CMDB Federation (CMDBf)

Public Interim Draft

Version 0.95, 01 August 2007

Authors

Forest Carlisle, CA
Barry Day, Microsoft
Scott Donohoo, IBM
Pratul Dublish, Microsoft
Jacob Eisinger, IBM
Greg Goodman, CA
Andrew Hately, IBM
Mark Johnson, IBM (Editor)
Vincent Kowalski, BMC Software (Editor)
Yannis Labrou, Fujitsu
Kenji Morimoto, Fujitsu
David Snelling, Fujitsu
William Vambenepe, HP (Editor)
Marv Waschke, CA (Editor)
Van Wiles, BMC Software
Klaus Wurster, HP

Copyright Notice

Abstract

This specification describes the architecture and interactions for federating data repositories together to behave as a data store that satisfies the role of a Configuration Management Database (CMDB). The federation provides an aggregate view of a resource even though the data and underlying repositories are heterogeneous. A query interface is defined for external clients to access these data.

Status

This document is an initial draft still under internal review. A feedback agreement is required before the working group can accept feedback.

At some future date, the contents may be published under another name or under several new specifications, as shall be agreed by the authors and their respective corporations at that time.

Table of Contents

1. Introduction

Many organizations are striving to base IT management on a CMDB (Configuration Management Database). A CMDB contains data describing managed resources like computer systems and application software, process artifacts like incident, problem and change records, and the relationships among these entities. The contents of the CMDB should be managed by a configuration management process and serve as the foundation for other IT management processes, such as change management and availability management.

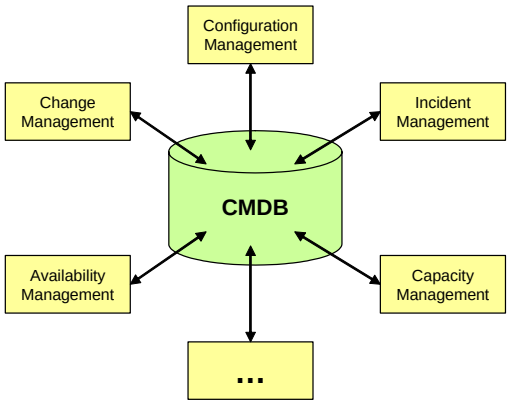


Figure 1 – Role of a CMDB

In practice this goal is challenging because the management data are scattered across repositories that are poorly integrated or coordinated.

The definition of a CMDB in the context of this specification is based on the definition described in the IT Infrastructure Library** (ITIL**): a database that tracks and records configuration items associated with the IT infrastructure and the relationships between them. Strictly speaking, the ITIL CMDB contains a record of the expected configuration of the IT environment, as authorized and controlled through the change management and configuration management processes. The federated CMDB in this specification extends this base definition to federate any management information that complies with the specification's patterns, schema, and interfaces, such as the discovered actual state in addition to the expected state. Typically, an administrator will select the data to be included by configuring the tool that implements the CMDB.

The federated CMDB described in this specification is a collection of services and data repositories that contain configuration and other data records about resources. The term 'resource' includes configuration items (e.g., a computer system, an application, or a router), process artifacts (e.g., an incident record, a change record), and relationships between configuration item(s) and/or process artifact(s). The architecture describes a logical model and does not necessarily reflect a physical manifestation.

1.1 Objectives

1.1.1 Functions

The federated CMDB resulting from using this specification will provide a single aggregate view of the data about an IT resource, even if the data is from different heterogeneous data repositories, as shown in Figure 2. Clients, such as IT processes, management applications, and IT staff will use a query service defined in the specification to access aggregated or non-aggregated views. Data repositories will use the services described in the specification to provide the aggregated view.

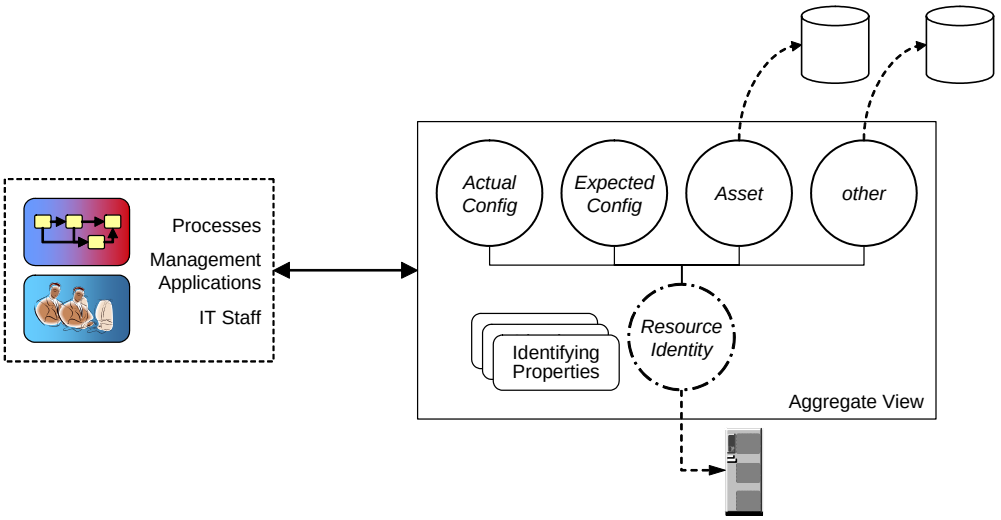


Figure 2 – Aggregate View from Federated Data

The federated CMDB could support the following scenarios (though which scenarios are supported is entirely left to the discretion of each implementation):

- Maintain accurate picture of IT inventory from a combination of asset information (finance) and deployment/configuration
- Reflect changes to IT resources, including asset and licensing data, across all repositories/data sources
- Compare expected configuration vs. actual configuration
- Enable version awareness. Examples:
  - Coordinate planned configuration changes
  - Track change history
- Relate configuration and asset data to other data/data sources, such as incident, problem, and service levels. Examples:
  - Integration of change/incident management with monitoring information
  - SLA incident analysis – use of service desk/incident information in a dependency analysis on both configurations and change records

1.1.2 Target IT Environment

This specification is intended to address requirements in IT environments with the following characteristics

- There are strong requirements to consolidate into one or more databases (logical and/or physical) at least some key data from the many management data repositories so that IT processes can be more effective and efficient.
- IT organizations that implement a CMDB that federates multiple management data repositories will be diverse in terms of their existing tools, process maturity level, usage patterns, and preferred adoption models.
- There are several and possibly many management data repositories (MDRs), each of which may be considered an authoritative source for some set of data.
- The authoritative data for a resource may be dispersed across multiple MDRs.
- It is often neither practical nor desirable for all management data to be kept in one data repository, though it may be practical and desirable to consolidate various subsets of the data into fewer databases.
- Existing management tools will often continue to use their existing data sources. Except over the very long haul, it is not realistic to expect them all to be modified to require and utilize new consolidated databases.

1.1.3 Non-Goals

The following are outside the scope of the specification.

- The mechanisms used by each management data repository to acquire data. For example, the mechanisms could be external instrumentation or proprietary federation and replication function.
- The mechanisms and formats used to store data. The specification is concerned only with the exchange of data. A possible implementation is a relational database that stores data in tables. Another possible implementation is a front-end that accesses the data on demand from an external provider, similar to a commonly used CIMOM/provider pattern.
- The processes used to maintain the data in the federated CMDB. The goal of the specification is to enable IT processes to manage this data, but not to require or dictate specific processes.
- The mechanisms used to change the actual configuration of the IT resources and their relationships. The goal of the specification is to provide means to represent changes after or as they are made, but not to be the agent that makes the change.

1.2 Background Terminology

This non-normative section defines terms used throughout this specification. For the most part, these terms are adopted from other sources. The terms are defined here to clarify their usage in this specification and, in some cases, to show their relationship to the use of the terms in other sources. In particular, this specification shares concepts with ITIL (Information Technology Infrastructure Library.) ITIL is not a standard and does not provide normative definitions of terms. However, the ITIL v3 glossary is quoted below as representative of the ITIL position.

Configuration Item (CI) A Configuration Item is a basic tangible or intangible entity in a configuration management solution such as a CMDB. ITIL v3 defines a CI as

> Any Component that needs to be managed in order to deliver an IT Service. Information about each CI is recorded in a Configuration Record within the Configuration Management System and is maintained throughout its Lifecycle by Configuration Management. CIs are under the control of Change Management. CIs typically include IT Services, hardware, software, buildings, people, and formal documentation such as Process documentation and SLAs.

Configuration Management Database (CMDB) ITIL defines a CMDB as

> A database used to store Configuration Records throughout their Lifecycle. The Configuration Management System maintains one or more CMDBs, and each CMDB stores Attributes of CIs, and Relationships with other CIs.

A Configuration Management Database (CMDB) is often implemented using standard database technology and typically persists CI lifecycle data as records (or Configuration Records) in that database. Configuration records are managed according to some data or information model of the IT environment. One of the goals of this specification is to expedite the federated implementation of multiple CMDBs in a single Configuration Management System.

Configuration Record ITIL defines a Configuration Record as

> A Record containing the details of a Configuration Item. Each Configuration Record documents the Lifecycle of a single CI. Configuration Records are stored in a Configuration Management Database.

For the purposes of this specification, a CI is a tangible or intangible entity treated in the abstract by this specification, while a Configuration Record contains concrete data pertaining to a CI. More than one Configuration Record may be associated with a given CI. Often Configuration Records will be from different data sources or document different points in the lifecycle of a CI. It is possible for Configuration Records associated with a single CI to contain data that may appear contradictory and require mediation.

Federated CMDB A federated CMDB is a combination of multiple management data repositories (MDRs), at least one of which federates the others, into an aggregate view of management data. Note that whereas “federated CMDB” refers to the combination of all the data repositories, “Federating CMDB” is a specific role performed by a data repository that federates other MDRs.

Federation The process of combining information from management data repositories (MDRs) into a single representation that can be queried in a consistent manner. Federation is often contrasted with Extract, Transform, and Load (ETL) systems which transfer and store data from one repository to another. This specification does not exclude ETL activities, especially for caching, but the main purpose of the specification is to support systems that minimize or eliminate transferring and storing data from MDRs in federators.

Graph A **graph** is a kind of data structure, specifically an abstract data type, that consists of a set of nodes and a set of edges that establish relationships (connections or links) between the nodes. In this specification the nodes are Items and the edges are Relationships.

Identity The federated CMDB contains data pertaining to real world entities. The identity of each of these real world entities is a set of qualities or characteristics that

distinguish the entity from other entities of the same or different types. This set of qualities may be called the 'identifying properties' of the entity.

ITIL ITIL stands for Information Technology Infrastructure Library and is a framework of best practices for delivering IT services. Two versions of ITIL are currently in use: version 2 released in 2000 and version 3 released in 2007. Since v3 has not yet superseded v2 in practice, both versions have been considered in preparing this specification. A CMDB is a key component in the ITIL best practices.

1.3 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC 2119].

This specification uses the following syntax to define outlines for messages:

- The syntax appears as an XML instance, but values in italics indicate data types instead of literal values.
- Characters are appended to elements and attributes to indicate cardinality:
  - "?" (0 or 1)
  - "*" (0 or more)
  - "+" (1 or more)
  - The absence of any of the above indicates the default (exactly 1)
- The character "|" is used to indicate a choice between alternatives.
- The characters "(" and ")" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- The characters "[" and "]" are used to call out references and property names.
- Ellipses (i.e., "...") indicate points of extensibility. Additional children and/or attributes MAY be added at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. By default, if a receiver does not recognize an extension, the receiver SHOULD ignore the extension; exceptions to this processing rule, if any, are clearly indicated below.
- XML namespace prefixes are used to indicate the namespace of the element being defined or referenced.

2. Technological Assumptions

This specification is based on some very specific assumptions with regard to underlying technology and the context of computing standards that exists at the time of its writing.

2.1 Underlying Technology

2.1.1 Web Services

Although the interface specification contained herein is generic, it assumed that implementations will be based on Web Services. Although interfaces based on programming languages such as Java and C# could be derived from this specification, such interfaces are considered out of scope and are not addressed here.

2.1.2 Database Management Systems

In general practice CMDBs are implemented using commercially available database technology. Although this is a specification about how one or more CMDBs federate data using a standard mechanism, no assumptions are made about how that federated data is stored or persisted. What is important are the interfaces; their behavior and the data types they convey. Database technology is clearly a needed component in the implementation of this specification, but its use is considered to be a hidden detail of such implementations.

