The HP Universal CMDB SPARQL Adapter

Steve Battle, David Booth

HP Laboratories

HPL-2009-349

Keyword(s):
CMDB, SPARQL, ITSM, ITILv3, CMS, RDF, OWL, JENA

Abstract:

IT Service Management (ITSM) is about making IT accountable to business. IT and business functions meet at the service interface where business functions define themselves in terms of the services they deliver to their customers. The IT function, in turn, must provide the infrastructural capabilities and resources necessary to support it. The IT Infrastructure Library (ITIL) proposes that the IT configuration should be explicitly modeled within a Configuration Management System (CMS). The HP Universal Configuration Management Database (UCMDB) is one component of a CMS; maintaining a comprehensive and up-to-date snapshot of all managed assets and their inter-relationships across the IT environment. The UCMDB is not particularly web-friendly; there is no easy way to access configuration data using a conventional browser. Another drawback is that configuration records do not correspond directly to the language of business. This report addresses the first of these issues; making the IT configuration navigable on the web.

External Posting Date: November 6, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: November 6, 2009 [Fulltext]

Presented at HP Software Universe 2007.

The HP Universal CMDB SPARQL Adapter[*]

Steve Battle
HP Labs, Bristol
steven.a.battle@googlemail.com

David Booth
http://www.dbooth.org/
david@dbooth.org

1 IT Service Management

IT Service Management (ITSM) is about making IT accountable to business. IT and business functions meet at the service interface where business functions define themselves in terms of the services they deliver to their customers. The IT function, in turn, must provide the infrastructural capabilities and resources necessary to support it. Service management is defined [14] as "a set of specialized organization capabilities for providing value to customers in the form of a service."

The aim is to align an organization's IT infrastructure with the business services it provides, so that the running of any server, network, or database is justified by its support for these services. This is not merely a paper excercise. The IT Infrastructure Library (ITIL) [9] proposes that the IT configuration should be explicitly modeled within a Configuration Management System (CMS). The CMS is the place where the enterprise reflects upon itself; recalling past configurations, displays an awareness of its present state, and plans for future change.

The HP Universal Configuration Management Database (UCMDB) is one component of a CMS; maintaining a comprehensive and up-to-date snapshot of all managed assets and their inter-relationships across the IT environment. It maintains this model using real-time discovery and asset tracking. The failure of any one of these assets would be spotted, and the impact of this failure on service delivery could be immediately assessed.

The UCMDB is not particularly web-friendly; there is no easy way to access configuration data using a conventional browser. Another drawback is that configuration records do not correspond directly to the language of business. We are beginning to see the emergence/convergence of a number of business ontologies including ITILv3 and COBIT that enable business users to describe services in terms of customer value and outcomes, rather than the technical language of web-services. It will become increasingly important for users to be able to engage with the CMS in their own language rather

[*]This work was presented at HP Software Universe 2007 [1].

than the language of IT. This report addresses only the first of these issues; making the IT configuration navigable on the web.

This work builds on the Joseki SPARQL server, the ARQ query engine, the Jena RDF framework, the Mercury Application Mapper (MAM) 6.5 functional demo, and the Mercury UCMDB SOAP API.

2 Configuration data

Service assets come in two varieties. *Resources* include commoditities such as financial capital, information, infrastructure including IT and facilities, software applications, and people. *Capabilities* are an organization's ability to manage resources to their competitive advantage and include softer assets such as management, organization, processes, knowledge and people. This wide ranging (and overlapping) set of interests provides a rich problem domain for enterprise knowledge management. Each asset, or *configuration item*, has a set of attributes that depend on the CI involved and the information required to manage the service. Some attributes are specific to a single CI type, whereas others are common to all configuration items. In addition to these attributes a CI will also be linked by relationships to other configuration items including changes, incidents, and known problems.

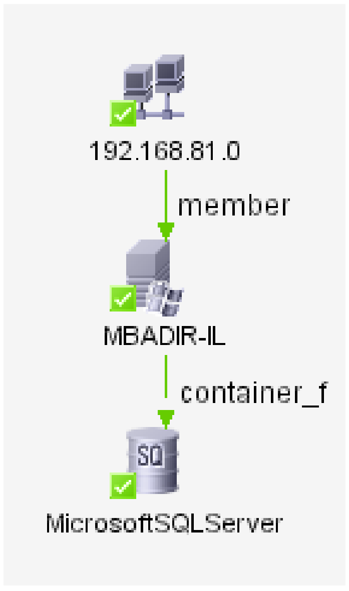

Figure 1: Topological view of configuration data showing a server (middle) as member of a network (top) hosting database (bottom).

