Document Number: DSP0252

Date: 2010-04-22

Version: 1.0.1

Configuration Management Database (CMDB) Federation Specification

Document Type: Specification

Document Status: DMTF Standard

Document Language: E

Copyright Notice

Copyright © 2009-2010 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. Members and non-members may reproduce DMTF specifications and documents, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted.

Implementation of certain elements of this standard or proposed standard may be subject to third party patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose, or identify any or all such third party patent right, owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third party patent rights, or for such party's reliance on the standard or incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any party implementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.

For information about patents held by third-parties which have notified the DMTF that, in their opinion, such patent may relate to or impact implementations of DMTF standards, visit http://www.dmtf.org/about/policies/disclosures.php.

CONTENTS

Figures

Tables

Foreword

The *Configuration Management Database (CMDB) Federation Specification* (DSP0252) was prepared by the CMDB Federation Working Group.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability.

Acknowledgements

The CMDB Federation Working Group wishes to acknowledge the following people.

Authors:

- Forest Carlisle – CA
- Jacob Eisinger – IBM
- Mark Johnson – IBM (Editor)
- Vincent Kowalski – BMC Software
- Jishnu Mukerji – HP
- David Snelling – Fujitsu
- William Vambenepe – Oracle
- Marv Waschke – CA
- Van Wiles – BMC Software

Conventions

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.
- The following 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 characters 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.
- xs:any and xs:anyAttribute indicate points of extensibility. Additional children or attributes may be added at the indicated extension points but shall not contradict the semantics of the parent 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.

- Ellipses (that is, "...") indicate that details are omitted for simplicity, and a further explanation is provided below.
- XML namespace prefixes are used to indicate the namespace of the element being defined or referenced.

Introduction

Many organizations are striving to base IT management on a Configuration Management Database (CMDB). A CMDB contains data describing the following entities:

- managed resources, such as computer systems and application software
- process artifacts, such as incident, problem, and change records
- relationships among managed resources and process artifacts

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, as shown in Figure 1.

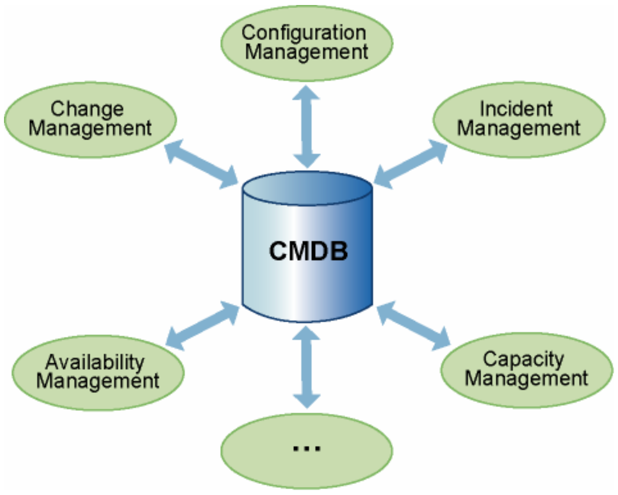


Figure 1 – CMDB as the Foundation for IT Management Processes

However, in practice it is challenging to implement such a CMDB 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 selects the data to be included in a CMDB 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 (for example, a computer system, an application, or a router), process artifacts (for example, an incident record or a change record), and relationships between configuration items and process artifacts. The architecture describes a logical model and does not necessarily reflect a physical manifestation.

Objectives

This section describes the functionality and target IT environment that this specification supports.

Functionality

The federated CMDB that would result from using this specification would 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 would use a query service defined in the specification to access aggregated or non-aggregated views. Data repositories would use the services described in the specification to provide the aggregated view.