2.2 Standards Basis

This specification builds upon the work of other standards in the area Web Services. The specific standards that this specification is based on are as follows.

- HTTP/1.1
- XML Schema Part 1: Structures
- SOAP 1.1
- WSDL 1.1
- WS-I Basic Profile 1.1

3. Architecture

The architecture defines four roles, which implement or use two services. In Figure 3 the roles are (green) shaded shapes with dotted edges and the services are (yellow) shaded rounded boxes with solid edges.

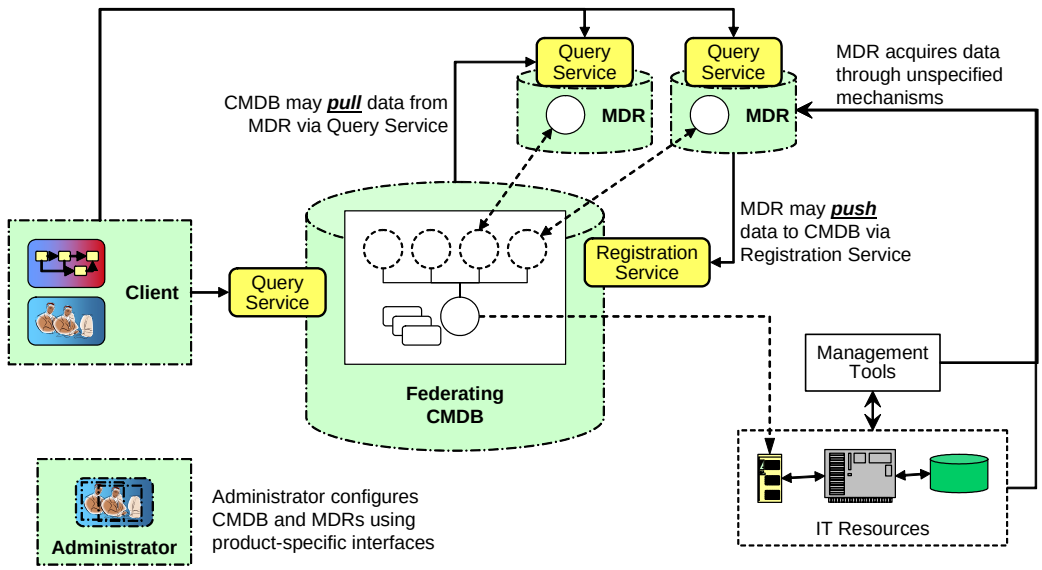


Figure 3 – Roles and Services

3.1 Roles

MDR (Management Data Repository). An MDR contains data about managed resources (e.g., computer systems, application software, and buildings) and/or

process artifacts (e.g., incident records and request for change forms), and the relationships between them. In this architecture, managed resources and process artifacts are both called 'items'. The means by which the MDR acquires data is not specified. Examples include direct from instrumented resources or indirectly through management tools.

Federating CMDB. A Federating CMDB federates data from MDRs, and may also contain non-federated data. It provides an aggregate view of an item or relationship, potentially using data from multiple MDRs. A Federating CMDB and all the MDRs together comprise a federated CMDB.

It is possible for one Federating CMDB to have its data federated by a second Federating CMDB. In this case, the first Federating CMDB would appear to the second Federating CMDB to be an MDR. The second Federating CMDB would not be aware of any federation performed by the first Federating CMDB.

Client. A Client is a consumer of management data, either directly from an MDR or an aggregated view from a Federating CMDB. Examples of clients are IT process workflows, management tools, and IT administrators. Clients only read data; there are no provisions for a client to update data through an interface defined in this architecture.

Administrator. An Administrator configures MDRs and Federating CMDBs so they can interact with each other. Administration includes selecting and specifying the data that is federated, describing service endpoints, and describing which data are managed through each endpoint. Administration is done using interfaces that are specific to each tool that acts in the MDR and/or Federating CMDB role.

3.2 Services Overview

The architecture defines two services. There is an implementer of a service and a client (caller) of a service.

Query Service. Both MDRs and Federating CMDBs make data available to Clients via a Query service. Queries may select and return items, relationships, and/or graphs containing items and relationships.

Registration Service. An MDR can register data that it has available for federation by a Registration service. A Federating CMDB declares the data types that its Registration service supports. An MDR maps its data to the supported types.

3.2.1 Federation Modes

There are two modes available to federate data. A Federating CMDB must use one or the other mode and MAY use both.

Push Mode. In push mode, the MDR initiates the federation. Typically an administrator configures the MDR by selecting to federate some data types that are supported by both the MDR and the registration service. The MDR notifies the Registration service any time this data is added, updated, or deleted. Depending on the extent of the data types, the registered data may be limited to identification data or it may include many other properties that describe the item or relationship state.

Pull Mode. In pull mode, the Federating CMDB initiates the federation. Typically, an administrator configures the Federating CMDB by selecting the MDR data types that will be federated. The Federating CMDB queries MDRs for instances of this data. Depending on the implementation, the Federating CMDB may pass through queries to MDRs without maintaining any state, or it may cache some set of MDR data, such as the data used to identify items and relationships.

3.2.2 Usage Profiles

Table 1 lists the service usage profiles for the roles described in section 3.1 that implement or use the services.

Table 1 – Service Usage Profiles

Role	Query service		Registration service	
	Implementation	Client	Implementation	Client
Federating CMDB – Push Mode	REQUIRED	Optional	REQUIRED	No support
Federating CMDB – Pull Mode	REQUIRED	REQUIRED	No support	No support
MDR – Push Mode	Optional	No support	No support	REQUIRED
MDR – Pull Mode	REQUIRED	No support	No support	No support
Client (external)	No support	REQUIRED	No support	No support

3.3 Identity Reconciliation

Managed resources are often identified in multiple ways, depending on the management perspective. Examples of management perspectives are a change management process and an availability monitoring tool. Understanding how to identify resources, and reconciling the identifiers across multiple perspectives, is an important capability of a Federating CMDB. The following pattern is used:

- Each MDR identifies a resource based on one or more identifying properties of the resource. Identifying properties are physical or logical properties that distinguish unique instances of resources. Examples are MAC addresses, host names, and serial numbers. Often, more than one property will be necessary to uniquely distinguish a resource, especially when information is incomplete. In addition, when two or more MDRs contain data on a single resource, individual MDRs may choose or have available different identifying properties, which they may use in their resource identifier for the item or relationship.
- Each MDR knows at least one unique and unambiguous identifier for each item or relationship it contains and/or provides access to via the Query service.
- A Federating CMDB attempts to reconcile the item and relationship identification information from each MDR, recognizing when they refer to the same item or relationship.

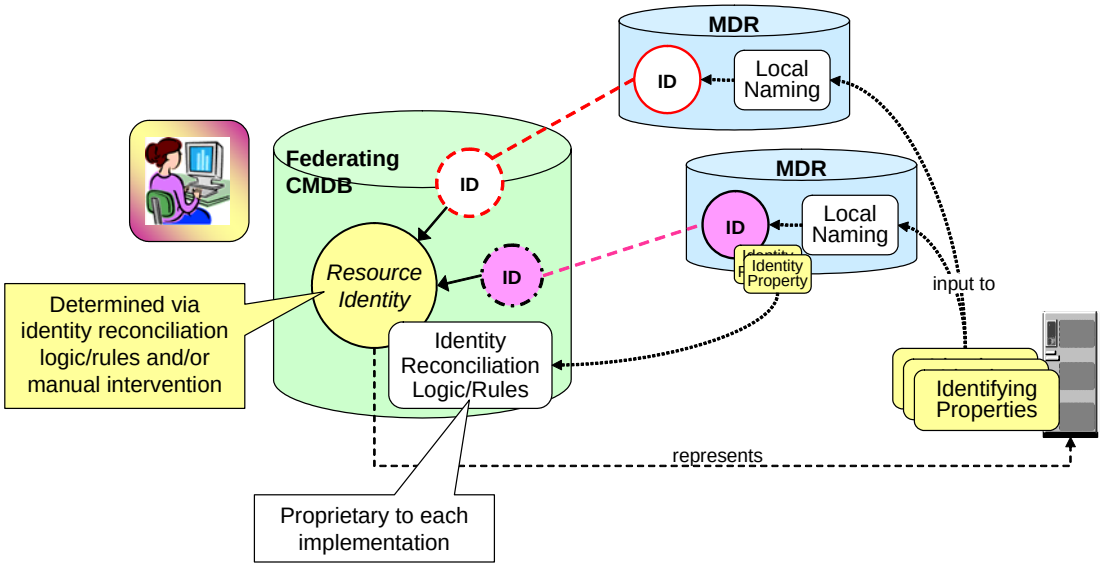


Figure 4 – Identity Reconciliation

The Federating CMDB performs this mapping using any combination of automated analysis and manual input, as shown in Figure 4. In a typical implementation the Federating CMDB analyzes the identifying properties to determine the resource identity. As each item or relationship is registered, the service determines if this item or relationship is already registered or is new. The determination of identity is seldom absolute and often must rely on heuristics because different MDRs typically know about different characteristics of an entity and thus establish different sets of identifying properties which characterize the entities they handle. Further, the determination may change as additional information is discovered and MDRs add, subtract, or change identifying properties as systems evolve.

3.4 Data Model Overview

3.4.1 Managed Data

The architecture defines three elements that wrapper properties that are specific to the type of item or relationship.

Item. An item represents a managed resource (e.g., computer systems, application software, and buildings) or a process artifact (e.g., incident record and request for change form). With this definition, 'item' is a superset of the 'configuration item' term defined in ITIL. Each item has at least one ID that is unique within the scope of the MDR that contains it and that serves as a key. Examples of when an item might have multiple IDs include when an item is reconciled across several MDRs and the Federating CMDB knows it by all of the IDs that have been assigned by different MDRs; when two items are thought to be different but are later reconciled to the same item; when an ID changes for any other reason. Once an ID has been assigned to an item, it can be used in any situation requiring an ID, and will never refer to anything except the original item.

Given that each MDR has a unique ID within the group of federated repositories, and that each MDR assigns a unique ID within its own scope, the combination of the MDR ID and the MDR-assigned item ID results in an instance ID that is unique within the group of federated repositories. This instance ID serves two purposes:

- It is an unambiguous identifier for the representation of the item held by the MDR that assigned the instance ID.
- The MDR ID portion of the instance ID identifies the MDR that assigned the instance ID. A client may introspect the instance ID to extract the MDR ID. The client may then use the MDR ID to acquire the query service address for this MDR. For example, the MDR ID might be the key in a registry that contains the service addresses for each MDR. The client may then issue a query to this address to retrieve the representation of the item.

When a Federating CMDB federates item data from an MDR, it may respond to queries for the representation of the item. It may reuse the instance ID assigned by the MDR as long as the representation that it returns is the same as the representation that would be returned by the MDR that assigned the instance ID. If the Federating CMDB alters the representation, such as overwriting some property values or associating other records to the same item, it must assign a new instance ID using its own MDR ID.

This constraint on reusing IDs is not meant to preclude caching of the MDR data in the Federating CMDB. In particular, it is recognized that because of the distributed configuration of the repositories, and the absence of any requirements that their data are entirely coherent, such as requiring transactional closure across the repositories for any update, at any instant in time a query to the Federating CMDB may return a different representation than the same query to the MDR.

Relationship. A relationship represents a connection from a source item to a target item. Examples include software 'runs on' operating system, operating system 'installed' on computer system, incident record 'affects' computer system, and service 'uses' (another) service. Like an item, each relationship has an ID that is unique within the scope of the MDR that contains it and that serves as a key. And like an item, a reconciled relationship can have more than one such ID.

Record. A record contains properties that describe an item or relationship. A record is associated with one item or relationship. A record may contain properties that are useful to identify the item or relationship, or other properties that describe the item or relationship. Several records may be associated to the same item or relationship. Records may differ from other records for various reasons, including types of data (e.g., asset vs. configuration), different sets of properties from different providers, different versions, and expected vs. observed data. A record is similar to a row in a SQL view. It is a projection of properties. The same property may appear in multiple records for the same item or relationship. The record may have no properties, in which case it serves as a marker. Each record has an ID that is unique within the scope of its associated item or relationship ID, and that serves as a key.

