances referencing a given CIM instance
Instance method invocation	A particular CIM method that can be invoked on a given CIM instance
Qualifier type collection	A collection of CIM qualifier types in a CIM namespace
Qualifier type	A single CIM qualifier type in a CIM namespace
Instance query	An instance-level query targeting a CIM namespace

Each of these resources can be addressed using a resource identifier; for details, see clause 7.

Each of these resources has a defined set of operations; for details, see clause 8.

Each of these resources has a defined resource representation in each of the supported representation formats; for details, see clause 9.

CIM-RS supports retrieval of parts of resources. These parts are selected through query parameters in the resource identifier URI addressing the resource. That renders these parts as separate resources, following the principles in the REST architectural style.

For more details on CIM-RS resources, see DSP-IS0201.

7 Resource identifiers in CIM-RS

The REST architectural style recommends that all addressing information for a resource be in the resource identifier (and not, for example, in the HTTP header). In addition, it recommends that resource identifiers be opaque to clients and clients should not be required to understand the structure (or format) of resource identifiers or be required to assemble any resource identifiers.

CIM-RS generally follows these recommendations. In CIM-RS, resource identifiers are fully represented in URIs, without any need for additional information in HTTP headers or HTTP payload. However, these recommendations do not detail whether client-driven assembly and modification of the query parameter portion of a URI is also discouraged. In CIM-RS, the query parameter portion of a URI is normatively defined and may be assembled or manipulated by clients.

The only URI a client needs to know up front in CIM-RS is the resource identifier URI of the WBEM server (that is, the RESTful service itself). That is the only URI for which CIM-RS normatively defines a format.

After that starting point, any other URIs are server-defined and opaque to clients. They are discovered by clients by means of links returned along with resource representations. While CIM-RS does not define the format of these URIs, DSP-IS0201 provides an informational Annex defining one possible way to structure these URIs. This information is meant only as an example of a URI format, and it must not be relied on by clients.

The main benefit of client-opaque URIs is that servers can use existing URI formats, even in a mix of different kinds of URI formats, directly as the CIM-RS URIs. This typically saves both performance and space, and it allows flexibility for future URI formats.

For more details on resource identifiers in CIM-RS, see DSP-IS0201.

8 Operations in CIM-RS

The REST architectural style recommends that the operations on resources be simple and follow certain constraints. Although the use of HTTP is not a requirement for REST, the HTTP methods satisfy these constraints and are therefore a good choice for a RESTful system.

CIM-RS uses the HTTP methods OPTIONS, GET, POST, PUT, PATCH, and DELETE as its operations.

The HTTP OPTIONS method is used to discover implementation-dependent decisions on optional features at the WBEM server level, such as support for pulled enumerations or query support. An alternative would be to use the HTTP GET method on artificial resources representing these decisions, but this approach was dismissed in order to void the definition of additional artificial resources. Another alternative (for some of the cases covered with OPTIONS) would be to simply move the issue into the modeling space and to define a CIM model for such discovery. This approach was also dismissed, because these implementation decisions are at the CIM-RS protocol level, so handling them at that level was simpler.

The HTTP GET method is used to retrieve the targeted resource (single resource or collection resource). For collection resources, the result is an enumeration of the collection member resources (for example, CIM instances).

GET is also used to execute an instance-level query in the scope of a CIM namespace. The GET method in this case targets an artificial resource under the namespace resource, acting as a query execution point for that namespace. An alternative would be to use the HTTP POST method, but that would also require the definition of an artificial resource, and because a query is a read-only operation, it seems more appropriate to use GET.

The HTTP PUT method is used for replacing the targeted resource, while PATCH is used for partial update of the targeted resource. (The use of PUT for partial update violates its idempotency constraint and thus should be avoided.) Support for the HTTP PATCH method is still limited in the industry, but it is definitely the best fit, so it seemed appropriate to put a stake in the ground in favor of PATCH support. An alternative to PATCH would be the HTTP POST method; this might be a more practical choice given the limited PATCH support.