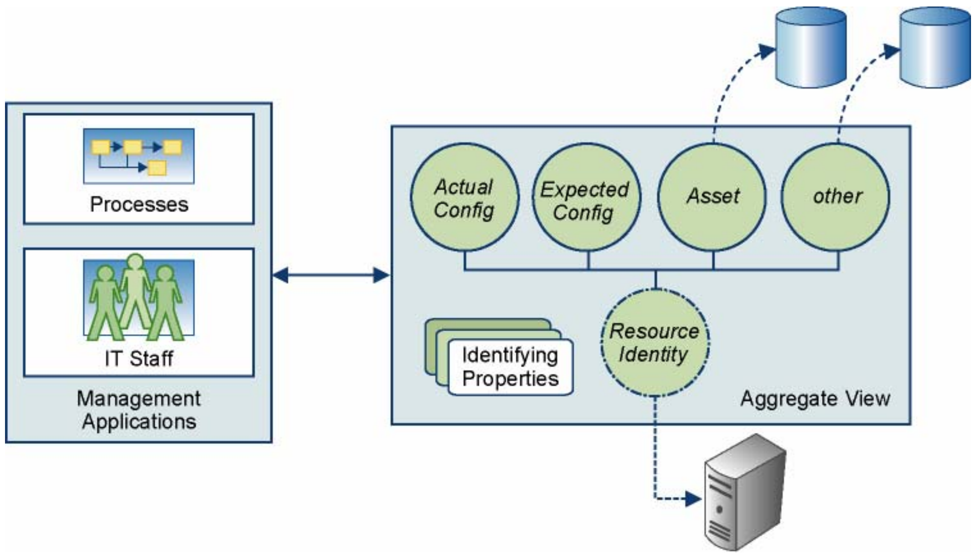


Figure 2 – Example Aggregate View from a Federated CMDB

The federated CMDB could support the following scenarios. (However, the scenarios that a federated CMDB supports are left entirely to the discretion of each implementation.)

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

Target IT Environment

This specification is intended to address requirements in IT environments that have the following characteristics:

- There are strong requirements to consolidate into one or more databases (logical 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 are 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. Only after an extended period of time would it be realistic to expect all of the existing management tools to be modified to require and utilize new consolidated databases.

Out-of-Scope Implementation Details

The following implementation details are outside the scope of this 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 the means to represent changes as or after they are made, but not to be the agent that makes the change.

Technological Assumptions

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

Underlying Technology

The technologies behind CMDBs include Web Services and database management systems.

Web Services

Although the interface specification contained herein is generic, it assumes 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.

Database Management Systems

In general practice CMDBs are implemented using commercially available database technology. Although this specification is 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. The specification focuses on 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.

Configuration Management Database (CMDB) Federation Specification

1 Scope

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), or as the federated repository that is the heart of a Configuration Management System, as described in the ITIL best practices, version 3. For brevity, the remainder of the document uses the term CMDB, even when the term Configuration Management System would be at least as appropriate. 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.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IETC RFC 2616, *Hypertext Transfer Protocol – HTTP/1.1*, June 1999, http://www.ietf.org/rfc/rfc2616.txt

ISO 8601, Third edition, 2004-12-01, *Data elements and interchange formats — Information interchange — Representation of dates and times*

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards,* http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

ITSMF, *ITIL Version 3 Glossary of Terms and Definitions*, May 2007, http://www.itsmf.co.uk/web/FILES/Publications/ITILV3_Glossary_English_v1_2007.pdf

W3C, *Simple Object Access Protocol (SOAP) 1.1*, May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

W3C, *SOAP Version 1.2 Part 1: Messaging Framework*, April 2007, http://www.w3.org/TR/2006/REC-xml-20060816/

W3C, *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, September 2006, http://www.w3.org/TR/2006/REC-xml-20060816/

W3C, *XML Schema 1.0 Part 1: Structures (Second Edition)*, October 2004, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

W3C, *XML Schema 1.0 Part 2: Datatypes (Second Edition)*, October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

W3C, *XML Path Language (XPath) 1.0*, November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116

W3C, *XML Path Language (XPath) 2.0*, January 2007, http://www.w3.org/TR/2007/REC-xpath20-20070123/

W3C, *XQuery 1.0 and XPath 2.0 Functions and Operators*, January 2007, http://www.w3.org/TR/xquery-operators/

W3C, *XSLT 2.0 and XQuery 1.0 Serialization*, January 2007, http://www.w3.org/TR/xslt-xquery-serialization/

W3C, *Web Services Description Language (WSDL) 1.1*, March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards,* http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

3 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

3.1 Requirements Terms

3.1.1
can
used for statements of possibility and capability, whether material, physical, or causal

3.1.2
cannot
used for statements of possibility and capability, whether material, physical or causal