The data contained in an MDR or Federating CMDB is a graph where the items are nodes and the relationships are links. The graph is not necessarily connected (there may not be a relationship trail from any item to any other item). The query interface described below allows queries to be constructed based on aspects of the graph (e.g. existence of a relationship between two items) and based on properties of the items and relationships (e.g. requirements for a certain value of a given record property or a certain type for the item/relationship).

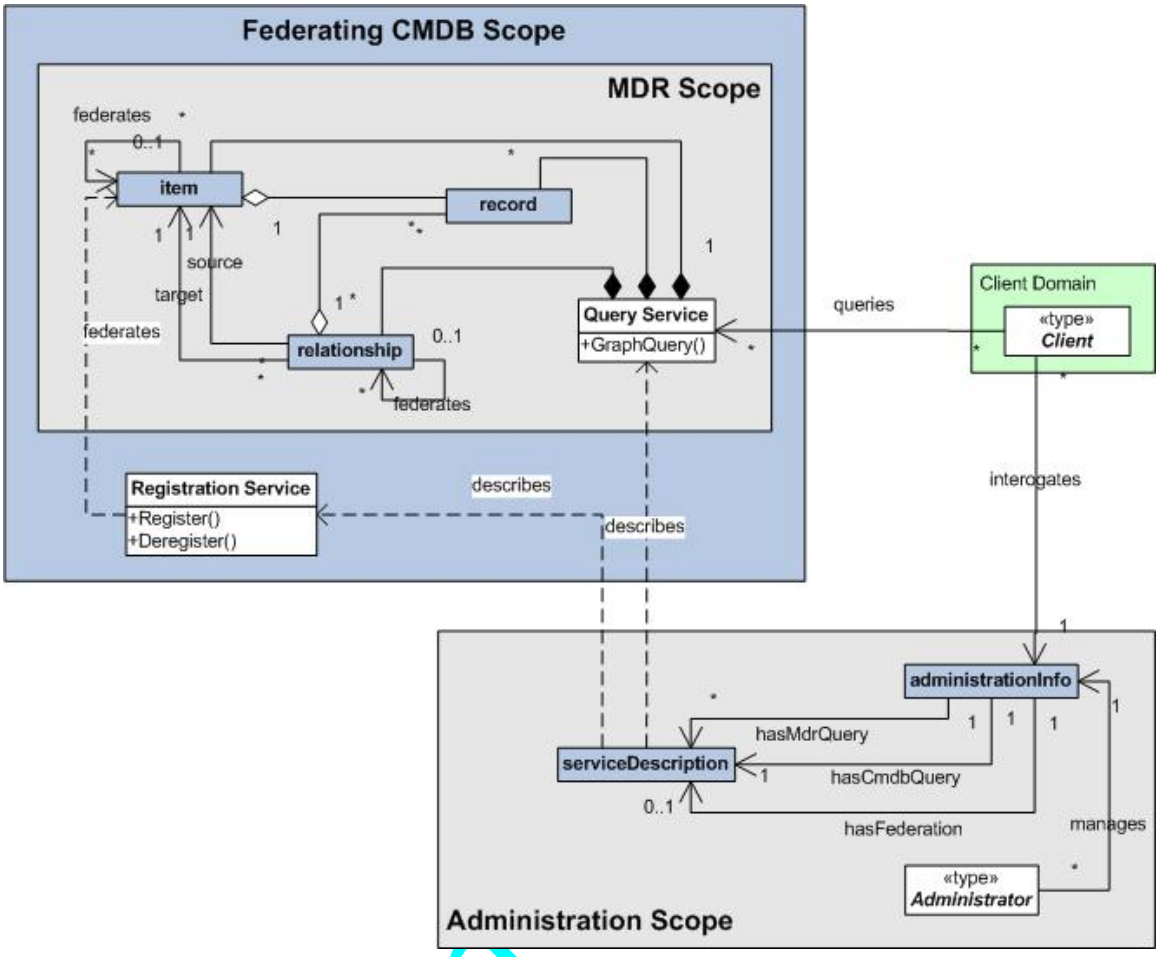


Figure 5 –Data & Services Overview

> Note: The specification, including this figure, indicate that the source and target of a relationship must be an item. It is expected that the specification will be extended to also permit the source and/or target of a relationship to be another relationship.

3.4.2 Administration Data

The architecture defines two elements that describe services.

Service Description. A serviceDecription describes an instance of a Query Service or Registration Service. The description includes an ID, descriptive text, the record types it supports and/or requires, and other capabilities that it supports (such as types of query selectors).

Administration Information. An administrationInfo element is the anchor for all the service descriptions. The specification does not describe operations for creating, accessing, or altering the descriptions. Each service implementation is expected to

use appropriate available mechanisms, such as creating the definitions in a service registry.

> Note: A normative definition of the XML schema for serviceDescription and administrationInfo will be added to the specification.
>
> Note: Administrative operations to retrieve instances of serviceDescription and/or administrationInfo may be added to the specification.

4. Query Service

4.1 Overview

The Query service can be provided by MDRs and Federating CMDBs. It provides a way to access the items and relationships that the provider (MDR or Federating CMDB) has access to, whether this provider actually holds the data or federates the source of the data. The Query service contains a single operation, GraphQuery, that can be used for anything from a simple instance query to a much more complex topological query.

A GraphQuery request describes the items and relationships of interest in the form of a graph. Constraints can be applied to the nodes (items) and edges (relationships) in that graph to further refine them. The GraphQuery response contains the items and relationships that, through their combination, compose a graph that satisfies the constraints of the graph in the query.

The following example and normative definition of the interface provide a more complete description of the request and response messages for the GraphQuery operation.

4.2 Example

Let's assume that an MDR contains two types of items (people and computers) and one type or relationships (a person "uses" a computer). Here is a simple query request to select all computers that are used by a person located in California:

```
(01) <query>

(02)   <itemTemplate id="user">
(03)     <propertyValueSelector namespace="http://example.com/people"
                                localName="state">
(04)       <equal>CA</equal>
(05)     </propertyValueSelector>
(06)     <recordTypeSelector namespace="http://example.com/people"
                             localName="person"/>
(07)   </itemTemplate>

(08)   <itemTemplate id="computer">
(09)     <recordTypeSelector namespace="http://example.com/computer"
                             localName="computer"/>
(10)   </itemTemplate>
```

```
(11)   <relationshipTemplate id="usage">
(12)     <recordTypeSelector namespace="http://example.com/computer"
                             localName="uses"/>
(13)     <source ref="user"/>
(14)     <target ref="computer"/>
(15)   </relationshipTemplate>

(16) </query>
```

The detailed syntax and semantics of the XML elements are described in details in later sections, but here is in summary what items and relationships get selected by this query:

The <itemTemplate> called “user” (line 02) selects all items that:

- have a record with a property called “state” (in the http://example.com/people namespace) for which the value is “CA”,
- have a record named “person” (defined in the namespace “http://example.com/people”), and
- are the source of a relationship that is selected by the <relationshipTemplate> called “usage” (line 11)

The <itemTemplate> called “computer” (line 08) selects all items that:

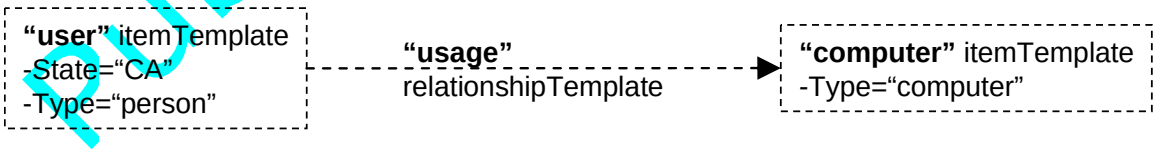
- have a record named “computer” (defined in the namespace “http://example.com/computer”), and
- are the target of a relationship that is selected by the <relationshipTemplate> called “usage” (line 11)

The <relationshipTemplate> called “usage” (line 11) selects all relationships that:

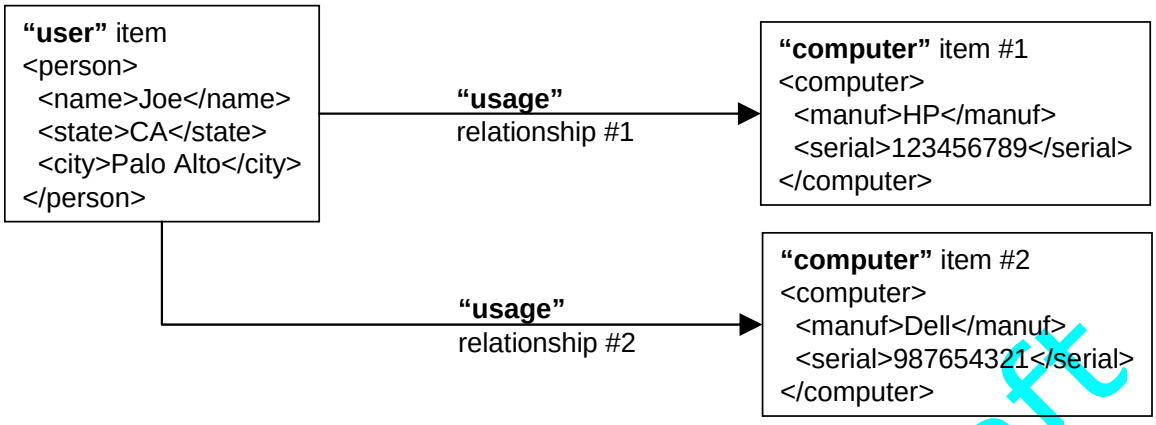
- have a record named “uses” (defined in the namespace “http://example.com/computer”),
- have a source that is selected by the <itemTemplate> called “user” (line 02), and
- have a target that is selected by the <itemTemplate> called “computer” (line 08).

As a result, if a user item does not “use” a computer, it will not be part of the response, whether the user is located in California or not.

Here is a graphical representation of the query:



If a user located in California happens to “use” two computers, this is represented in the response by three items (one for the user and one for each computer) and two relationships (going from the user to each of his/her computers). Later section will describe the syntax and semantics of the response message in more details. Here is a graphical representation of this response:



In effect, the response contains two graphs, each made of a user, a computer and the relationship between the two, that both meet the constraints of the query graph. In this example, the two graphs in the response happen to overlap (they share the same "user") but in another example they could be disjoint (e.g. if the second computer was instead "used" by another user also located in California).

If the <relationshipTemplate> element (line 11) was not part of the query, the semantics of the query would be very different. The query would return all the items of type "person" that are in California and all the items of type "computer". It would not return the relationships between users and computers. The existence (or not) of these relationships would have no bearing on what items get selected.

4.3 Normative definition

4.3.1 GraphQuery

As illustrated in the previous example, a GraphQuery request consists of a <query> element containing <itemTemplate> and <relationshipTemplate> elements. Templates (of either kind) can contain selectors. The same selector types are used (with the same meaning) inside <itemTemplate> and <relationshipTemplate> elements. In addition to selectors, <relationshipTemplate> elements also contain a <source> and a <target> element. These elements each point (using the xs:ID/xs:IDREF mechanism) to an <itemTemplate>.

Here is the pseudo-schema of the payload of a GraphQuery request:

```
(01) <query>
(02)   <itemTemplate id="xs:ID" dropDirective="xs:boolean"?>
(03)     <instanceIdSelector>...</instanceIdSelector> ?
(04)     <propertyValueSelector ...>...</propertyValueSelector> *
(05)     <xpath1Selector>...</xpath1Selector> *
(06)     <recordTypeSelector ... /> *
(07)     <propertySubsetDirective>
(08)       <selectedProperty namespace="xs:anyURI"
(09)                         localName="xs:NCName" /> *
(10)     </propertySubsetDirective> ?
(11)     ...
(12)   </itemTemplate> *
(13)   <relationshipTemplate id="xs:ID" dropDirective="xs:boolean"?>
(14)     <instanceIdSelector>...</instanceIdSelector> ?
(15)     <propertyValueSelector>...</propertyValueSelector> *
```

```
(16)     <xpath1Selector>...</xpath1Selector> *
(17)     <recordTypeSelector>...</recordTypeSelector> *
(18)     <propertySubsetDirective>
(19)      <selectedProperty namespace="xs:anyURI" localName="xs:NCName"/> *
(20)     </propertySubsetDirective> ?
(21)
(22)     <source ref="xs:IDREF" minimum="xs:int"? maximum="xs:int"?/>
(23)     <target ref="xs:IDREF" minimum="xs:int"? maximum="xs:int"?/>
(24)     ...
(25)   </relationshipTemplate> *
(26) </query>
```

The exact syntax and semantics of each selector element (<instanceIdSelector>, <propertyValueSelector>, <xpath1Selector> and <recordTypeSelector>) will be described in a later section. For now suffice to say that the evaluation of a selector on an item or relationship returns a Boolean. If the value of the Boolean is “true” then the item or relationship is deemed to meet the constraint defined by the selector.