So what does configuration data actually look like? The Mercury Application Mapper provides a topological view of configuration data as illustrated in figure 1. This shows a server (middle), which is a member of a network (top), hosting a database (bottom). This configuration is viewed as a simple graph where each node represents a so-called *configuration item* (CI). Each CI has a 128 bit ID that uniquely identifies it within the CMS. CIs are also associated with a display label which is used to name each CI in the graph of figure 1.

3 The UCMDB Query Interface

The HP UCMDB offers a sophisticated set of tools for monitoring and querying configuration records. Users working with the Mercury Application Mapping tool are able to create graphical topological queries that express the configuration topologies they are interested in. TQL is primarily a graphical language; The XML serialization used at the SOAP API was not developed as a stand-alone query language, so is not particularly human readable. This includes the Topological Query Language (TQL) that allows end-users to express queries graphically. Configuration data is viewed, not in terms of relations, but as a user configurable graph. A TQL query is a graphical template that enables us to search the configuration graph for all subgraph instances of a given pattern graph. The graphical query illustrated in figure 2 represents a graph pattern that will match the configuration data of figure 1. Each node of this query is a CI type that will match only instances of that type.

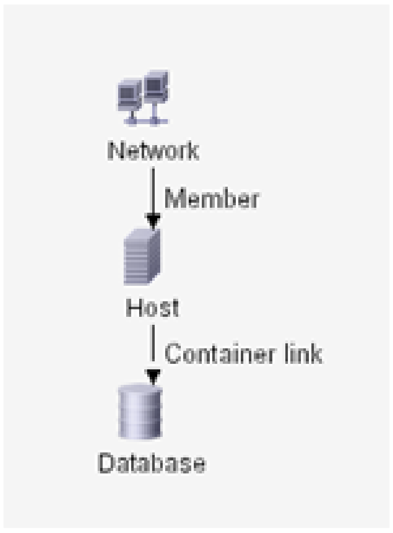

Figure 2: Topological query selecting simple network, host, server configurations (screenshot from MAM TQL builder).

The UCMDB presents a comprehensive SOAP API that enables it to be programmatically queried and updated from a client language such as Java. In this report we consider only the query aspects of this interface. This API is described in WSDL (Web Services Description Language) detailing the operations available at the interface. To support the use of the SOAP API from Java, we use Apache Axis to generate the appropriate framework classes and stub code directly from the WSDL. The UCMDB API supports a wide variety of operations, but we are primarily intersted in executing a topological query (TQL) expressed in XML.

4 Configuration metadata

One man's data is another's *metadata*. The data/metadata distinction is not an absolute but a relative distinction. The UCMDB doesn't provide access to the data in a database but instead provides metadata *about* the database. The W3C has established frameworks for handling metadata that are a good fit for the graphical model we see in the UCMDB. According to Sir Tim Berners-Lee [4] - inventor of the world-wide web - “The Semantic Web is a web of data, in some ways like a global database.” In the case of the UCMDB, this data is not published globally, but could be thought of as part of a Semantic IntraWeb.

The semantic web is a new take on the web where online content can be read & understood not only by humans using browsers, but also by software agents. The original vision of the web as a vast global hyper-text is largely opaque to machine understanding. The semantic web builds on this vision, first and foremost, by adding labels to the links that capture the kind of relationship that exists between a pairs of resources; transforming the web into a global knowledge-base. At the heart of this vision are W3C standards: the Resource Description Framework (RDF) [11], the Web Ontology Language (OWL) [2], and the SPARQL Protocol and RDF Query Language (SPARQL) [15]. However, it is not the use of any particular language that is key to understanding the semantic web. The semantic web provides a *logical* (model theoretic) semantics for the web which serves as the foundation for model-driven activities drawing on web data.

The Jena Semantic Web toolkit developed in HP Labs is a comprehensive Java based platform for manipulating RDF graphs and their associated ontology, as well as providing sophisticated query and inference capabilities. Jena is available under a liberal BSD-style license. Jena is also embedded in a number of tools including the popular Protégé [7] and TopBraid Composer [16] ontology editors, the Metatomix MetaStudio semantic development environment for eclipse [13], and most recently EulerGUI, ”a lightweight IDE for Artificial Intelligence” [17].

The semantic web technology stack is often visualized as a layer-cake, spanning XML, metadata, query, ontology, and rules. We analyse the issues of mapping configuration data into the semantic web by climbing the stack from it’s XML foundations upwards.