3.1.3
conditional
indicates requirements to be followed strictly in order to conform to the document when the specified conditions are met

3.1.4
mandatory
indicates requirements to be followed strictly in order to conform to the document and from which no deviation is permitted

3.1.5
may
indicates a course of action permissible within the limits of the document

3.1.6
need not
indicates a course of action permissible within the limits of the document

3.1.7
optional
indicates a course of action permissible within the limits of the document

3.1.8
shall
indicates requirements to be followed strictly in order to conform to the document and from which no deviation is permitted

3.1.9
shall not

indicates requirements to be followed strictly in order to conform to the document and from which no deviation is permitted

3.1.10
should

indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

3.1.11
should not

indicates that a certain possibility or course of action is deprecated but not prohibited

3.2 Background Terminology

This 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 Information Technology Infrastructure Library (ITIL). ITIL is not a standard and does not provide normative definitions of terms. However, the ITIL version 3 glossary is quoted below as representative of the ITIL position.

3.2.1
configuration item
CI

a basic tangible or intangible entity in a configuration management solution such as a CMDB.

ITIL version 3 defines a CI as follows:

> "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."

3.2.2
configuration management database
CMDB

ITIL defines a CMDB as follows:

> "A database used to store Configuration Records throughout their Lifecycle. The Configuration Management System maintains one or more CMDBs, and each CMDB stores *Attribute*s 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.

3.2.3
configuration management system
CMS

ITIL defines (in part) a configuration management system as follows:

> "A set of tools and databases that are used to manage an IT Service Provider's Configuration data. The CMS also includes information about Incidents, Problems, Known Errors, Changes

> and Releases; and may contain data about employees, Suppliers, locations, Business Units, Customers and Users."

A configuration management system is presumed to be a federation of CMDBs and other management data repositories. The federated CMDB described in this specification is a good match with the database requirements of a configuration management system.

3.2.4
configuration record

ITIL defines a configuration record as follows:

> 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.

3.2.5
federated CMDB

a combination of multiple management data repositories (MDRs), at least one of which federates the others, into an aggregate view of management data.

NOTE: 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.

3.2.6
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.

3.2.7
graph

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.

3.2.8
identity

a set of qualities or characteristics that distinguish an entity from other entities of the same or different types. This set of qualities may be called the "identifying properties" of the real world entity for which the CMDB contains data.

3.2.9
Information Technology Infrastructure Library
ITIL

a framework of best practices for delivering IT services. Two versions of ITIL are commonly in use: version 2 released in 2000 and version 3 released in 2007. Because ITIL version 3 has not yet superseded version 2 in practice, both versions have been considered in preparing this specification. A CMDB is a key component in the ITIL best practices.