The value of the @ref attributes of the <source> and <target> elements must each correspond to the value of the id attribute of an <itemTemplate> element in the query. They indicate which <itemTemplate> elements represent the items that should play the role of source and target, respectively, for the relationships selected by this <relationshipTemplate>.

The optional @minimum and @maximum on <source> and <target> are used to specify minimum and maximum cardinality. For example, only finding servlet containers in which at least 10 servlets but not more than 20 are deployed. The precise usage for these attributes is described below.

An item is selected by an <itemTemplate> if and only if:

- the item meets all the constraints defined by all the selectors in the <itemTemplate> (in effect, there is an implicit AND joining the selectors),
- for every <relationshipTemplate> that points to the <itemTemplate> as its source, there is a relationship selected by this <relationshipTemplate> that has the item as its source, and
- for every <relationshipTemplate> that points to the <itemTemplate> as its target, there is a relationship selected by this <relationshipTemplate> that has the item as its target.

Relationships cannot be selected by an <itemTemplate>.

An item can be selected at most once per <itemTemplate>. But the same item can be selected by more than one <itemTemplate> inside a given query. When this is the case, the item appears in the response once for each <itemTemplate> that selects it (and each of these occurrences follows the representation directives, i.e. the “dropped” and “property Subset” directives described below, in the corresponding <itemTemplate>)

A relationship is selected by a <relationshipTemplate> if and only if:

- the relationship meets all the constraints defined by all the selectors in the <relationshipTemplate> (in effect, there is an implicit AND joining the selectors),
- the source item of the relationship is selected by the <itemTemplate> referenced as <source> by the <relationshipTemplate>, and

- the target item of the relationship is selected by the <itemTemplate> referenced as <target> by the <relationshipTemplate>, and
- for each of the "minimum" or "maximum" attributes that appear on either the <source> or <target> element in the <relationshipTemplate>, the corresponding cardinality condition below is met (if the attribute is not present then no cardinality condition applies, which is equivalent to saying that "minimum" defaults to zero and "maximum" defaults to "infinite"):
    - if n is the value of <source>/@minimum, there are at least n relationships (including the current one) selected by the <relationshipTemplate> that share the same source item,
    - if n is the value of <source>/@maximum, there are at most n relationships (including the current one) selected by the <relationshipTemplate> that share the same source item,
    - if n is the value of <target>/@minimum, there are at least n relationships (including the current one) selected by the <relationshipTemplate> that share the same target item,
    - if n is the value of <target>/@maximum, there are at most n relationships (including the current one) selected by the <relationshipTemplate> that share the same target item.

Items cannot be selected by a <relationshipTemplate>.

The optional dropDirective attribute and <propertySubsetDirective> element do not influence which items and relationships get selected. They only affect how the items and relationships are represented in the response message. See the "Definition of directives" section below for a description of their effect.

4.3.1.1 ***Definition of selectors***

Selectors and directives are defined identically whether they are contained inside of an <itemTemplate> or a <relationshipTemplate> element. In this section and the following one, we use the term "instance" to mean either an item or a relationship.

<instanceIdSelector>

The <instanceIdSelector> element is used to point to a specific instance by its Id. The pseudo-schema of this selector is:

```
(01) <instanceIdSelector>
(02)   <mdrId>xs:anyURI</mdrId>
(03)   <localId>xs:anyURI</localId>
(04) </instanceIdSelector>
```

There can be at most one <instanceIdSelector> in an <itemTemplate> or a <relationshipTemplate> element.

An instance Id is composed of a pair of URIs. The first URI, <mdrId>, is the ID of the MDR that assigned this instance Id to the instance. The second URI, <localId>, is the Id that uniquely identifies the instance within the MDR. The combination of these two URIs identifies the instance in a globally unique way.

There is no expectation that these two URIs are able to be de-referenced.

More than one instance Id may be attached to one instance. For example, a Federating CMDB may know, for a given reconciled instance, instance Ids provided by each of the MDR that have content about the instance, plus possibly an additional instance Id for the instance assigned by the Federating CMDB itself.

The selector returns a positive result if one of the known instance Ids for the instance corresponds to the requested value, i.e. if both the <mdrId> and the <localId> match (using string comparison).

<propertyValueSelector>

Each instance is associated with zero or more records. These records contain properties whose values are accessible through an XML representation of the instance. The <propertyValueSelector> element can only be used on properties that have a type that is a subtype of the xs:anySimpleType type. While the type must be known, it is not required that an XML schema definition of the property be available.

Instances may be selected based on property values. The <propertyValueSelector> element is one way to do so for properties that are defined as a simple type (as defined by XML schema). It is not applicable to properties that are defined as a complex type.

The pseudo-schema of this selector is:

```
(01) <propertyValueSelector namespace="xs:anyURI"
(02)                        localName="xs:NCName"
(03)                        matchAny="xs:boolean">
(04)   <equal caseSensitive="xs:boolean"? negate="xs:boolean"? >
(05)     xs:anySimpleType
(06)   </equal> *
(07)   <less negate="xs:boolean"? >xs:anySimpleType</less> ?
(08)   <lessOrEqual negate="xs:boolean"? >xs:anySimpleType</lessOrEqual> ?
(09)   <greater negate="xs:boolean"? >xs:anySimpleType</greater> ?
(10)   <greaterOrEqual negate="xs:boolean"?>
(11)     xs:anySimpleType
(12)   </greaterOrEqual> ?
(13)   <contains caseSensitive="xs:boolean"? negate="xs:boolean"? >
(14)     xs:string
(15)   </stringContains> *
(16)   <like caseSensitive="xs:boolean"? negate="xs:boolean"? >
(17)     xs:string
(18)   </like> *
(19)   <isNull negate="xs:boolean"? /> ?
(20)   ...
(21) </propertyValueSelector>
```

This selector can appear any number of times in an <itemTemplate> or a <relationshipTemplate>. Its namespace and localName attributes define the QName of the property being tested. The children elements of <propertyValueSelector> are called operators. The matchAny attribute on <propertyValueSelector> defines whether the operators inside that element are logically AND-ed or OR-ed. The default value is false. If the value of the matchAny attribute is false, the selector returns a positive result for an instance if the instance has a record that contains the property identified by the QName and if the value of that property satisfies all the operators in the selector. If the value of the matchAny attribute is true, the selector returns a positive result for an instance if the instance has a record that contains the property identified by the QName and if the value of that property satisfies at least one of the operators in the selector.

A <propertyValueSelector> is considered to be positive (true) if the operators return a positive (true) result for one or more records associated with the instance.

The operators are largely defined in terms of XPath 2.0 [XPath 2.0] comparison operators. This does not require that an XPath 2.0 implementation be used but only that the operators be evaluated in a way that is consistent with the XPath 2.0 definitions, as described below.

- **equal** - this operator is defined in terms of the XPath 2.0 value comparison operators "eq". To evaluate, the left hand operand is the property value from the record and the right hand operand is the value of the selector from the query. The type of the value of the selector must be interpreted to be of the same type as the value from the property in the record. This operator is valid for properties of any simple type. A list of comparison behaviors is available in the XPath 2.0 Appendix B.2 Operator Mappings.
- **less**, **lessOrEqual**, **greater**, and **greaterOrEqual** - these operators are defined in terms of the XPath 2.0 value comparison operators of "lt", "le", "gt", and "ge", respectively. To evaluate, the left hand operand is the property value from the record and the right hand operand is the value of the selector from the query. The type of the value of the selector must be interpreted to be of the same type as the value from the property in the record. This operator is only valid for properties that are numerals, dates and strings. A list of comparison behaviors is available in the XPath 2.0 Appendix B.2 Operator Mappings. For example, if a property is of type date, the operator <less>2000-01-01T00:00:00</less> returns true if the property value is a date before the year 2000. If the property value was a string then "2000-01-01T00:00:00" would be interpreted as a string and compared with the property value using string comparison.
- **contains** - this operator is mapped to the XPath 2.0 function fn:contains(). It is only valid for properties of type string and used to test if the property value contains the specified string as a substring. The result of the contains operator is as if the fn:contains() function was executed with the first parameter being the property value and the second parameter being the string specified.
- **like** - this operator is similar in functionality to the SQL LIKE clause. The operator works like the equal operator with the inclusion of two special characters: the underscore ("_") acts as a wild card for any single character and the percent sign ("%") acts as a wild card for zero or more characters. To escape the wild cards, the backslash("\") can be used. For example, <like>Joe_Smith%</like> tests whether the property value starts with the string "Joe_Smith" and would match values such as "Joe_Smith", "Joe_Smith123" and "Joe_Smith_JR". It would not match "JoeHSmith123". A double backslash ("\\") represents the single backslash string ("\").
- **isNull** - this operator tests whether the element corresponding to the property is "nilled". It is equivalent to the result of applying the XPath 2.0 "fn:nilled" function on the element corresponding to the property.

Additional attributes defined:

- **caseSensitive** - equal, contains, and like operators have an optional attribute, caseSensitive, with a default value of true. If the property value of the record is an instance of xs:string and the attribute caseSensitive is false, the string comparison is case-insensitive. More precisely, the result of the comparison is as if the XPath 2.0 function fn:upper-case() was called on both

the property value and the string value before comparison. If the property value of the record is not an instance of a xs:string, the caseSensitive attribute has no impact on the comparison.

- **negate** - all operators have an optional attribute, negate, with a default value of false. When the negate attribute is true, the result of the comparison is negated.

As a summary, the following table shows what operators are supported on the various XSD built-in types. Unless explicitly specified, the caseSensitive attribute is not supported.

Built-in Datatypes	**equal**	**isNull**	**less, lessOrEqual, greater, greaterOrEqual**	**contains**	**like**
"String-related types" (String, anyURI and types derived from string)	Yes, including optional caseSensitive attribute	Yes	Yes	Yes, including optional caseSensitive attribute	Yes, including optional caseSensitive attribute
"Time-related and numeric types" (duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth, float, double, decimals and all types derived from decimals)	Yes	Yes	Yes	No	No
"Others" (boolean, QName, NOTATION, base64Binary, and hexBinary)	Yes	Yes	No	No	No

Multiple instances of a property:

If there is more than one property using the same QName, the comparison only has to hold true for one of the property values. For example, if there is a computer with three IP addresses:

```
(01) <comp:ComputerConfig xmlns:comp="http://example.com/computers">
(02)   ...
(03)   <comp:ip>1.2.3.4</comp:ip>
(04)   <comp:ip>1.2.3.5</comp:ip>
(05)   <comp:ip>1.2.3.6</comp:ip>
(06)   ...
(07) </comp:ComputerConfig>
```

The following property selector would return a positive result:

```
(01) <propertyValueSelector namespace="http://example.com/computers"
(02)   localName="ip">
(03)   <equal>1.2.3.5</equal>
(04) </propertyValueSelector>
```

When the negate attribute is used on a list of properties, the negation is taken after the operator executes. When negating the equal operator, a positive result is

returned when none of the properties are equal to the given value. For example, on the same computer with three IP addresses:

```
(01) <propertyValueSelector namespace="http://example.com/computers"
(02)   localName="ip">
(03)   <equal negate="true">1.2.3.5</equal>
(04) </propertyValueSelector>
```

The property selector would not select the item above because the equality comparison matches one IP address in the list.

Similary, <less negate="true">12</less> is equivalent to <greaterOrEqual>12</greaterOrEqual> if there is only one instance of the property being tested. But if there is more than one instance of the property, then the first operator is true if all of the instances have a value of more than 12, while the second one is true if at least one of the instances has a value of more than 12.