4.1 XML foundations

One of the key concepts that XML brings to RDF is the concept of the *namespace* providing a context in which local names are defined. When we need to introduce different naming contexts such as ITIL and COBIT, it becomes essential to have a mechanism for disambiguating different terminologies. The UCMDB itself defines a naming context which will be used in conjunction with the Configuration Item ID, a unique numeric identifier, to define a URI for each named resource. The UCMDB namespace is arbitrary (but should be stable) and for our experiments is defined as follows:

$$ci \quad = \quad \langle http://cmdb.mercury.com\#\rangle$$

For ease of distinguishing the namespace from the *fragement identifier* postfix, the namespace is terminated with the '#' character. The CI ID is prefixed with a label representing it's unique primary type name. For example, the full URI for a CI representing a Windows NT server would be as follows:

$$\langle http://cmdb.mercury.com\#nt.35014541\rangle$$

In future this URI mapping may help to resolve the identity of configuration items across multiple, federated CMDB. This contrasts with the current approach that designates a single UCMDB as the master system.

The use of an HTTP reference in metadata does not imply that the URI is dereferenceable. However, depending upon the underlying architecture we could opt to serve representations of individually addressed resources, or return HTTP response code 303 (See Other) to indicate explicitly that there is no representation for the accessed URI [10].

The names of object types, attributes, and inter-object links are similarly mapped onto named URIs. To avoid name clashes they are defined in three separate, but related, namespaces:

$$\begin{aligned} object &= \langle http://cmdb.mercury.com/object\#\rangle \\ attr &= \langle http://cmdb.mercury.com/attribute\#\rangle \\ link &= \langle http://cmdb.mercury.com/link\#\rangle \end{aligned}$$

Another XML foundation that RDF builds on is the use of XML schema datatypes. The primitive dataypes used in the UCMDB: Boolean, Byte, Date, Double, Float, Integer, Long, String are mapped into the XML Schema datatypes: xs:boolean, xs:base64Binary, xs:date, xs:double, xs:float, xs:integer, xs:long, xs:string respectively, where the 'xs' prefix is defined as follows:

$$xs = \langle http://www.w3.org/2001/XMLSchema\#\rangle$$

The UCMDB datatype for XML data is mapped to the new RDF datatype rdf:XMLLiteral. Attributes may also be lists of primitive types. Additional structured datatypes for String & Integer Lists may be represented by RDF sequences. The label associated with the windows server referred to above is the string typed value:

$$\text{“MBADIR-IL”}\hat{}\hat{}\langle http://www.w3.org/2001/XMLSchema\#string\rangle$$

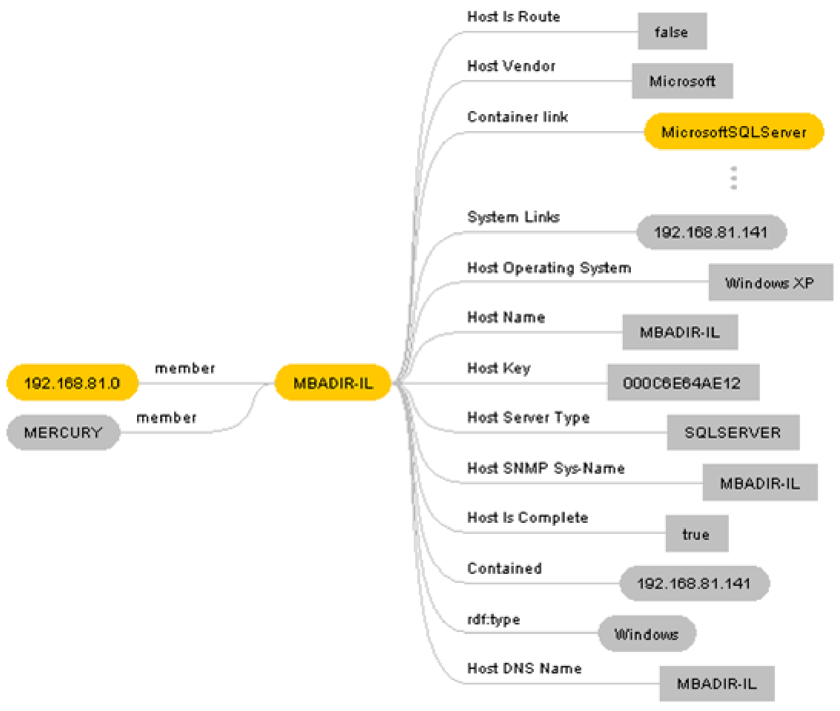

Figure 3: Resource description of Windows host 'MBADIR-IL'.

4.2 Metadata: a Resource Description Framework

RDF is not the syntax. Neither is it the way the metadata is stored. RDF provides standard ways to disseminate metadata with standard serializations including RDF/XML [6] and N3 (Notation 3) [3] preferred by many for its readability. However, the subtlety of RDF lies in the logical model underlying the metadata, best imagined as a graph. It may be stored directly as RDF/XML or in an RDF triple store, but we may also create wrappers around existing data sources like the UCMDB. RDF allows us to pull together a description of a resource, from possibly many different data sources, into a single model.