4 Symbols and Abbreviated Terms

4.1
CI
configuration item

4.2
CMDB
configuration management database

4.3
CMDBf
configuration management database federation

4.4
CMS
configuration management system

4.5
ITIL
Information Technology Infrastructure Library

4.6
MDR
management data repository

4.7
SACM
service asset and configuration management

4.8
SLA
service level agreement

4.9
WSDL
Web Service Definition Language

5 Architecture

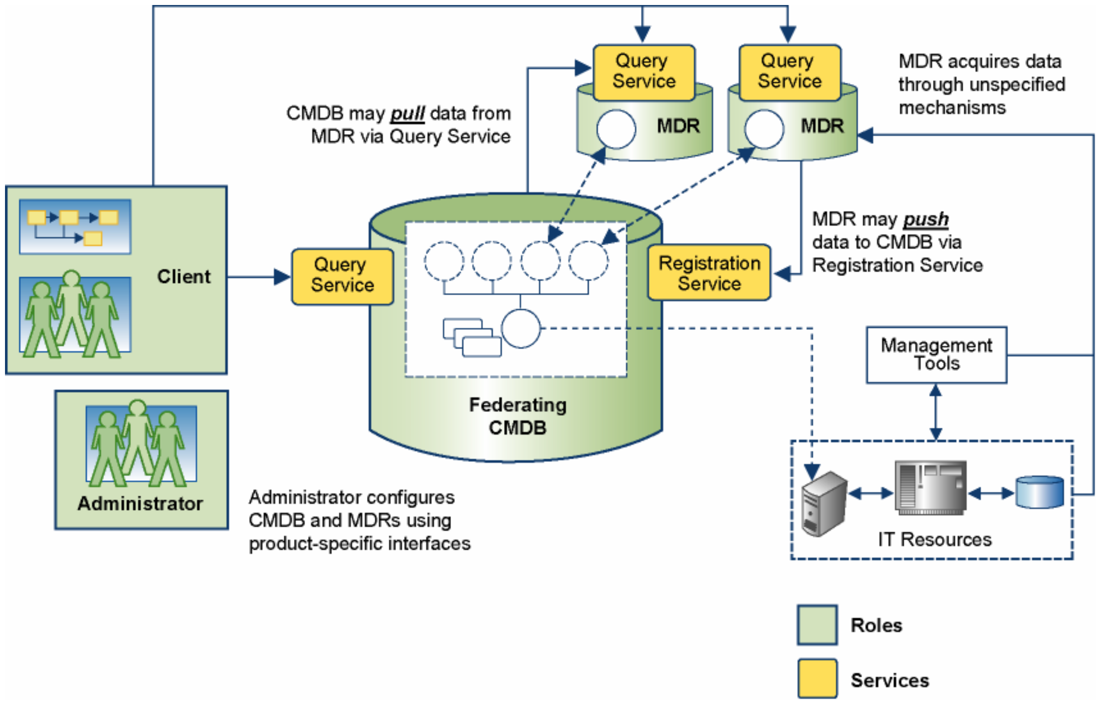
5.1 Overview

As shown in Figure 3, the architecture defines the following four roles:

- management data repository
- federating CMDB
- client
- administrator

These roles implement or use the following two services:

- Query Service
- Registration Service



Query Service
Query Service
MDR acquires data through unspecified mechanisms
CMDB may *pull* data from MDR via Query Service
MDR
MDR
Client
Query Service
Registration Service
MDR may *push* data to CMDB via Registration Service
Management Tools
Federating CMDB
Administrator
Administrator configures CMDB and MDRs using product-specific interfaces
IT Resources
Roles
Services

Figure 3 – CMDB Roles and Services

5.2 Roles

5.2.1 Management Data Repository (MDR)

An MDR provides data about managed resources (for example, computer systems, application software, and buildings), process artifacts (for example, 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, but the means can include acquiring data directly from instrumented resources or indirectly through management tools.

Each MDR has an ID that is unique within (at least) a group of federated MDRs, and preferably globally unique.

5.2.2 Federating CMDB

A federating CMDB is an MDR with additional capabilities. It federates data from MDRs; it 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.

5.2.3 Client

A client is a consumer of management data, either directly from an MDR or through 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.

5.2.4 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 not defined in this architecture and that may be specific to each tool that acts in the MDR or federating CMDB role.

5.3 Services Overview

The subsequent clauses explain service types, federation modes, and service usage patterns.

5.3.1 Service Types

The architecture defines two services: Query Service and Registration Service. A service has an implementor and a client (caller).

5.3.1.1 Query Service

Both MDRs and federating CMDBs may implement the Query Service to make data available to Clients. Queries may select and return items, relationships, or graphs containing items and relationships, and the data records associated with each item and relationship. An MDR or a federating CMDB may declare the data record types that its Query Service supports.

5.3.1.2 Registration Service

A federating CMDB may implement the Registration Service. An MDR may call the Registration Service to register data that it has available for federation. A federating CMDB may declare the data types that its Registration Service supports. An MDR maps its data to the supported types.

5.3.2 Federation Modes

The two modes available to federate data are push mode and pull mode. A federating CMDB shall use at least one mode and may use both.

5.3.2.1 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 other properties that describe the item or relationship state.

5.3.2.2 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.

5.3.3 Service Usage Patterns

Table 1 lists the service usage patterns for the roles described in 5.2 that implement or use the services.

Table 1 – Service Usage Patterns

Pattern (Role + Mode)	Query Service		Registration Service	
	Implementation	Client	Implementation	Client
Federating CMDB – Push Mode	Required	Optional	Required	N/A
Federating CMDB – Pull Mode	Required	Required	N/A	N/A
MDR – Push Mode	Optional	N/A	N/A	Required
MDR – Pull Mode	Required	N/A	N/A	N/A
Client (external)	N/A	Required	N/A	N/A

5.4 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 typically used for identity reconciliation:

- 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 about 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 or provides access to through 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.