A simple example of using <propertyValueSelector>:

In the following example, "Manufacturer" is a property defined in the "http://example.com/Computer" namespace. The selector is testing whether the instance has a record containing this property and where the value of the property is "HP".

```
(01) <propertyValueSelector namespace="http://example.com/Computer"
(02)                        localName="Manufacturer" >
(03)   <equal>HP</equal>
(04) </propertyValueSelector>
```

A more complex example:

The <itemTemplate> below selects any item that has a CPUCount greater than or equal to 2, for which the OSName property contains "Linux" (with that exact mix of upper and lower case) and for which the OSName property also contains either "ubuntu" or "debian" (irrespective of case).

```
(01) <itemTemplate id="linuxMachine">
(02)   <propertyValueSelector namespace="http://example.com/computers"
(03)                          localName="CPUCount">
(04)     <greaterOrEqual>2</greaterOrEqual>
(05)   </propertyValueSelector>
(06)   <propertyValueSelector namespace="http://example.com/computers"
(07)                          localName="OSName">
(08)     <contains>Linux</contains>
(09)   </propertyValueSelector>
(10)   <propertyValueSelector namespace="http://example.com/computers"
(11)                          localName="OSName"
(12)                          matchAny="true">
(13)     <contains caseSensitive="false">ubuntu</contains>
(14)     <contains caseSensitive="false">debian</contains>
(15)   </propertyValueSelector>
(16) </itemTemplate>
```

<xpath1Selector>

This selector is an alternate mechanism to filter instances based on the content of their records. The pseudo-schema of this selector is:

```
(01) <xpath1Selector>
```

```
(02)   <prefixMapping prefix="xs:NCName" value="xs:anyURI"/> *
(03)   <xpathExpression>xs:string</xpathExpression>
(04) </xpath1Selector>
```

This selector can appear any number of times inside an <itemTemplate> or a <relationshipTemplate>.

The <xpathExpression> element contains an XPath 1.0 predicate (the part that goes inside [] in XPath 1.0). When testing an instance for this selector, the result is positive if the instance contains a record such that the evaluation of the predicate with the following context returns true:

- Context Node: the first child element of the record element
- Context Position: 1
- Context Size: 1
- Variable Binding: none
- Function Libraries: core function library
- Namespace Declarations: each <prefixMapping> child element of the <xpath1Selector> element defines a namespace declaration for the XPath evaluation. The prefix for this declaration is provided by the <prefixMapping>/@prefix attribute and the namespace URI is provided by the <prefixMapping>/@value attribute.

In the following example, “Name” is a property defined in the “http://example.com/people” namespace. The selector is testing whether the instance has a record containing this property and where the value of the property is “Pete the Lab Tech”.

```
(01) <xpath1Selector>
(02)   <prefixMapping prefix="hr" value="http://example.com/people"/>
(03)   <xpathExpression>hr:name="Pete The Lab Tech"</xpathExpression>
(04) </xpath1Selector>
```

<recordTypeSelector>

This selector is used to test whether an instance has a record of a given type. The pseudo-schema of this selector is:

```
(01) <recordTypeSelector namespace="xs:anyURI" localName="xs:NCName"/>
```

One way for an instance to be selected when tested for this selector is if the instance has a record of that type. More specifically, if the instance contains a record element that has, as first child element, an element in the namespace corresponding to the value of the <recordTypeSelector>/@namespace attribute and where the local name of that first child element is the value of the <recordTypeSelector>/@localName attribute. But this is not the only way for an item to match this selector. A match simply requires that the instance has the characteristics of the requested type. That could be achieved by having an element that is an extension of that QName as a record (for example, comp:Linux might be defined as an extension of comp:OperatingSystem). It could also be achieved by having several records of the instance combined give the instance the characteristics of the requested type.

4.3.1.2 *Definition of directives*

Directives in the query do not influence what instances get selected, but they influence if and how the selected instances get returned in the response. Instances that get selected but not returned play an important role because they influence what other instances get selected. For example, a user may want to retrieve all servlet engines that have at least 30 servlets deployed, but not want to actually retrieve the servlets. The dropDirective attribute can be used to that effect, as described below.

dropDirective

When the dropDirective attribute is present and set to "true" on a template, the instances selected by this template do not get returned in the response. They are only used to further filter instances that are selected by other templates. If the attribute is not present or if its value is false, the instances get returned.

For example, the following simplified query will selected all the servlet engines that have at least 30 servlets deployed, as well as the servlets and the deployment relationships. But it will only return the servlet engines, not the servlets or the "deployedIn" relationships.

```
(01) <query>
(02)   <itemTemplate id="servletEngine">...</itemTemplate>
(03)   <itemTemplate id="servlet"
(04)                 dropDirective="true">...</itemTemplate>
(05)   <relationshipTemplate id="deployedIn" dropDirective="true">
(06)     ...
(07)     <source ref="servlet" minimum="30"/>
(08)     <target ref="servletEngine"/>
(09)   </relationshipTemplate>
(10) </query>
```

<propertySubsetDirective>

If a template contains a <propertySubsetDirective> element, the instances that are selected by this template get returned (unless the template is also marked with dropDirective="true") but the records for the instance are pared down. More specifically, only the properties that are listed (via their namespace and local name) inside the <propertySubsetDirective> element get returned.

A <propertySubsetDirective> with no child element means that the selected instances still get returned, but without any <record> elements. This is different from using dropDirective, with which the instance doesn't appear at all in the response.

In the following example, only the "name" and "telephone" properties in the http://example.com/models/people namespace get returned for the items that match the "user" <itemTemplate>.

```
(01) <query>
(02)   <itemTemplate id="servletEngine">
(03)     ...
(04)     <propertySubsetDirective>
```

```
(05)       <selectedProperty namespace="http://example.com/models/people"
(06)                         localName="name"/>
(07)       <selectedProperty namespace="http://example.com/models/people"
(08)                         localName="telephone"/>
(09)     </propertySubsetDirective>
(10)   </itemTemplate>
(11) </query>
```

4.3.2 GraphQuery Response

The pseudo-schema for the query response message is:

```
(01) <queryResult>
(02)     <nodes templateId="xs:ID">
(03)       <item>
(04)          <record recordId="xs:anyURI">xs:any</record> *
(05)          <instanceId>
(06)            <mdrId>xs:anyURI</mdrId>
(07)            <localId>xs:anyURI</localId>
(08)          </instanceId> +
(09)          <additionalRecordType namespace="xs:anyURI"
(10)                                localName="xs:NCName"/> *
(11)       </item> +
(12)     </nodes> *
(13)     <edges templateId="xs:ID">
(14)       <relationship>
(15)         <sourceItem>
(16)           <mdrId>xs:anyURI</mdrId>
(17)           <localId>xs:anyURI</localId>
(18)         </sourceItem>
(19)         <targetItem>
(20)           <mdrId>xs:anyURI</mdrId>
(21)           <localId>xs:anyURI</localId>
(22)         </targetItem>
(23)         <record recordId="xs:anyURI">xs:any</record> *
(24)         <instanceId>
(25)           <mdrId>xs:anyURI</mdrId>
(26)           <localId>xs:anyURI</localId>
(27)         </instanceId> +
(28)         <additionalRecordType namespace="xs:anyURI"
(29)                               localName="xs:NCName"/> *
(30)       </relationship> +
(31)     </edges> *
(32) </queryResult>
```

Each time an item matches an <itemTemplate>, an <item> element appears inside a <nodes> element in the <queryResult>. The templateId attribute of this element contains the same value as the id attribute of the <itemTemplate> in the original request. If the item is selected by more than one <itemTemplate>, the <item> will be contained in the <nodes> for each <itemTemplate> matched by the item (each one with the appropriate value for its templateId attribute).

Similarly, each time a relationship matches a <relationshipTemplate>, a <relationship> element appears inside an <edges> element in the <queryResult>. The templateId attribute of this element contains the same value as the id attribute of the <relationshipTemplate> in the original request. If the relationship is selected by more than one <relationshipTemplate>, the <relationship> will be contained in the <edges> for each <relationshipTemplate> matched by the relationship (each one with the appropriate value for its templateId attribute).

If no item is part of the response, there are no <nodes> elements. If no relationship is part of the response, there are no <edges> elements.

Items and relationships can contain any number of records. Each is represented by a <record> element. That element contains a single child element. The children of that child are the properties associated with the record.

Items and relationship MUST contain at least one <instanceId> element. The instance Id, through a combination of two URIs (<mdrId> to represent the MDR that assigned the ID and <localId> to uniquely represent the item or relationship inside this MDR), uniquely and globally identifies the item or relationship. There can be more than one <instanceId> element, in the case where the item or relationship has been reconciled from a more fragmented view.

The <sourceItem> child element of a relationship identifies the item that is the source of the relationship. The format of this element matches the format of the <instanceId> element on the item.

The <targetItem> child element of a relationship identifies the item that is the target of the relationship. The format of this element matches the format of the <instanceId> element on the item.

4.4 GraphQuery Example

In this example, the data model contains item records of type ContactInfo and ComputerConfig and relationship records of type 'administers'. ComputerConfigs are related to ContactInfo through the 'administers' relationship to allow for modeling logic such as, "UserA administers ComputerB."

This example queries the graph of the computers which are administrated by Pete the Lab Tech and returns all items and relationships involved in this graph. The response shows two computers administrated by one user.

Here the data we assume the query is executed against.

'User' data:

name	**phone**	**employeeNumber**
Lab Tech	111-111-1111	109
Joe the Manager	111-111-4567	12
Frank the CEO	111-111-9999	1

'Computer' data:

name	**primaryMACAddress**	**CPUType**	**assetTag**	**...**
LabMachineA	00A4B49D2F41	AMD Athlon 64	XYZ9753	
LabMachineB	00A4B49D2F42	AMD Athlon 64	XYZ9876	

LabMachineC	00A4B49D2H11	Intel Pentium 4	XYZ9900	
LabMachineD	00A4B49D2H53	Intel Pentium 4	XYZ9912	

‘Administers’ data:

‘User’ name	**‘Computer’ name**	**adminSupportHours**
Pete the Lab Tech	LabMachineA	24/7
Pete the Lab Tech	LabMachineB	business hours only
Joe the Manager	LabMachineD	24/7

Example "GraphQuery" involving a relationship traversal

```
(01) <query>
(02)   <itemTemplate id="user">
(03)     <propertyValueSelector namespace="http://example.com/people"
(04)                            localName="name">
(05)       <equal>Pete the Lab Tech</equal>
(06)     </propertyValueSelector>
(07)     <recordTypeSelector namespace="http://example.com/people"
(08)                         localName="ContactInfo"/>
(09)   </itemTemplate>
(10)   <itemTemplate id="computer">
(11)     <recordTypeSelector namespace="http://example.com/computerModel"
(12)                         localName="ComputerConfig"/>
(13)   </itemTemplate>
(14)   <relationshipTemplate id="administers">
(15)     <recordTypeSelector namespace="http://example.com/computerModel"
(16)                         localName="administers"/>
(17)     <source ref="user"/>
(18)     <target ref="computer"/>
(19)   </relationshipTemplate>
(20) </query>
```