An RDF *description* is a summary of everything we know about a given resource. At its simplest, this description will comprise a set of name & primitive value pairs. An RDF graph contains many resource descriptions and is composed of triples, comprising the resource being described, a property, and the object of the statement. Many of these properties will be so-called datatype properties, with a simple typed value. For example, we saw above that a CI representing a host server has a string typed label. The name of this property is itself named by a URI, *attr:host_dnsname*. Other object properties reference other CIs. The 'MBADIR-IL' server is host to (*link:container_f*) the Microsoft SQLServer database. In the N3 notation we may list the different property/values belonging to the same resource description with the semicolon separator. The description may also include the subject type, here *object:nt*. In N3 the keyword 'a' is simply shorthand for *rdf:type*. The resource description is terminated with a full-stop.

$$\begin{array}{ll} ci:nt.35014541 & \text{a } object:nt; \\ & attr:host\_dnsname \text{ ``MBADIR-IL''}\hat{}\hat{}xs:string; \\ & link:container\_f \; ci:sqlserver.77593429. \end{array}$$

This information and more is captured in the graphical representation of a resource description shown in figure 3. Literal datatyped values are shown as rectangular boxes, while CIs are rounded boxes. Instances of *Network*, *Host*, and *Database* are highlighted in orange. Each graph edge corresponds to a triple read from left to right.

4.3 Ontology

The UCMDB defines a type hierarchy, including the types 'network', 'host' & 'database' which are themselves configuration items, meaning that new types can be added. A CI must be assigned a single type from this hierarchy. The type defines the *attributes* a CI may have; for example the host server has an attribute, *host_os*, that describes the type of operating system and would have a simple string value, such as "Linux".

The OWL Web Ontology Language is the W3C standard for defining the classes and properties that can be used within RDF. They are a shared, formal, conceptualization of an application domain. Ontologies encourage users to create and re-use common models, enhancing the value of shared metadata. OWL builds on RDF schema (RDFS) [5], used to declare its vocabulary, adding additional terminology to define the logical conditions for class membership.

For the UCMDB the ontology defines the objects and relationships in the application universe, defining CI classes and link taxonomies. These are configured within the UCMDB and may be automatically converted into OWL or RDFS representations for use with standard semantic web tools. The ontology will assert the subclass relationship between a Windows NT server *object:nt* and the more generic concept of an object:host. This information enables a user to interactively build a query by supporting a drop-down list (see figure 4) of the available classes and relationships that may be found in the data. This information is presented hierarchically; to locate the *Host* class as required in our query, the user would begin at the *Root* type, and drill down through *Data*, *Object*, *IT Universe*, and *System*.

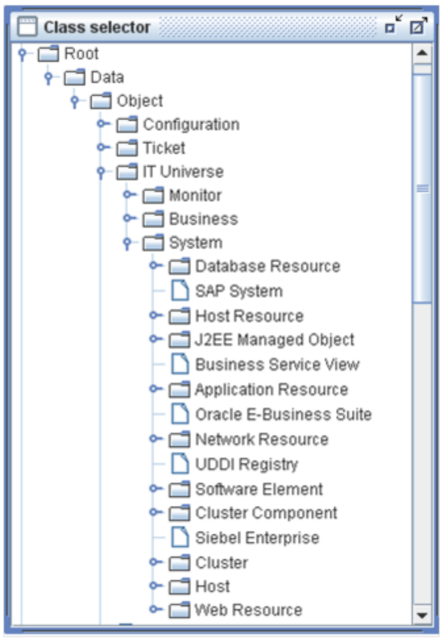

Figure 4: Snapshot of UCMDB taxonomy.

Note that these names are based on user-friendly class labels rather than the formal class URI.

$$\begin{array}{rll} object : data & rdfs : subClassOf & object : root. \\ object : object & rdfs : subClassOf & object : data. \\ object : it\_world & rdfs : subClassOf & object : object. \\ object : system & rdfs : subClassOf & object : it\_world. \\ object : host & rdfs : subClassOf & object : system. \\ object : nt & rdfs : subClassOf & object : host. \end{array}$$

The links are directed, binary relationships between a pair of CIs. The domain and range of such a link may each be restricted to specific types. As we saw above, an NT server may host (be a container for) services such as databases and other host resources such as disks and cpus. This is represented in OWL as a class restriction on a specific property in the context of a given class. This information may be used by query builders to provide contextual support for class selection, in this case offering only host resources (and its subclasses) as options for the object of the containment relationship when querying a host.