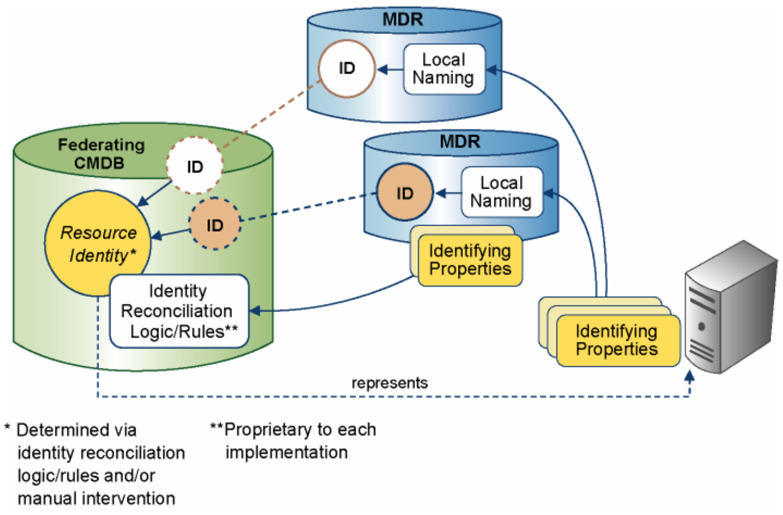

MDR
ID
Local Naming
MDR
ID
Local Naming
Identifying Properties
Federating CMDB
ID
ID
Resource Identity*
Identity Reconciliation Logic/Rules**
Identifying Properties
represents
* Determined via identity reconciliation logic/rules and/or manual intervention
**Proprietary to each implementation

Figure 4 – Identity Reconciliation

The federating CMDB performs this identity 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 that 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.

5.5 Data Elements Overview

Subsequent clauses provide an overview of the elements used to organize the data in MDRs and federating CMDBs.

5.5.1 Managed Data

The architecture defines three elements that organize the data that repositories exchange: item, relationship, and record.

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. (In other words, 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 (for example, existence of a relationship between two items) and based on properties of the items and relationships (for example, requirements for a certain value of a given record property or a certain type for the item and relationship).

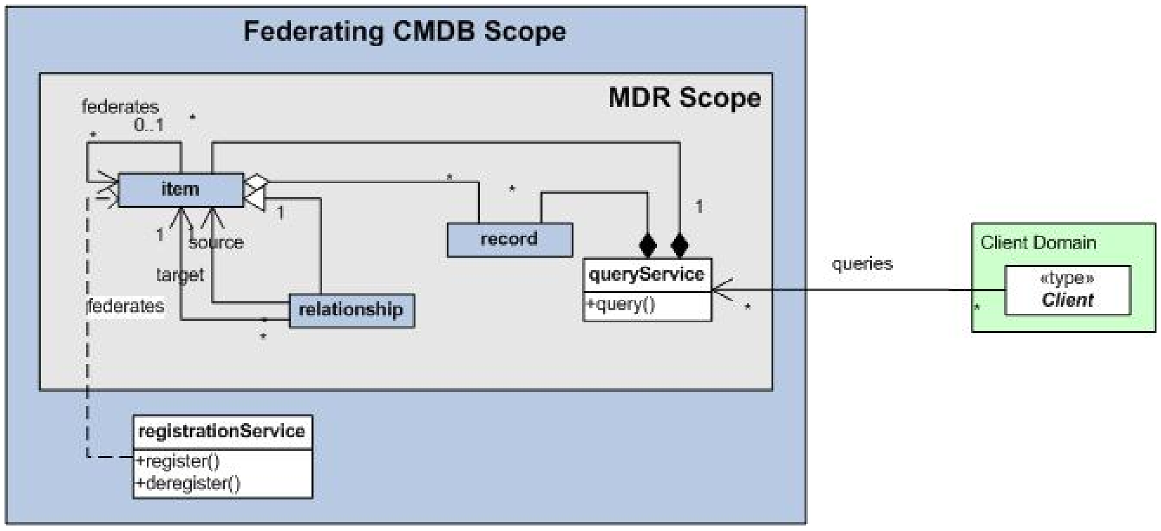


Figure 5 – Data and Services Overview

5.5.1.1 Item

An item represents a managed resource (for example, computer systems, application software, and buildings) or a process artifact (for example, an incident record and request for change form). With this definition, "item" is a superset of the "configuration item" term defined in ITIL. Formally:

- Each item shall have at least one ID that is unique within the scope of the MDR that contains it and that serves as a key.
- After an ID has been assigned to an item, it may be used in any situation requiring an ID.