The HTTP DELETE method is used for removing the targeted resource.

The HTTP POST method when targeting a collection resource is used for creating a resource of the collection member type in that collection.

POST is the non-idempotent operation in HTTP that has many uses. The Request-URI in the header of a POST identifies the resource that will handle the entity enclosed in the message of the request, not necessarily the entity affected by the POST (see RFC2616, page 54). Following this pattern, POST is used for CIM method invocation by targeting a resource that represents the method invocation point of a CIM class or CIM instance.

The following descriptions provide more detail about some typical operations in CIM-RS:

- HTTP GET method targeting an instance resource:

  Retrieve a given instance's representation (that is, mainly property values as defined in the CIM class of that CIM instance, representing the values of some attributes of the managed object, or derivations or combinations thereof, dependent on the CIM modeling). The number of properties of that instance that get returned can be filtered using query parameters in the resource identifier URI.

- HTTP GET method targeting an instance collection resource:

  Retrieve a representation of the given collection, including representations of instances in the collection. The set of instances, and how much of each instance gets returned, can be filtered using query parameters in the resource identifier URI.

- HTTP GET method targeting an instance associator collection resource:

  Navigate to the CIM instances associated to a given CIM instance (that is, the CIM instance of the instance associator collection resource), and retrieve a representation of that instance collection, including representations of all instances in the collection. The association to traverse, the set of associated instances, and how much of each such instance gets returned can be filtered using query parameters in the resource identifier URI.

- HTTP POST method targeting an instance collection resource:

  Create a new instance of a given CIM class (that is, the CIM class of the instance collection resource). The CIM model defines whether that is possible, and, if so, what managed object needs to be created in the managed environment as a result.

- HTTP PUT or PATCH method targeting an instance resource:

  Modify the given CIM instance, either fully (PUT) or partially (PATCH). The CIM model defines whether that is possible, and, if so, what managed object needs to be modified in the managed environment as a result.

- HTTP DELETE method targeting an instance resource:

  Delete the given CIM instance. The CIM model defines whether that is possible, and, if so, what managed object needs to be deleted in the managed environment as a result.

- HTTP POST method targeting an instance method invocation resource:

Invoke the CIM method on the given CIM instance (that is, the CIM instance of the instance method invocation resource). Depending on the CIM model definition for that method, some activity happens in the managed environment, related to the managed object represented by that CIM instance.

For more details on operations in CIM-RS, see DSP-IS0201.

9 Data representation in CIM-RS

The REST architectural style promotes late binding between the abstracted resource that is addressed through a resource identifier and the resource representation that is chosen in the interaction between client and server.

CIM-RS follows this architecture by supporting multiple HTTP payload formats that are chosen through HTTP content negotiation.

The set of payload formats supported by CIM-RS is open for future extension, and currently consists of the following:

- JSON, as defined in DSP-IS0202
- XML, as defined in DSP0230 (WS-CIM)

JSON and XML were chosen because each is considered a premier choice for a representation format of RESTful systems, depending on the REST framework used and the technical and business environment.

A client or server needs to support at least one of these payload formats to conform to CIM-RS. However, there is no common subset of formats required to be supported in the current (Incubator-defined) CIM-RS. This could lead to a situation where the client supports only one format and the server supports only the other format, so they are unable to communicate. A future standardization of CIM-RS will need to provide a solution for that scenario.

It is important to understand that the entities to be represented in the HTTP payload are not only the resource representations. For example, operations such as method invocation or query execution require the representation of input and output data entities that are not resources (in the sense that they cannot be the target of CIM-RS operations).

Table 2 lists the protocol payload elements defined in CIM-RS. These are the entities that need to be represented in any payload format of CIM-RS.