$$\begin{array}{ll} object:nt & a\ owl:Class;\ rdfs:subClassOf \\ & [\ a\ owl:Restriction; \\ & owl:allValuesFrom\ object:hostresource; \\ & owl:onProperty\ link:container\_f\ ]. \end{array}$$

4.4 Query

With the underlying configuration data reflected as RDF and described with OWL, we need a mechanism to replicate the query capabilities of TQL. We are investigating the use of SPARQL (SPARQL Protocol And RDF Query Language) as an alternative way to access these configuration records. SPARQL is the W3C recommended metadata query language, providing a powerful SQL-like syntax for RDF queries. Unlike SQL, the graph patterns used in a SPARQL query are understood in terms of the underlying, triple-oriented, RDF model rather than the n-ary relational model. The results of a query may be returned in one of the standard RDF serializations, including RDF/XML and N3. For web-services it is possible to return data bindings using the JSON result serialization.

The written form of SPARQL will be superficially familiar to SQL developers, making it easy for them to make the switch. However, given this similarity, we consider later why we do not use SQL from the get-go. The SPARQL query below represents the same search pattern as the TQL query shown earlier in figure 2. Variables, prefixed by '?', are bound to resources in the matching graph, generating a *result set* that lists all the matching combinations in the queried graph.

```
PREFIX object:   <http://cmdb.mercury.com/object#>;
PREFIX link:     <http://cmdb.mercury.com/link#>;

SELECT ?network ?host ?database
WHERE {
  ?network a object:network; link:member ?host.
  ?host a object:host; link:container_f ?database.
  ?database a object:database.
}
```

There are essentially two steps in the process of presenting a SPARQL query to the UCMDB. The first step is to translate the SPARQL query into the XML form of TQL. SPARQL queries may contain complex nested patterns which would be difficult to translate directly into TQL. The ARQ query engine handles the basic parsing and processing of queries, breaking the problem down into separate stages each of which represents a basic graph pattern. These stages are then composed into a pipeline with the output of each stage feeding into the input of the next. The queries generated at each stage are presented to the UCMDB which returns a sub-graph containing all the resources that match the query. This sub-graph may contain multiple solutions to the original query, and the ARQ query engine contains the logic to iterate through and recombine the results from the different query stages. Looking at the example above, the resulting sub-graph may contain multiple network, host and database objects from which all possible binding combinations for ?network, ?host & ?database are returned.

Once the query has been processed the XML graphs returned by the UCMDB must be translated into standard RDF before being passed back to the query engine. At this point there is a discrepancy between the classes expressed in the query and those in the data. The query requests a host object of type *object:host* whereas the graph returned from the UCMDB contains objects of type *object:nt*. The UCMDB is smart enough to identify these Windows NT servers as kinds of Host (*object:nt* is a sub-class of *object:host*), but this inference remains implicit in the returned graph. To enable the ARQ query engine to make use of this inference it must be made explicit in the graph. This requires the use of Jena inference, described in the next section.

4.5 Rules

Jena includes a general purpose rule-based reasoner which is used to implement both the RDFS and OWL reasoners but is also available for general use. This reasoner supports rule-based inference over RDF graphs and supports forward chaining, backward chaining or a hybrid execution model combining the two. There are two rule engines; a forward chaining Rete [8] engine and a tabled datalog engine. They can be used independently, or the forward chaining engine can be used to prime the backward chaining engine which in turn can be used to answer queries.

Rather than use the generic OWL reasoner, we make use of the general-purpose forward chaining reasoner. This allows us to fine-tune the rule-base and optmize performance. The input to the inference engine combines the sub-graph returned by the UCMDB with ontology generated earlier. This provides us with two key facts:

$$\begin{array}{rll} ci:nt.35014541 & rdf:type & object:nt. \\ object:nt & rdfs:subClassOf & object:host. \end{array}$$

We are interested in hierarchical closure over classes and properties as they apply to instance data. These rules are a small subset of the RDFS closure rules. We have two rules (*rdfs6* & *rdfs9*) each of which has an antecedent (if-part) and a consequent (then-part) on the left and right-hand sides of the implication operator '→'. Each logical term is a triple enclosed in brackets, any may contain variables (prefixed with '?') local to the clause they appear in.

```
# RDFS Closure rules
[rdfs6:  (?a ?p ?b), (?p rdfs:subPropertyOf ?q) -> (?a ?q ?b)]
[rdfs9:  (?a rdf:type ?x), (?x rdfs:subClassOf ?y) -> (?a rdf:type ?y)]
```

Using the rule *rdfs9* with ?a, ?x, and ?y bound to *ci:nt.35014541*, *object:nt*, and *object:host* respectively, we may infer the additional triple:

$$ci : nt.35014541 \quad rdf : type \quad object : host$$

With this information the ARQ query engine is now able to bind the Windows NT host *ci:nt.35014541* to the *?host* variable in the query.