- After an ID has been assigned to an item, it shall never refer to anything except the original item.
- An instance ID of an item is the composition of the unique MDR ID and the unique item ID assigned by that MDR. The instance ID is therefore unique within the group of federated repositories.

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; or when an ID changes for any other reason.

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.

5.5.1.2 Relationship

A relationship represents a connection from a source item to a target item. Examples include software "runs" on an operating system, an operating system is "installed" on a computer system, an incident record "affects" a computer system, and service "uses" (another) service. Relationships have the following characteristics:

- A relationship links exactly two items, one the source and one the target, and provides information pertaining to that relationship.
- A relationship is a subclass of an item (though the relationship XML schema does not formally extend the item XML schema), and has all the characteristics of an item. For example, each relationship shall have an ID that is unique within the scope of the MDR that contains it and that serves as a key, and a reconciled relationship may have more than one ID.

5.5.1.2.1 Relationship Roles

The two endpoints of a relationship are not equivalent. In the general case, items at these endpoints play different roles in the relationship. Some relationships may not have any such semantic distinction because they are symmetrical (e.g. "sibling"), but this is not the general case. An example of the general case is an "employment" relationship which links an "employer" to an "employee".

CMDBf designates the endpoints as "source" and "target" to distinguish them. There are no semantics attached to these terms, other than a convention that when a relationship is represented graphically by an arrow, the arrow goes from the source to the target. The relationship record type (see 5.5.1.3) documentation should describe the role semantics of the "source" and "target" endpoints.

5.5.1.3 Record

A record contains properties that describe an item or relationship. Records have the following characteristics:

- A record is associated with exactly one item or relationship.

- A record may contain properties that are useful to identify the item or relationship, or it may contain other properties that describe the item or relationship.
- Several records, possibly of various types, may be associated with the same item or relationship.

Records may differ from other records for various reasons, including types of data (for example, asset versus configuration), different sets of properties from different providers, different versions, and expected versus 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 may have the following metadata properties that describe the record itself (as opposed to properties that describe the item or relationship):

- an ID that is unique within the scope of its associated item or relationship and that serves as a key (optional if there is only one record for the item or relationship)
- the date/time the record was last modified (optional)
- a baseline ID that may be used to indicate the expected (authorized) configuration baseline this record represents (optional)
- a snapshot ID that may be used to indicate the configuration observations this record represents (optional)

Each record has exactly one "record type". Note that a record type may extend one or more other record types, as described in 8.2.2.3. A record type is:

- A characterization of an item or relationship.
- A collection of properties that can be used to describe an item or relationship. The properties may be simple or complex XML elements.
- A record type may be used in a query to limit the items or relationships returned by a query operation to instances with a record considered by the query service to be of the requested type.

A record type may also be the QName of the first child of a record element in a query response.

5.5.2 Common Data Element Types

The cmdbf:MdrScopedIdType is used in several places to identify an item or relationship. It is described here for convenience so other sections of this document may refer to it without repeating the definition.

The `<instanceId>` element is of the type of cmdbf:MdrScopedIdType. The pseudo-schema of the `<instanceId>` element is as follows:

```
<instanceId>
  <mdrId>xs:anyURI</mdrId>
  <localId>xs:anyURI</localId>
</instanceId>
```

This can be abbreviated in a pseudo schema as the following:

```
<instanceId>cmdbf:MdrScopedIdType</instanceId>
```

The cmdbf:MdrScopedIdType 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.

Every `<record>` element has exactly one child element of unrestricted content (which is typically used to describe the item or relationship with which the record is associated), followed by an optional (if there is only one record associated with the item or relationship) `<recordMetadata>` element that contains common information about the record itself.