Table 2 – CIM-RS protocol payload elements

Protocol payload element	Meaning
NamespaceCollection	Representation of namespace collection resource
Namespace	Representation of namespace resource
ClassCollection	Representation of class collection resource
Class	Representation of class resource
InstanceCollection	Representation of instance collection resource
Instance	Representation of instance resource
QualifierTypeCollection	Representation of qualifier type collection resource
QualifierType	Representation of qualifier type resource
MethodInvocationRequest	Input data for CIM method invocation

Protocol payload element	Meaning
MethodInvocationResponse	Output data for CIM method invocation
InstanceModificationRequest	Input data for partial update of CIM instance
InstanceQueryRequest	Input data for instance query execution
ErrorResponse	Output data in case of operation-level errors

The particular flavors of JSON and XML that have been chosen as CIM-RS payload formats deserve some attention.

Because CIM-RS is intended to provide a RESTful protocol for existing CIM models and infrastructures, one goal was to reuse existing DMTF standards where possible. DMTF has two standards for an XML representation of CIM elements:

- CIM-XML, as defined in DSP0201
- WS-CIM, as defined in DSP0230

CIM-XML supports the payload elements listed in Table 2 directly (that is, one can identify CIM-XML elements that correspond directly to these payload elements).

WS-CIM supports only a subset of the payload elements listed in Table 2. The elements not supported are the representations of class and qualifier type resources and collections. WS-CIM has been chosen as the XML payload format for CIM-RS because it represents CIM instances in a way that is more in tune with current Web Services frameworks. It is expected that this way of representing instances will also be supported in REST frameworks.

The resources not supported in WS-CIM are expected to be covered at the CIM modeling level in the future (by modeling classes and qualifier types in a future CIM schema inspection model). At that point, classes and qualifier types would no longer be resources that could be targeted, thus no longer requiring operations to support them. Inspection of classes and qualifier types would be handled at the instance level, and any modifications could be done through properly defined methods in such a model.

Anticipating such a model-based solution, it seemed acceptable to leave this gap in the current definition of the Incubator-defined CIM-RS.

Other alternatives for XML-based representations that were considered but not used in the current versions of DSP-IS0201 and DSP-IS0202 are as follows:

- Atom Syndication Format, as defined in RFC4287

  The Atom Syndication Format is targeted at the distribution of news entries. A number of the CIM-RS payload elements are not supported directly and would need to be added as extensions. The value provided by Atom to CIM-RS is relatively small. One argument in favor of using Atom was the expectation that a large number of tools would emerge that support Atom. However, given the number of extensions needed for a RESTful protocol such as CIM-RS, such tools would also need to support these extensions to become really useful. In the end, the minimal value provided by the Atom definitions outweighed these expectations on the tooling environment.

- Open Data Protocol, as defined on http://www.odata.org
- Google Data Protocol, as defined on http://code.google.com/apis/gdata/

For JSON, no specification exists that describes the representation of CIM elements in JSON. A number of approaches exist that attempt to map XML formats into JSON formats, but none of them resulted in a sufficiently simple and appealing use of JSON. The value of JSON lies in its simplicity; if mapping approaches diminish that simplicity, a lot of the value of using JSON is lost.

For these reasons, CIM-RS defines its own representation of payload elements in JSON, as described in DSP-IS0202 (*CIM-RS Binding to JSON*). That representation attempts to stay in the spirit of JSON. DSP-IS0202 currently uses a draft of the emerging JSON schema RFC to describe the JSON structures. Alternatively, they could also be described using plain text and examples.

10 Considerations for implementing CIM-RS

CIM-RS is implemented in two places: a centralized server and many clients. The server provides access to CIM objects, and the client accesses those objects. One of the goals of REST is to enable clients, such as generic HTTP browsers, to discover and access RESTful services without specialized documentation or programming. CIM-RS enables this kind of access, but, realistically, such usage would be too granular and awkward for most tasks. More likely, CIM-RS will be used in the background as a Web service that performs operations and collects data on IT infrastructure. The code that combines individual REST requests into task-oriented applications can be implemented either on the server side or on the client side.