4.5.1 SPARQL vs. SQL

Why isn't SQL the language of choice rather than SPARQL. Jim Melton of Oracle puts it succinctly [12], "Because typical questions asked of RDF involve several, sometimes many, join operations, SPARQL provides a more compact notation that is perhaps easier to get right with less debugging time spent." What does this mean in practice? The RDF model is, fundamentally, relational. Yet it is a relational model that is normalised to the extent that it captures only binary relations (ternary if you count language attributes on literals).

The following SPARQL query is designed to select only networked Linux servers with a database. It demonstrates how we may specify simple conditions on object attributes. This is modelled as a pair of links between a network object and a host object *link:member*, and between the host object and a database object *link:container_f*. The example also includes datatype attributes of these configuration items *attr:host_os, attr:display_label*. These links can be modeled as a pair of binary relations, whereas the classifications *object:network*, *object:host*, *object:database* are thought of logically as unary relations.

```
PREFIX object:   <http://cmdb.mercury.com/object#>
PREFIX link:     <http://cmdb.mercury.com/link#>
PREFIX attr:     <http://cmdb.mercury.com/attribute#>
PREFIX xs:       <http://www.w3.org/2001/XMLSchema#>

SELECT ?hostname
WHERE {
  ?network a object:network; link:member ?host.
  ?host a object:host; link:container_f ?database;
  attr:host_os "Linux"^^xs:string; attr:display_label ?hostname.
  ?database a object:database.
}
```

We will compare this SPARQL query with an equivalent SQL query. It should be made clear that this comparison is made in the context of data already modeled as a highly normalized graph. The nodes and links in this graph may be hierarchically classified which makes the problem even harder than it appears at face value. The properties that define the links can be sub-properties of other properties, and the classes can be sub-classes of other classes. Relational hierarchies are not native to relational databases, so to avoid masive duplication of data, this hierarchy must be explicitly represented some way. In the Common Data Model (CDM) used by the UCMDB, both the *member* and *container_f* links are sub-properties of a common, top-level *link* property. The super-relation comprising all the *member* and *container_f* property instances are recorded just once in the *link* relation. Individual link instances are then identified and classified according to which property relation they belong to using *CDM_MEMBER_1* and *CDM_CONTAINER_F_1*.

The database has a single table for each configuration item class *CDM_NETWORK_1*, *CDM_HOST_1*, *CDM_DATABASE_1*. However, these are not unary relations. Attributes are associated with classes and are defined as additional columns in these tables. The example is concerned with two attributes of the host server; the *host_os* is associated with the host specifically as a 'host', while the *display_label* is associated with the host as a generic 'data' item *CDM_DATA_1*, the superclass to all CDM classes. These must be joined on the host ID to ensure that the attribute values pertain to the same configuration item.

Neither does the relational model or SQL provide support for namespaces. SPARQL is a web technology so is built on a foundation of URI naming. This ranges from the names of classes, properties, and datatypes to the identifiers used for individual configuration items. SPARQL simplifies namespace management by allowing namespace prefix definitions. In SQL we can resort to tricks like adding *CDM_* to table names, but in a federated, distributed world we have to move to a new mindset where we have globally scoped identifiers instead of local IDs.

The SQL query below performs an equivalent query against the database underlying the UCMDB. It is described as a sequence of inner joins, together with a single 'WHERE' clause selecting only hosts running the 'Linux' operating system. By sacrificing n-ary relations, SPARQL achieves simplicity by dispensing with column names. The subject, predicate and object of a relation are identified positionally rather than by name (e.g. END1_ID, END2_ID). One benefit of this approach is that things that only appear once, like the network, the host and database, may remain anonymous. Not so in SQL, where the explicit join requires everything to be named.

In addition, the joins are explicit (...INNER JOIN... ON...). As can be seen below, the overall sense of the query is obscured by the number of join operations. At best we could remove 'INNER JOIN's, moving the join conditions into the 'WHERE' clause but this would have the additional disadvantage of splitting individual graph edges across the 'FROM' (containing the predicates) and 'WHERE' clauses containing the subjects, and objects.

```
SELECT CDM_DATA_1.A_DISPLAY_LABEL AS hostname
FROM
CDM_MEMBER_1
INNER JOIN CDM_LINK_1 AS member
ON CDM_MEMBER_1.CMDB_ID = member.CMDB_ID
INNER JOIN CDM_CONTAINER_F_1
INNER JOIN CDM_LINK_1 AS container_f
ON CDM_CONTAINER_F_1.CMDB_ID = container_f.CMDB_ID
ON member.END2_ID = container_f.END1_ID
INNER JOIN CDM_NETWORK_1
ON member.END1_ID = CDM_NETWORK_1.CMDB_ID
INNER JOIN CDM_HOST_1
ON member.END2_ID = CDM_HOST_1.CMDB_ID
INNER JOIN CDM_DATABASE_1
ON container_f.END2_ID = CDM_DATABASE_1.CMDB_ID
INNER JOIN CDM_DATA_1
ON member.END2_ID = CDM_DATA_1.CMDB_ID
WHERE (CDM_HOST_1.A_HOST_OS = 'Linux')
```