The `<recordMetadata>` element may contain these properties:

- recordId: the unique ID of the record in the MDR. If there is more than one record for an item or a relationship, the recordId is required.
- lastModified: the time/date the record was last modified in ISO 8601 format. The applicable time zone or UTC shall be indicated.
- baselineId: the name or other identifier used to group records into a particular baseline configuration. A value of "0" indicates that this record is not part of any baseline configuration.
- snapshotId: the name or other identifier used to group records observed in a configuration snapshot (discovery). A value of "0" indicates that this record is not part of any snapshot configuration.
- extensibility elements: additional metadata elements not defined by the specification may also be included

6 Query Service

6.1 Overview

The Query Service can be provided by MDRs and federating CMDBs (see Table 1 – Service Usage Patterns on page 18). 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 GraphQuery operation 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 subsequent subclauses provide a more complete description of the request and response messages for the GraphQuery operation. Examples are provided in ANNEX D.

6.2 GraphQuery Operation Outline

A GraphQuery request consists of a `<query>` element that contains `<itemTemplate>` and `<relationshipTemplate>` elements. Content selectors and constraints can be used inside `<itemTemplate>` or `<relationshipTemplate>` elements, and have the same form in both.

In addition to constraints, `<relationshipTemplate>` elements also contain a `<sourceTemplate>` and a `<targetTemplate>` element. These elements each point (using the xs:ID/xs:IDREF mechanism) to an `<itemTemplate>`.

The pseudo-schema for the payload of a GraphQuery request is as follows:

```
<query>
  <itemTemplate id="xs:ID" suppressFromResult="xs:boolean" ?>
    (<contentSelector ...>...</contentSelector> ?
    <instanceIdConstraint>...</instanceIdConstraint> ?
    <recordConstraint>
      <recordType ... /> *
      <propertyValue ...>...</propertyValue> *
      <xpathConstraint>...</xpathConstraint> ?
    </recordConstraint> *)
    xs:any
  </itemTemplate> *
  <relationshipTemplate id="xs:ID" suppressFromResult="xs:boolean" ?>
    (<contentSelector ...>...</contentSelector> ?
    <instanceIdConstraint>...</instanceIdConstraint> ?
    <recordConstraint>
      <recordType>...</recordType> *
      <propertyValue>...</propertyValue> *
      <xpathConstraint>...</xpathConstraint> ?
    </recordConstraint> *)
    <sourceTemplate ref="xs:IDREF" minimum="xs:int"?
     maximum="xs:int"?/> ?
    <targetTemplate ref="xs:IDREF" minimum="xs:int"?
     maximum="xs:int"?/> ?
    <depthLimit ... /> ?
    xs:any
  </relationshipTemplate> *
</query>
```

The syntax and semantics for each constraint element are provided in later clauses (for `<instanceIdConstraint>` see 6.4.1, for `<propertyValue>` see 6.4.2.2, for `<recordType>` see 6.4.2.1, and for `<xpathConstraint>` see 6.4.2.3). The evaluation of a constraint on an item or relationship returns a Boolean expression. If the value of the Boolean expression is true, then the item or relationship is deemed to satisfy the defined constraint.

Templates are used to identify matching items and relationships to be returned in the graph response.

The optional “suppressFromResult” attribute, if present and set to true, indicates that the items or relationships that correspond to the template carrying the attribute should be suppressed from the result. Templates with this attribute set to true are still meaningful in that it may help constrain other templates in the query. For example, in order to retrieve all items that have a “dependsOn” relationship with application “foo”, the query may set this attribute to true on the template for the “foo” item and the template for the “dependsOn” relationship but not on the template for the items on which “foo” depends. Only the latter items would appear in the response. If the “suppressFromResult” attribute is not present or set to false on a template, then all the selected instances for this template are returned in the query result.

6.2.1 itemTemplate

An item matches an `<itemTemplate>` if and only if all of the following provisions are true:

- The item satisfies all the constraints defined by the `<itemTemplate>`. (In effect, an implicit AND joins the constraints.)
- For every `<relationshipTemplate>` that points to the `<itemTemplate>` as its sourceTemplate, there is a relationship matching this `<relationshipTemplate>` that has the item as its source.