On the server side, SOAP implementations respond to SOAP calls that are usually transported by HTTP as a layer under the SOAP stack. The RESTful stack is less elaborate because the layer corresponding to the SOAP calls is eliminated and calls are received directly from the HTTP server. Correspondingly, on the client, in SOAP implementations calls are made through the SOAP stack and transported by HTTP. In REST, calls are made using native HTTP verbs. REST simplicity comes with a price. The SOAP stack and the additional specifications that have been written over SOAP add rich functionality that may require extra effort to implement the equivalent in REST.

With the addition of CIM-RS, applications based on objects defined using CIM metadata can be surfaced through the CIM-RS RESTful protocol. The choice of protocol affects both the server implementation and the client implementation. In theory, the applications that result should be the same, but in practice there may be differences, based on factors such as the statelessness of REST and the ease of implementing some interaction patterns.

Many implementations are expected to involve using CIM-RS with existing implementations. The ease of these implementations will largely depend on the layering of the architecture of the CIM implementation. Ideally, the implementation of the CIM objects should be crisply separated from the transport mechanism. In that case, the CIM-RS implementation, using appropriate frameworks for interfacing underlying code with HTTP, such as JAX-RS, should be straight forward and relatively quick to implement.

Every implementation decision is based on many factors, including:

- The experience of the personnel involved. A group accustomed to RESTful applications will be better prepared to work with CIM-RS than a SOAP-based implementation. A group not familiar with REST may experience difficulty.
- The environment. For example, an implementation behind a corporate firewall will not realize as many advantages from a REST implementation as an implementation that spans widely separated architectures involving many firewalls.
- The purpose of the implementation. Some implementations will involve management of massive storms of events. Others will involve long lists of managed objects. Yet others will involve only light traffic but complex control operations. Every implementation has its own footprint. The REST architecture is designed to optimize the capacity, scalability, and upgradability of the server. The archetypical REST implementation is a server that serves an enormous number of clients, for example, a Web storefront serving hundreds of thousands of clients simultaneously, but with data exchange for each client that is intermittent, granular, and relatively small. This is far different from an enterprise IT management application that manages and correlates data from hundreds of thousands of objects, but only has a handful of clients. RESTful interfaces have proven themselves in the first example, but they have not yet acquired a long track record

in the second example. This is not to say that REST, and CIM-RS in particular, is not appropriate for the second example, only that it may present new challenges.

CIM-RS provides an alternative to SOAP-based implementations and allows implementers to take advantage of the unique characteristics of REST. The decision to use CIM-RS should be made in the full context of the experience of the implementers, the environment, and the purpose of the implementation.

11 Conclusion

CIM-RS is not a complete standard. The specifications produced by the CIM-RS Incubator are works in progress and published as informational documents. Nevertheless, the goal of the project is to provide a rigorous and generalized REST interface to resources modeled following the principles of the CIM model. The immediate and obvious benefit of achieving this goal is to provide REST access to management instrumentation based on the more than 1,400 pre-existing classes in the DMTF CIM Schema and in DMTF management profiles.

The development of a REST interface addresses an important issue in the industry: RESTful interfaces have become an interface of choice for application interaction over the Internet. With rising interest in cloud computing, which largely depends on Internet communications, the importance of REST interfaces is also rising. Consequently, a protocol that promises to give existing applications a RESTful interface with minimal investment is extremely attractive.

CIM-RS provides more than an additional interface to existing CIM-based implementations. The CIM model is a general object-oriented modeling approach and can be applied to many modeling challenges. Thus, for any applications built using the CIM meta-schema, not just tools based on CIM itself, CIM-RS specifies a standards-based RESTful interface that will increase interoperability. Developers can use the CIM-RS specifications as the basis for a design pattern and avoid reinventing a RESTful API for each application, saving time and effort and minimizing testing.

CIM-RS has the potential to become a basic pattern for application communication within the enterprise, between enterprises, and within the cloud. It applies to existing implementations of CIM objects, future CIM object implementations, and implementations of new objects modeled following the CIM model.