Example "GraphQuery" response

```
(01)    <queryResult>
(02)        <nodes templateId="user">
(03)          <item>
(04)            <record xmlns:hr="http://example.com/people"
(05)                  recordId="http://example.com/33333/Current">
(06)              <hr:ContactInfo>
(07)                <hr:name>Pete the Lab Tech</hr:name>
(08)                <hr:phone>111-111-1111</hr:phone>
(09)                <hr:employeeNumber>33333</hr:employeeNumber>
(10)              </hr:ContactInfo>
(11)            </record>
(12)            <instanceId>
(13)              <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
(14)              <localId>http://example.com/PeteTheLabTech</localId>
```

```
(15)            </instanceId>
(16)          </item>
(17)        </nodes>
(18)        <nodes templateId="computer">
(19)          <item>
(20)            <record xmlns:comp="http://example.com/computerModel"
(21)              recordId="http://example.com/machines/XYZ9753/scanned">
(22)              <comp:ComputerConfig>
(23)                <comp:CPUType>AMD Athlon 64</comp:CPUType>
(24)                <comp:assetTag>XYZ9753</comp:assetTag>
(25)                <comp:primaryMACAddress>
(26)                  00A4B49D2F41
(27)                </comp:primaryMACAddress>
(28)                <comp:name>LabMachineA</comp:name>
(29)                ...
(30)              </comp:ComputerConfig>
(31)            </record>
(32)            <instanceId>
(33)              <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
(34)              <localId>http://example.com/machines/XYZ9753</localId>
(35)            </instanceId>
(36)          </item>
(37)          <item>
(38)            <record xmlns:comp="http://example.com/computerModel"
(39)              recordId="http://example.com/machines/XYZ9876/scanned">
(40)              <comp:ComputerConfig>
(41)                <comp:CPUType>AMD Athlon 64</comp:CPUType>
(42)                <comp:assetTag>XYZ9876</comp:assetTag>
(43)              <comp:primaryMACAddress>
(44)                00A4B49D2F42
(45)              </comp:primaryMACAddress>
(46)                <comp:name>LabMachineB</comp:name>
(47)                ...
(48)              </comp:ComputerConfig>
(49)            </record>
(50)            <instanceId>
(51)              <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
(52)              <localId>http://example.com/machines/XYZ9876</localId>
(53)            </instanceId>
(54)          </item>
(55)        </nodes>
(56)        <edges templateId="administers">
(57)          <relationship>
(58)            <sourceItem>
(59)              <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
(60)              <localId>http://example.com/PeteTheLabTech</localId>
(61)            </sourceItem>
(62)            <targetItem>
(63)              <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
```

```
(64)              <localId>http://example.com/machines/XYZ9876</localId>
(65)            </targetItem>
(66)            <record xmlns:foo="http://example.com/computerModel"
(67)                  recordId="http://example.com/administers">
(68)              <foo:administers>
(69)                <foo:adminSupportHours>
(70)                  business hours only
(71)                </foo:adminSupportHours>
(72)              </foo:administers>
(73)            </record>
(74)            <instanceId>
(75)              <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
(76)              <localId>
(77)            http://example.com/administers/PeteTheLabTechToLabMachineB
(78)              </localId>
(79)            </instanceId>
(80)          </relationship>
(81)          <relationship>
(82)            <sourceItem>
(83)              <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
(84)              <localId>http://example.com/PeteTheLabTech</localId>
(85)            </sourceItem>
(86)            <targetItem>
(87)              <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
(88)              <localId>http://example.com/machines/XYZ9753</localId>
(89)            </targetItem>
(90)            <record xmlns:foo="http://example.com/computerModel"
(91)                  recordId="http://example.com/administers">
(92)              <foo:administers>
(93)                <foo:adminSupportHours>24/7</foo:adminSupportHours>
(94)              </foo:administers>
(95)            </record>
(96)            <instanceId>
(97)              <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
(98)              <localId>
(99)          http://example.com/administers/PeteTheLabLabTechToLabMachineA
(100)             </localId>
(101)           </instanceId>
(102)         </relationship>
(103)       </edges>
(104)   </queryResult>
```

5. Registration Service

5.1 Overview

The Registration service is used in push mode federation, as described in section 3.2.1 (Federation Modes).

The fundamentals of push mode federation are:

- The MDR invokes the Register operation for items and/or relationships that it wishes to register. Each item or relationship must be associated with at least one record type supported by the Registration service. The MDR may register a subset of the data records it has about any item or relationship.
- The Registration service responds with the registration status for each item or relationship named in the Register operation. The status is either accepted or declined.
    - If the return status is accepted, the Registration service returns the ID that identifies the item or relationship within the Registration service. For accepted data, the MDR is expected to update the Registration service whenever any of the registered data changes. The specification does not stipulate how soon after the data changes the update must occur – this would typically be determined by local policy.
    - If the return status is declined, the Registration service is presumably not maintaining the registration data, and no updates to that data are accepted.
- The specification does not stipulate what the Registration service should or must do with the registered data. The semantics of accepted and declined only have meaning with respect to the obligations of the MDR to update the Registration service when the data changes.
- The MDR also uses the Register operation to update registered data. An update may consist of any combination of:
    - Changes to existing data, such as a property value change
    - Registering an additional record type for this item or relationship
    - Deregistering a previously registered record type for this item or relationship
- The MDR uses the Deregister operation to remove an existing registration for an item or relationship. For example, if the item or relationship is deleted, the MDR would typically delete its own records and deregister the previous registration. Another example when Deregister would be used is if an administrator decides to stop federating the data about this item or relationship, even though the item or relationship still exists and the MDR still maintains data about it.
- The specification does not stipulate what the Registration service should or must do after a Deregister operation. To cite some non-prescriptive examples:
    - If it has the same data from another MDR that this MDR deregisters, it might disassociate the data with the deregistering MDR, while maintaining the existing data.
    - If it has data from another MDR about the deregistered item or relationship, it might delete the deregistered data while maintaining the data from the other MDR.
    - If it has the same data from another MDR, but it considers the deregistering MDR the authoritative source, it might mark the item or relationship as deleted.

- If the deregistering MDR is the only source of data about the item or relationship, it might delete all knowledge of the item or relationship.

5.2 Normative definition

5.2.1 Common data element types

The cmdbf:MdrScopedIdType is used in several places to identify an item or relationship. It contains two URIs: one that is the ID of the enclosing MDR (<mdrId>), and one that is a local ID that is unique within the scope of the MDR (<localId>). The <instanceId> element is of the type of cmdbf:MdrScopedIdType. The pseudo-schema of the <instanceId> element is:

```
(01) <instanceId>
(02)   <mdrId>xs:anyURI</mdrId>
(03)   <localId>xs:anyURI</localId>
(04) </instanceId>
```

This could be abbreviated in a pseudo schema to be:

```
(01) <instanceId>cmdbf:MdrScopedIdType</instanceId>
```

5.2.2 Register

The **Register operation** is used by an MDR to notify a Registration service that new items have been discovered or updated and data is now available in the MDR.

The outline for the Register operation is as follows.

```
(01) <registerRequest>
(02)   <mdrId>cmdbf:MdrScopedIdType</mdrId>
(03)   <itemList>
(04)     <item>
(05)       <instanceId>cmdbf:MdrScopedIdType</instanceId> +
(06)       <record recordId=”xs:anyURI”>
(07)         xs:any
(08)       </record> *
(09)       <additionalRecordType namespace="xs:anyURI"
(10)                             localName="xs:NCName"/> *
(11)     </item> *
(12)   <itemList> ?
(13)   <relationshipList>
(14)     <relationship>
(15)       <instanceId>cmdbf:MdrScopedIdType</instanceId> +
(16)       <sourceItem>cmdbf:MdrScopedIdType</sourceItem>
(17)       <targetItem>cmdbf:MdrScopedIdType</targetItem>
(18)       <record recordId=”xs:anyUri”>
(19)         xs:any
(20)       </record> *
(21)       <additionalRecordType namespace="xs:anyURI"
(22)                             localName="xs:NCName"/> *
(23)     </relationship> *
(24)   <relationshipList> ?
(25) </registerRequest>
```

The following describes additional constraints on the outline listed above:

mdrId

The ID of the MDR registering its data. This ID MUST be unique among all of the MDRs and Federating CMDBs that are federated together.

itemList

The list of items being registered. The list contains any number of <item> elements, though if it contains zero <item> elements, including <itemList> serves no purpose. An <item> SHOULD NOT be repeated in the list.

itemList/item

Some or all of the contents of an <item>.

itemList/item/instanceId

The <instanceId> that serves as a unique key for the <item>. There MUST be at least one for each <item>. The <instanceId> MUST contain the values that would select the <item> in a query using an <instanceIdSelector>.

itemList/item/record

Each <item> contains any number of <record> elements. The <record>@recordId attribute represents a unique key with this MDR for this record.

The <record> element MUST contain exactly one child element. The namespace and local name of the child element together are the record type.

The <record> type MUST be supported by the registration service.

The MDR may support queries for <record> types that it chooses to not federate through the registration service.

There MAY be multiple <record> elements. The set of passed elements will be considered a complete replacement if the registration service already has data from this MDR about this <item>. For example, if the MDR had previously registered this <item> with a ComputerConfiguration and ComputerAsset record, and another registration call is made for the same item with only the ComputerConfiguration record, then it will be treated as a deletion of the ComputerAsset record from the federation.

itemList/item/additionalRecordType

An MDR MAY support through its query interface record types for an item that are not included in the registerRequest message. If so, it MAY indicate the record types for the item by including one or more <additionalRecordType> elements. The <additionalRecordType>/@namespace and <additionalRecordType/@localName attributes together represent the record type. The MDR SHOULD NOT include a <additionalRecordType> if for the same record type it includes a <record>.

For example, the MDR may support for queries ComputerIdentification, ComputerConfiguration, and ComputerAsset records. If the registerRequest message includes only the ComputerIdentification record contents in the <record> element, the MDR may provide in <additionalRecordType> elements the localName and namespace URIs for the ComputerConfiguration and ComputerAsset records

relationshipList

The list of relationships being registered. The list contains any number of <relationship> elements, though if it contains zero <relationship> elements, including <relationshipList> serves no purpose.

relationshipList/relationship

Some or all of the contents of a <relationship>.

relationshipList/relationship/instanceId

The <instanceId> that serves as a unique key for the <relationship>. There MUST be at least one for each <relationship>. The <instanceId> MUST contain the values that would select the <relationship> in a query using an <instanceIdSelector>.

relationshipList/relationship/sourceItem

The <instanceId> that serves as a unique key for the <item> referenced by the source side of a relationship. There MUST be exactly one for each <relationship>. The <instanceId> MUST contain one of the values that would select the source <item> in a query using an <instanceIdSelector>.

relationshipList/relationship/targetItem

The <instanceId> that serves as a unique key for the <item> referenced by the target side of a relationship. There MUST be exactly one for each <relationship>. The <instanceId> MUST contain one of the values that would select the source <item> in a query using an <instanceIdSelector>.

relationshipList/relationship/record

Each <relationship> contains any number of <record> elements. The <record> type MUST be supported by the registration service.

The MDR may support queries for <record> types that it chooses to not federate through the registration service.

There MAY be multiple <record> elements. The set of passed elements will be considered a complete replacement if the registration service already has data from this MDR about this <relationship>. For example, if the MDR had previously registered this <relationship> with a RunsOn and DependsOn record, and another registration call is made for the same item with only the RunsOn record, then it will be treated as a deletion of the DependsOn record from the federation.

relationshipList/relationship/additionalRecordType

An MDR MAY support through its query interface more record types for a relationship than it federates through the registration service. If so, it MAY indicate the record types per relationship instance by including one or more <additionalRecordType> elements. The <additionalRecordType>/@namespace and <additionalRecordType/@localName attributes together represent the record type. The MDR SHOULD NOT include an <additionalRecordType> if for the same record type it includes a <record>.

5.2.3 Register Response

The outline for the response to a Register operation is as follows.

```
(01) <registerResponse>
(02)   <instanceResponse>
(03)     <instanceId>cmdbf:MdrScopedIdType</instanceId>
(04)       <accepted>
(05)         <alternateInstanceId>
(06)           cmdbf:MdrScopedIdType
(07)         </alternateInstanceId> *
(08)       </accepted> ?
```

```
(09)       <declined>
(10)         <reason>xs:string</reason> *
(11)       </declined> ?
(12)    <instanceResponse> *
(13) </registerResponse>
```

The following describes additional constraints on the outline listed above:

instanceResponse

An element that indicates the action taken for one item or relationship in the Register request. There can be any number of <instanceResponse> elements. There SHOULD be exactly one <instanceResponse> element per item or relationship in the Register request.

instanceResponse/instanceId

One of the <instanceId> elements from the Register request for an item or relationship.

instanceResponse/accepted

An element that indicates that the item or relationship instance was accepted.

Exactly one of either <accepted> or <declined> MUST be present.

instanceResponse/accepted/alternateInstanceId

Zero or more element that contain other IDs by which the item or relationship is known, each one of which is acceptable as a key to select the item or relationship in a query.

instanceResponse/declined

An element that indicates that the item or relationship instance was declined.

Exactly one of either <accepted> or <declined> MUST be present.

instanceResponse/declined/reason

Zero or more strings that contain reason(s) why the registration was declined.

5.2.4 Deregister

The Deregister operation is used by an MDR to notify the Registration service that the data that an MDR has about an item or relationship will no longer be registered.

The outline for the Deregister operation is as follows.

```
(01) <deregisterRequest>
(02)   <mdrId>cmdbf:MdrScopedIdType</mdrId>
(03)   <itemIdList>
(04)     <instanceId>cmdbf:MdrScopedIdType</instanceId> *
(05)   <itemIdList> ?
(06)   <relationshipIdList>
(07)     <instanceId>cmdbf:MdrScopedIdType</instanceId> *
(08)   <relationshipIdList> ?
(09) </deregisterRequest>
```

The following describes additional constraints on the outline listed above:

mdrId

The ID of the MDR deregistering its data. This ID MUST be the ID used when the data was registered using the Register request.

itemIdList

The list of items being deregistered. The list contains any number of <instanceId> elements, though if it contains zero <instanceId> elements, including <itemIdList> serves no purpose.

itemIdList/instanceId

The <instanceId> that serves as a key for the <item>. The <instanceId> MUST be either the <instanceId> from the Register request, or an <alternateInstanceId> from a <registerResponse>. An <instanceId> SHOULD NOT be repeated in the list.

relationshipIdList

The list of relationships being deregistered. The list contains any number of <instanceId> elements, though if it contains zero <instanceId> elements, including <relationshipList> serves no purpose.

relationshipIdList/instanceId

The <instanceId> that serves as a key for the <relationship>. The <instanceId> MUST be either the <instanceId> from the Register request, or an <alternateInstanceId> from a <registerResponse>. An <instanceId> SHOULD NOT be repeated in the list.