Because of these differences, we contend that it is much easier to see the SPARQL query as a graph, albeit flattened into the serial confines of a written language. Another difference we observe is that in the relational model, the data type of an attribute is implicitly associated with the column, as defined by the database schema. In the SQL query, 'Linux' can only be a string because of the type of the 'A_HOST_OS' attribute. In SPARQL this is not the case; an attribute or link could have a range of valid types. Consequently, we have to be explicit about the datatype, declaring it to be an XML schema string.

Perhaps more significantly, SQL is primarily suited to tabular data, returning only variable bindings over structured relations. SPARQL, however, is desgned not only to match graph patterns, but to return complete graphs in the results (*describe* queries), even when that graph structure is not expressed in the query.

5 SPARQL vs. TQL

SPARQL is at least as expressive as the graphical TQL. The SPARQL queries shown so far have demonstrated support for basic topological query over nodes & links, and simple conditions on object attributes. TQL includes additional constructs for querying link attributes, simple, disjunctive & negative attribute conditions, minimum & maximum cardinality constraints, and qualifier conditions. We show how these constructs may be represented in RDF & SPARQL.

5.1 Link attributes

Links between CIs are themselves a kind of CI, are defined in the same type hierarchy as other CIs, and may have attributes of their own. This is similar to the UML technique of defining an *association class* for complex n-ary relations. For example, the membership link between the network and host server carries additional information about the actual network address, recorded as the *member_netaddress* attribute on the link itself. Ordinarily, in RDF, the relationship between a pair of resources may not have properties of its own. However, RDF includes the idea of *reification* where an abstract relationship may be given concrete form and treated as a resource in its own right.

```
PREFIX object:   <http://cmdb.mercury.com/object#>;
PREFIX link:     <http://cmdb.mercury.com/link#>;
PREFIX attr:     <http://cmdb.mercury.com/attribute#>;
PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>;

SELECT ?hostname ?ip
WHERE {
  ?network a object:network; link:member ?host.
  ?host a object:unix; attr:display_label ?hostname.
  ?member attr:member_netaddress ?ip.
  # relationship reified as ?member
  ?member rdf:subject ?network;
   rdf:predicate link:member; rdf:object ?host.
}
```

In the query above, the reified relationship between network and host, bound to *?member*, is related to its respective subject, predicate & object by the *rdf:subject, rdf:predicate, & rdf:object* relationships (the presence of these relationships signals the process that lifts the UCMDB data into RDF to assert these reifications). As can be seen in the query we may use this reified link to access the *attr:member_netaddress* property.

5.2 Conditional filters

The semantic web exhibits what is known as an open-world semantics; the absence of a fact doesn't prove the negation. What it shares with the UCMDB is that there is no way to express a negative fact in the configuration data. Nevertheless, conditional filters provide a limited form of negation allowing users to express disjunctive and negative conditions on attribute values. For example, the query below filters for either "Linux" or "Windows XP" servers.

```
PREFIX attr: <http://cmdb.mercury.com/attribute#>
PREFIX object: <http://cmdb.mercury.com/object#>
PREFIX link: <http://cmdb.mercury.com/link#>

SELECT * WHERE {
  ?network a object:network; link:member ?host.
  ?host a object:host; attr:host_os ?os; link:container_f ?database.
  ?database a object:database.
  FILTER (str(?os)="Linux" || str(?os)="Windows XP")
}
```

We could adapt this example to select anything but "Linux" servers by replacing the filter condition with "FILTER afn:not(str(?os)="Linux")". This uses SPARQL's extensibility merchanisms to allow new conditional filters to be added. This extension is defined in the namespace $afn = \langle java : com.hp.hpl.jena.sparql.function.library.\rangle$