ANNEX A

Change Log

Version	Date	Description
1.0.0	2011-10-26	Released as a white paper

Bibliography

Documents published by standards development organizations

DMTF DSP0004, CIM Infrastructure Specification 2.6,
http://www.dmtf.org/standards/published_documents/DSP0004_2.6.pdf

DMTF DSP0201, *Representation of CIM in XML 2.3,*
http://www.dmtf.org/standards/published_documents/DSP0201_2.3.pdf

DMTF DSP0223, *Generic Operations 1.0,*
http://www.dmtf.org/standards/published_documents/DSP0223_1.0.pdf

DMTF DSP0230, *WS-CIM Mapping Specification 1.1,*
http://www.dmtf.org/standards/published_documents/DSP0230_1.1.pdf

DMTF DSP-IS0201, *CIM Operations Over RESTful Services 1.0*,
http://www.dmtf.org/standards/published_documents/DSP-IS0201_1.0.pdf

DMTF DSP-IS0202, *CIM-RS Binding to JSON 1.0*,
http://www.dmtf.org/standards/published_documents/DSP-IS0202_1.0.pdf

DMTF Technologies Diagram, http://www.dmtf.org/standards/stackmap

ECMA-262, *ECMAScript Language Specification, 5th Edition*, December 2009,
http://www.ecma-international.org/publications/standards/Ecma-262.htm

IETF RFC2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, November 1996, http://tools.ietf.org/html/rfc2045

IETF RFC2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, November 1996,
http://tools.ietf.org/html/rfc2046

IETF *RFC2141, URN Syntax*, May 1997
http://tools.ietf.org/html/rfc2141

IETF RFC2616, *Hypertext Transfer Protocol – HTTP/1.1*, June 1999,
http://tools.ietf.org/html/rfc2616

IETF RFC3986, *Uniform Resource Identifier (URI): Generic Syntax*, January 2005,
http://tools.ietf.org/html/rfc3986

IETF RFC3987*, Internationalized Resource Identifiers (IRIs)*, January 2005,
http://tools.ietf.org/html/rfc3987

IETF RFC4287, *The Atom Syndication Format*, December 2005,
http://tools.ietf.org/html/rfc4287

IETF RFC5023, *The Atom Publishing Protocol*, October 2007,
http://tools.ietf.org/html/rfc5023

IETF RFC5789, *PATCH Method for HTTP*, March 2010,
http://tools.ietf.org/html/rfc5789

IETF RFC-json-schema (Internet-Draft), *A JSON Media Type for Describing the Structure and Meaning of JSON Documents*, March 2010, http://tools.ietf.org/html/draft-zyp-json-schema-02
DSP-IS0202 uses a draft of the upcoming draft version 03, available at http://groups.google.com/group/json-schema/browse_thread/thread/5dc7d56a18abd81b

ISO/IEC 10646:2003, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*, http://standards.iso.org/ittf/PubliclyAvailableStandards/c039921_ISO_IEC_10646_2003(E).zip

The Unicode Consortium, *The Unicode Standard, Version 5.2.0, Annex #15: Unicode Normalization Forms*, http://www.unicode.org/reports/tr15/

Other documents

R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, PhD thesis, University of California, Irvine, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

R. Fielding, *REST APIs must be hypertext driven*, October 2008, http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

J. Holzer, *RESTful Web Services and JSON for WBEM Operations*, Master thesis, University of Applied Sciences, Konstanz, Germany, June 2009, http://mond.htwg-konstanz.de/Abschlussarbeiten/Details.aspx?id=1120

A. Manes, *REST principle: Separation of Representation and Resource*, March 2009, http://apsblog.burtongroup.com/2009/03/rest-principle-separation-of-representation-and-resource.html

L. Richardson and S. Ruby, *RESTful Web Services*, May 2007, O'Reilly, ISBN 978-0-596-52926-0, http://www.oreilly.de/catalog/9780596529260/