5.2.5 Deregister Response

The outline for the response to a Deregister operation is as follows.

```
(01) <deregisterResponse>
(02)   <instanceResponse>
(03)     <instanceId>cmdbf:MdrScopedIdType</instanceId>
(04)     <accepted /> ?
(05)     <declined>
(06)       <reason>xs:string</reason> *
(07)     </declined> ?
(08)   <instanceResponse> *
(09) </deregisterResponse>
```

The following describes additional constraints on the outline listed above:

instanceResponse

An element that indicates the action taken for one item or relationship in the Deregister request. There can be any number of <instanceResponse> elements. There SHOULD be exactly one <instanceResponse> element per item or relationship in the Register request.

instanceResponse/instanceId

The <instanceId> from the Deregister request for an item or relationship.

instanceResponse/accepted

An element that indicates that the item or relationship instance was accepted.

Exactly one of either <accepted> or <declined> MUST be present.

instanceResponse/declined

An element that indicates that the deregistration of the item or relationship instance was declined. An example of when a Deregister request might be declined is when the Registration service does not recognize <instanceId> in the Deregister request.

Exactly one of either <accepted> or <declined> MUST be present.

instanceResponse/declined/reason

Zero or more strings that contain reason(s) why the deregistration was declined.

6. Secure, Reliable, Asynchronous Federation

This specification does not address a number of features that will predictably be required in an operational environment. Such features may be considered largely orthogonal to the operations defined in this specification and will affect no change to their definition. As a reference we list here some features which have been considered by the authors, but have been deemed out of scope. For the convenience of the reader references to other applicable standards are provided. These could be composed into the Web Services environment of an implementer needing or desiring the given functionality.

6.1 Security

Security may encompass the areas of the security of the SOAP messages as well as the authentication of users to a service and the authorization of use of certain resources. For such functionality the reader is referred to the following standards:

- XML Signature Syntax and Processing
- XML Encryption Syntax and Processing
- WS-Security 1.0
- WS-SecureConversation 1.0
- WS-Basic Security Profile 1.0

6.2 Reliability

Reliability is the ability for a sender of a given SOAP message to know that his or her message will be delivered to the correct receiver(s) with no loss of data. This is feature is addressed by the following Web Services standards and specifications:

- WS-ReliableMessaging 1.0, 1.1
- WS-I Reliable Secure Profile (in development)

6.3 Asynchrony

An asynchronous Web Service is one in which a request is made, but a response may not be given until some later time. During this intervening time the requestor is freed to do other operations. In this sense we consider asynchronous Web Services to be of a non-blocking nature. Asynchrony is addressed in the following Web Services standards and specifications:

- WS-Addressing 1.0

7. Acknowledgements

The authors would like to acknowledge the contributions of the CMDB Federation Business Committee whose members included:

Mike Baskey, IBM

Tom Bishop, BMC Software

Josh Cohen, Microsoft
Rob Orr, IBM
Jim Saliba, CA
William Vambenepe, HP
John Van Son, IBM
Yoshinari Abe, Fujitsu

The authors would also like to acknowledge the CMDB Federation Use Case Working Group whose members included:

Mark Johnson, IBM
Pam Molennor, CA
Mike Oitzman, (formerly of) BMC Software
Klaus Wurster, HP

Finally, the authors would like to acknowledge people who have had some involvement in the discussion of the specification at various times during its development, including:

Dale Clark, CA
Ken Huang, BMC Software
Stefan Negritoiu, Microsoft
Tim van Ash, HP
Marshall Whatley, HP
Boris Yanishpolsky, Microsoft

8. References

[RFC 2119]
S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997. (See http://www.ietf.org/rfc/rfc2119.txt.)

[XPath 2.0]
"XML Path Language (XPath) 2.0", W3C Recommendation, January 2007 (See http://www.w3.org/TR/xpath20/.)

Appendix A Detailed UML Class Diagrams

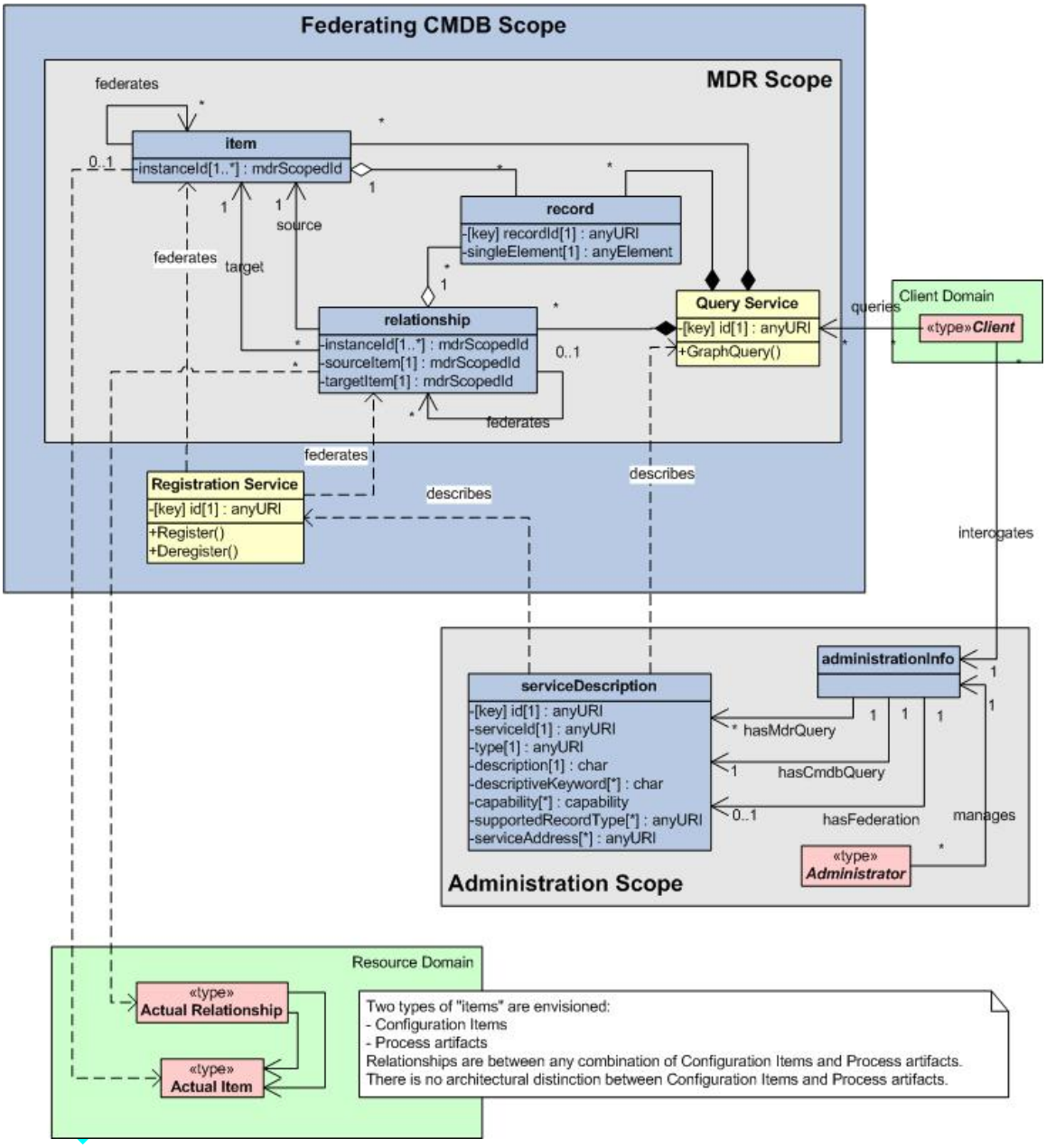


Figure 6 – Overall Class Diagrams

Appendix B XML Schema

A normative copy of the XML Schema [XML Schema Part 1, Part 2] description for this specification can be retrieved from the following address:

http://schemas.cmdbf.org/0-9-5/cmdbfDataModel.xsd

A non-normative copy of the XML Schema description is listed below for convenience.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!--
Copyright Notice
© Copyright 2007 by BMC Software, CA, Fujitsu, Hewlett-Packard, IBM,
and Microsoft. All Rights Reserved.

## Any permissions and license grants would go here ##

THE SPECIFICATION IS PROVIDED "AS IS," AND THE AUTHORS MAKE NO
REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT
LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE
SPECIFICATION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION
OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS,
TRADEMARKS OR OTHER RIGHTS.

THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL,
INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY
USE OR DISTRIBUTION OF THE SPECIFICATION.

The name and trademarks of the Authors may NOT be used in any manner,
including advertising or publicity pertaining to the Specification or
its contents without specific, written prior permission. Title to
copyright in the Specification will at all times remain with the
Authors.

No other rights are granted by implication, estoppel or otherwise.
-->


<xs:schema targetNamespace="http://schemas.cmdbf.org/0-9-5/datamodel"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" blockDefault="#all"
  xmlns:cmdbf="http://schemas.cmdbf.org/0-9-5/datamodel">

  <!-- Message Global Element Declarations -->

  <xs:element name="query" type="cmdbf:QueryType" />
  <xs:element name="queryResult" type="cmdbf:QueryResultType" />
  <xs:element name="registerRequest"
```

```
    type="cmdbf:RegisterRequestType" />
  <xs:element name="registerResponse"
    type="cmdbf:RegistrationServiceResponseType" />
  <xs:element name="deregisterRequest"
    type="cmdbf:DeregisterRequestType" />
  <xs:element name="deregisterResponse"
    type="cmdbf:RegistrationServiceResponseType" />

  <!-- Query Request Definitions -->
  <xs:complexType name="QueryType">
    <xs:sequence>
      <xs:element name="itemTemplate" type="cmdbf:ItemTemplateType"
        minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="relationshipTemplate"
        type="cmdbf:RelationshipTemplateType" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="ItemTemplateType">
    <xs:sequence>
      <xs:element name="instanceIdSelector"
        type="cmdbf:MdrScopedIdType"
        minOccurs="0" maxOccurs="1" />
      <xs:element name="propertyValueSelector"
        type="cmdbf:PropertyValueSelectorType" minOccurs="0"
        maxOccurs="unbounded" />
      <xs:element name="xpath1Selector" type="cmdbf:XPath1SelectorType"
        minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="recordTypeSelector" type="cmdbf:QNameType"
        minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="propertySubsetDirective"
        type="cmdbf:PropertySubsetDirectiveType" minOccurs="0"
        maxOccurs="1" />
      <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##other"
        processContents="lax" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required" />
    <xs:attribute name="dropDirective" type="xs:boolean" use="optional"
      default="false" />
  </xs:complexType>

  <xs:complexType name="RelationshipTemplateType">
    <xs:sequence>
      <xs:element name="instanceIdSelector"
```