5.3 Cardinality constraints

TQL provides the facility to place conditions on the number of times a particular relationship occurs. In this case we need to provide a custom filter to support this behaviour. The *cardinality* function is a custom extension designed to mimic the functionality found in TQL. The example below demonstrates the use of a minimum cardinality constraint to search for multi-processor unix hosts. It counts the number of *container_f* links between the host and any CPU (a kind of host resource).

```
PREFIX attr: <http://cmdb.mercury.com/attribute#>
PREFIX object: <http://cmdb.mercury.com/object#>
PREFIX link: <http://cmdb.mercury.com/link#>
PREFIX fun: <java:com.hp.hpl.cmdb.sparql.function.>

SELECT DISTINCT ?host ?cpu
WHERE {
  ?host a object:unix; link:container_f ?cpu. ?cpu a object:cpu.
  FILTER (fun:cardinality(?host, link:container_f, object:cpu) > 1).
}
```

5.4 Qualifier conditions

The simple UCMDB class taxonomy is augmented with the idea of *qualifiers* which are non-inheritable properties of types. To give a familiar example from programming, a given class may be qualified as an 'abstract class which wouldn't be inherited by its (concrete) sub-classes. However, where these qualifiers tend to be fixed in programming languages, the UCMDB is extensible, allowing a customer to add domain specific qualifiers.

In the example below, we select for sub-classes of *object:host* that are qualified as network devices. This would select for routers, load-balancers, firewalls, and networked printers, which are all classified as hosts. It does not match servers which are not suitably qualified. Qualifiers augment the UCMDB taxonomy providing it with a limited form of multiple classification. If we were to re-engineer the ontology from the ground up we might well model this as a fully-fledged OWL ontology. However, to maintain compatibility with the UCMDB and the non-inheritability of these qualifiers they are captured as a meta-model on top of the existing class hierarchy. It is the classes themselves that are appropriately classified, hence the test below for a class that is itself an instance of a qualifier.

```
PREFIX object:   <http://cmdb.mercury.com/object#>;
PREFIX link:     <http://cmdb.mercury.com/link#>;
PREFIX qual:     <http://cmdb.mercury.com/qualifier#>;

SELECT ?network ?host
WHERE {
  ?network a object:network; link:member ?host.
  ?host a object:host; a [ a qual:NETWORK_DEVICES ].
}
```

6 Conclusion

In this report we have demonstrated the ability to bring existing service management tools like the UCMDB into the semantic web. We have seen how this is a suitable basis for presenting a standard and web-friendly query interface in SPARQL. Given that configuration data is intrinsically a graph-based model then SPARQL appears to be a more suitable query language than SQL. Finally, we have shown that many capabilities of the graphical TQL can be translated into SPARQL so there need be no loss of functionality in moving from TQL to SPARQL. Combined with graphical tools for building SPARQL queries and visualising the results this shows the potential for making the exploration of configuration data as easy as surfing the web.

References

[1] Steve Battle and David Booth. Integrating with the hewlett packard universal CMDB using semantic web technologies. `http://h41112.www4.hp.com/events/su2007/technical_tutorials_tmp.php?id=0184`, November 2007.

[2] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea

Stein. OWL web ontology language reference. http://www.w3.org/TR/owl-ref/, 2004.

[3] Tim Berners-Lee. Notation 3. http://www.w3.org/DesignIssues/Notation3.html, 1998.

[4] Tim Berners-Lee. Semantic web road map. http://www.w3.org/DesignIssues/Semantic.html, September 1998.

[5] Dan Brickley and R.V. Guha (eds). RDF vocabulary description language 1.0: RDF schema. http://www.w3.org/TR/rdf-schema/, 2004.

[6] Dave Beckett (ed). RDF/XML syntax specification. http://www.w3.org/TR/rdf-syntax-grammar/, 2004.

[7] Stanford Center for Biomedical Informatics Research. The Protg ontology editor and knowledge aquisition system. http://protege.stanford.edu/.

[8] Charles Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:1737, 1982.

[9] The IT Service Management Forum. *An Introductory Overview of ITIL® V3*. itSMF Ltd, 2007.

[10] W3C Technical Architecture Group. Dereferencing HTTP uris. http://www.w3.org/2001/tag/doc/httpRange-14/2007-08-31/HttpRange-14.html#associating-info-resources, August 2007.

[11] Graham Klyne and Jeremy J. Carroll (eds). Resource description framework (RDF): Concepts and abstract syntax. http://www.w3.org/TR/rdf-concepts/.

[12] Jim Melton. Sql, xquery, and sparql. http://www.nesc.ac.uk/talks/683/SQL-XQuery-and-SPARQL.pdf, 2005.

[13] Metatomix. Metatomix MetaStudio (m3t4 studio). http://www.metatomix.com/360solutions/application/.

[14] Office of Government Commerce (GB). *Service Design*. The Stationary Office, 2007.

[15] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/, January 2008.

[16] TopQuadrant. TopBraid Composer. http://www.topquadrant.com/products/TB_Composer.html.

[17] Jean-Marc Vanel. EulerGUI. http://eulergui.sourceforge.net/.