```
1674        type="cmdbf:MdrScopedIdType"
1675        minOccurs="0" maxOccurs="1" />
1676      <xs:element name="propertyValueSelector"
1677        type="cmdbf:PropertyValueSelectorType" minOccurs="0"
1678        maxOccurs="unbounded" />
1679      <xs:element name="xpath1Selector" type="cmdbf:XPath1SelectorType"
1680        minOccurs="0" maxOccurs="unbounded" />
1681      <xs:element name="recordTypeSelector" type="cmdbf:QNameType"
1682        minOccurs="0" maxOccurs="unbounded" />
1683      <xs:element name="propertySubsetDirective"
1684        type="cmdbf:PropertySubsetDirectiveType" minOccurs="0"
1685        maxOccurs="1" />
1686      <xs:element name="source" type="cmdbf:RelationshipRefType"
1687        minOccurs="0" maxOccurs="1" />
1688      <xs:element name="target" type="cmdbf:RelationshipRefType"
1689        minOccurs="0" maxOccurs="1" />
1690      <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##other"
1691        processContents="lax" />
1692    </xs:sequence>
1693    <xs:attribute name="id" type="xs:ID" use="required" />
1694    <xs:attribute name="dropDirective" type="xs:boolean" use="optional"
1695      default="false" />
1696  </xs:complexType>
1697
1698  <xs:complexType name="RelationshipRefType">
1699    <xs:attribute name="ref" type="xs:IDREF" use="required" />
1700    <xs:attribute name="minimum" type="xs:int" />
1701    <xs:attribute name="maximum" type="xs:int" />
1702  </xs:complexType>
1703
1704  <xs:complexType name="PropertyValueSelectorType">
1705    <xs:sequence>
1706      <xs:element name="equal" type="cmdbf:EqualOperatorType"
1707        minOccurs="0" maxOccurs="unbounded" />
1708      <xs:element name="less" type="cmdbf:ComparisonOperatorType"
1709        minOccurs="0" maxOccurs="1" />
1710      <xs:element name="lessOrEqual"
1711        type="cmdbf:ComparisonOperatorType"
1712        minOccurs="0" maxOccurs="1" />
1713      <xs:element name="greater" type="cmdbf:ComparisonOperatorType"
1714        minOccurs="0" maxOccurs="1" />
1715      <xs:element name="greaterOrEqual"
1716        type="cmdbf:ComparisonOperatorType" minOccurs="0"
1717        maxOccurs="1" />
1718      <xs:element name="contains" type="cmdbf:StringOperatorType"
```

```
1719          minOccurs="0" maxOccurs="unbounded" />
1720        <xs:element name="like" type="cmdbf:StringOperatorType"
1721          minOccurs="0" maxOccurs="unbounded" />
1722        <xs:element name="isNull" type="cmdbf:NullOperatorType"
1723          minOccurs="0" maxOccurs="1" />
1724        <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##other"
1725          processContents="lax" />
1726      </xs:sequence>
1727      <xs:attribute name="namespace" type="xs:anyURI" use="required" />
1728      <xs:attribute name="localName" type="xs:NCName" use="required" />
1729      <xs:attribute name="matchAny" type="xs:boolean" use="optional"
1730        default="false" />
1731    </xs:complexType>
1732
1733    <xs:complexType name="XPath1SelectorType">
1734      <xs:sequence>
1735        <xs:element name="prefixMapping"
1736          type="cmdbf:PrefixMappingType" />
1737        <xs:element name="xpathExpression" type="xs:string" />
1738      </xs:sequence>
1739    </xs:complexType>
1740
1741    <xs:complexType name="PrefixMappingType">
1742      <xs:attribute name="prefix" type="xs:NCName" use="required" />
1743      <xs:attribute name="namespace" type="xs:anyURI" use="required" />
1744    </xs:complexType>
1745
1746    <xs:complexType name="PropertySubsetDirectiveType">
1747      <xs:sequence>
1748        <xs:element name="selectedProperty" type="cmdbf:QNameType"
1749          minOccurs="0" maxOccurs="unbounded" />
1750      </xs:sequence>
1751    </xs:complexType>
1752
1753    <!-- property value selectors -->
1754    <xs:complexType name="ComparisonOperatorType">
1755      <xs:simpleContent>
1756        <xs:extension base="xs:anySimpleType">
1757          <xs:attribute name="negate" type="xs:boolean" use="optional"
1758            default="false" />
1759        </xs:extension>
1760      </xs:simpleContent>
1761    </xs:complexType>
1762
1763    <xs:complexType name="StringOperatorType">
```

```
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="caseSensitive" type="xs:boolean"
          use="optional" default="true" />
        <xs:attribute name="negate" type="xs:boolean" use="optional"
          default="false" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="EqualOperatorType">
    <xs:simpleContent>
      <xs:extension base="xs:anySimpleType">
        <xs:attribute name="caseSensitive" type="xs:boolean"
          use="optional" default="true" />
        <xs:attribute name="negate" type="xs:boolean" use="optional"
          default="false" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="NullOperatorType">
    <xs:attribute name="negate" type="xs:boolean" use="optional"
      default="false" />
  </xs:complexType>

  <!-- Query Response definition -->
  <xs:complexType name="QueryResultType">
    <xs:sequence>
      <xs:element name="nodes" type="cmdbf:NodesType" minOccurs="0"
        maxOccurs="unbounded" />
      <xs:element name="edges" type="cmdbf:EdgesType" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="NodesType">
    <xs:sequence>
      <xs:element ref="cmdbf:item" minOccurs="1"
        maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="templateId" type="xs:ID" use="required" />
  </xs:complexType>

  <xs:complexType name="EdgesType">
```

```
    <xs:sequence>
      <xs:element ref="cmdbf:relationship" minOccurs="1"
        maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="templateId" type="xs:ID" use="required" />
  </xs:complexType>

  <!-- Registration Service -->
  <xs:complexType name="RegisterRequestType">
    <xs:sequence>
      <xs:element name="mdrId" type="xs:anyURI" />
      <xs:element name="itemList" type="cmdbf:ItemListType"
        minOccurs="0" maxOccurs="1" />
      <xs:element name="relationshipList"
        type="cmdbf:RelationshipListType" minOccurs="0"
        maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="ItemListType">
    <xs:sequence>
      <xs:element ref="cmdbf:item" minOccurs="1"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="RelationshipListType">
    <xs:sequence>
      <xs:element ref="cmdbf:relationship" minOccurs="1"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="DeregisterRequestType">
    <xs:sequence>
      <xs:element name="mdrId" type="xs:anyURI" />
      <xs:element name="itemIdList" type="cmdbf:MdrScopedIdListType"
        minOccurs="0" maxOccurs="1" />
      <xs:element name="relationshipIdList"
        type="cmdbf:MdrScopedIdListType" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="MdrScopedIdListType">
    <xs:sequence>
      <xs:element ref="cmdbf:instanceId" minOccurs="1"
```

```
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="RegistrationServiceResponseType">
    <xs:sequence>
      <xs:element name="instanceResponse"
        type="cmdbf:InstanceResponseType" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="InstanceResponseType">
    <xs:sequence>
      <xs:element name="instanceId" type="cmdbf:MdrScopedIdType"
        minOccurs="1" maxOccurs="1" />
      <xs:element name="accepted" type="cmdbf:AcceptedType"
        maxOccurs="1" minOccurs="0" />
      <xs:element name="declined" type="cmdbf:DeclinedType"
        maxOccurs="1" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="AcceptedType">
    <xs:sequence>
      <xs:element name="alternativeInstanceId"
        type="cmdbf:MdrScopedIdType" maxOccurs="unbounded"
        minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="DeclinedType">
    <xs:sequence>
      <xs:element name="reason" type="xs:string" maxOccurs="unbounded"
        minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

  <!-- Shared elements definition -->
  <xs:element name="item" type="cmdbf:ItemType" />
  <xs:complexType name="ItemType">
    <xs:sequence>
      <xs:element ref="cmdbf:record" minOccurs="0"
        maxOccurs="unbounded" />
      <xs:element ref="cmdbf:instanceId" minOccurs="1"
```

```
        maxOccurs="unbounded" />
      <xs:element name="additionalRecordType" type="cmdbf:QNameType"
        minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:element name="relationship" type="cmdbf:RelationshipType" />
  <xs:complexType name="RelationshipType">
    <xs:sequence>
      <xs:element name="sourceItem" type="cmdbf:MdrScopedIdType"
        minOccurs="1" maxOccurs="1" />
      <xs:element name="targetItem" type="cmdbf:MdrScopedIdType"
        minOccurs="1" maxOccurs="1" />
      <xs:element ref="cmdbf:record" minOccurs="0"
        maxOccurs="unbounded" />
      <xs:element ref="cmdbf:instanceId" minOccurs="1"
        maxOccurs="unbounded" />
      <xs:element name="additionalRecordType" type="cmdbf:QNameType"
        maxOccurs="unbounded" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

  <xs:element name="record" type="cmdbf:RecordType" />
  <xs:complexType name="RecordType">
    <xs:sequence>
      <xs:any namespace="##other" processContents="lax" />
    </xs:sequence>
    <xs:attribute name="recordId" type="xs:anyURI" use="required" />
  </xs:complexType>

  <xs:element name="instanceId" type="cmdbf:MdrScopedIdType" />
  <xs:complexType name="MdrScopedIdType">
    <xs:sequence>
      <xs:element name="mdrId" type="xs:anyURI" minOccurs="1"
        maxOccurs="1" />
      <xs:element name="localId" type="xs:anyURI" minOccurs="1"
        maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="QNameType">
    <xs:attribute name="namespace" type="xs:anyURI" use="required" />
    <xs:attribute name="localName" type="xs:NCName" use="required" />
  </xs:complexType>
</xs:schema>
```

A normative copy of the WSDL [WSDL 1.1] description for this specification can be retrieved from the following addresses:

```
http://schemas.cmdbf.org/0-9-5/cmdbfQuery.wsdl
http://schemas.cmdbf.org/0-9-5/cmdbfRegistration.wsdl
```

A non-normative copy of the WSDL descriptions are listed below for convenience.

8.1 Query Service WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<!--
Copyright Notice
(c) 2007 BMC Software, CA, Fujitsu, Hewlett-Packard Development Company
(HP), International Business Machines Corporation (IBM), and Microsoft
Corporation. All rights reserved.
-->
<wsdl:definitions
  targetNamespace="http://schemas.cmdbf.org/0-9-5/query"
  xmlns:tns="http://schemas.cmdbf.org/0-9-5/query"
  xmlns:cmdbf="http://schemas.cmdbf.org/0-9-5/datamodel"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>
    <xs:schema
      targetNamespace="http://schemas.cmdbf.org/0-9-5/datamodel">
      <xs:include
        schemaLocation=
        "http://schemas.cmdbf.org/0-9-5/cmdbfDataModel.xsd" />
    </xs:schema>
  </wsdl:types>

  <wsdl:message name="QueryRequest">
    <wsdl:part name="body" element="cmdbf:query" />
  </wsdl:message>

  <wsdl:message name="QueryResponse">
    <wsdl:part name="body" element="cmdbf:queryResult" />
  </wsdl:message>

  <wsdl:portType name="QueryPortType">
    <wsdl:operation name="GraphQuery">
      <wsdl:input message="tns:QueryRequest" />
      <wsdl:output message="tns:QueryResponse" />
    </wsdl:operation>
  </wsdl:portType>
```

```
</wsdl:definitions>
```

8.2 Registration Service WSDL

```
<?xml version='1.0' encoding='UTF-8' ?>
<!--
Copyright Notice
(c) 2007 BMC Software, CA, Fujitsu, Hewlett-Packard Development Company
(HP), International Business Machines Corporation (IBM), and Microsoft
Corporation. All rights reserved.
-->
<wsdl:definitions
  targetNamespace="http://schemas.cmdbf.org/0-9-5/registration"
  xmlns:tns="http://schemas.cmdbf.org/0-9-5/registration"
  xmlns:cmdbf="http://schemas.cmdbf.org/0-9-5/datamodel"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>
    <xs:schema
      targetNamespace="http://schemas.cmdbf.org/0-9-5/datamodel">
      <xs:include
        schemaLocation=
        "http://schemas.cmdbf.org/0-9-5/cmdbfDataModel.xsd" />
    </xs:schema>
  </wsdl:types>

  <wsdl:message name="RegisterRequest">
    <wsdl:part name="body" element="cmdbf:registerRequest" />
  </wsdl:message>
  <wsdl:message name="RegisterResponse">
    <wsdl:part name="body" element="cmdbf:registerResponse" />
  </wsdl:message>

  <wsdl:message name="DeregisterRequest">
    <wsdl:part name="body" element="cmdbf:deregisterRequest" />
  </wsdl:message>
  <wsdl:message name="DeregisterResponse">
    <wsdl:part name="body" element="cmdbf:deregisterResponse" />
  </wsdl:message>

  <wsdl:portType name="RegistrationPortType">
    <wsdl:operation name="Register">
```

```
      <wsdl:input message="tns:RegisterRequest" />
      <wsdl:output message="tns:RegisterResponse" />
    </wsdl:operation>

    <wsdl:operation name="Deregister">
      <wsdl:input message="tns:DeregisterRequest" />
      <wsdl:output message="tns:DeregisterResponse" />
    </wsdl:operation>
  </wsdl:portType>

</wsdl:definitions>
```