- For every `<relationshipTemplate>` that points to the `<itemTemplate>` as its targetTemplate, there is a relationship matching this `<relationshipTemplate>` that has the item as its target.

An item can match more than one `<itemTemplate>` inside a given query. When this is the case, the item appears in the response once for each matching `<itemTemplate>` (unless suppressed by the "suppressFromResult" attribute).

An item template will not return relationship instances.

6.2.2 relationshipTemplate

A relationship matches a `<relationshipTemplate>` if and only if all of the following provisions are true:

- The relationship meets all the constraints in the `<relationshipTemplate>`. (In effect, an implicit AND joins the constraints.)
- The source item of the relationship matches the `<itemTemplate>` referenced as `<sourceTemplate>` by the `<relationshipTemplate>`.
- The target item of the relationship matches the `<itemTemplate>` referenced as `<targetTemplate>` by the `<relationshipTemplate>`.
- The cardinality conditions on the `<sourceTemplate>` and `<targetTemplate>` elements are satisfied, as defined by the @minimum and @maximum attributes defined 6.2.2.1.
- The depth, or the number of edges between source and target nodes in the graph, satisfies the `<depthLimit>` condition defined in 6.2.2.2.

Items, which do not have a source or target, cannot match a `<relationshipTemplate>`.

6.2.2.1 relationshipTemplate/sourceTemplate and relationshipTemplate/targetTemplate

The `<sourceTemplate>` and `<targetTemplate>` elements each refer to an `<itemTemplate>` element using the required @ref attribute. The value of the @ref attribute shall match the value of the @id attribute of an `<itemTemplate>` element in the query.

Additionally, `<sourceTemplate>` and `<targetTemplate>` elements may have the following optional attributes:

@minimum – If n is the value of the @minimum attribute, there shall be at least n relationships matching the `<relationshipTemplate>` that share the same source or target item. For example, a query to find computers that at least five services depend upon might specify `minimum="5"` on a `<sourceTemplate>` that selects services, combined with a `<targetTemplate>` that selects computers and other constraints that select a 'dependsOn' relationship.

@maximum – If n is the value of the @maximum attribute, there may be at most n relationships matching the `<relationshipTemplate>` that share the same source or target item.

6.2.2.2 relationshipTemplate/depthLimit

The `<depthLimit>` element is used to extend the relationship template to traverse multiple edges and nodes. For example, this element may be used to find all the components of an aggregate system, or all the dependencies of a business service, even if these items are not directly related to the item in question. This extended relationship is also called a "relationship chain."

The pseudo-schema of the `<depthLimit>` element is as follows:

```
<depthLimit maxIntermediateItems="xs:positiveInteger" ?
  intermediateItemTemplate="xs:IDREF" />
```

@maxIntermediateItems – The maximum number of intermediate items in the relationship chain between source and target items. A value of 1 indicates that the `<relationshipTemplate>` can traverse one intermediate item between the source item and target item. This attribute is optional. If it is not present, then the number of intermediate items between the source and the target is unlimited.

@intermediateItemTemplate – The value of the intermediateItemTemplate corresponds to the @id attribute of an `<itemTemplate>` element that is used as a prototype for intermediate items in the relationship chain. The value of the @intermediateItemTemplate attribute is also used to represent the intermediate items in the `<nodes>` element of the query response.

6.3 Content Selection

The `<contentSelector>` element determines how instances matching the template are returned in the response. If a template does not contain a `<contentSelector>` element, all matching instances and associated records are returned in the response. The term "instance" means either an item or a relationship.

If a template contains a `<contentSelector>` element, the records and properties returned for the instances that match this template are limited to those explicitly selected. Records and properties are explicitly selected by specifying their namespace and local name in the `<selectedRecordType>` element or an XPath expression in the `<xpathSelector>` element. The use of `<selectedRecordType>` and `<xpathSelector>` are mutually exclusive per content selector.

The pseudo-schema of the `<contentSelector>` element is as follows:

```
<contentSelector>
  (<selectedRecordType namespace="xs:anyURI" localName="xs:NCName" >
    <selectedProperty namespace="xs:anyURI" localName="xs:NCName" /> *
  </selectedRecordType> * |
  <xpathSelector dialect="xs:anyURI">
    <prefixMapping prefix="xs:NCName" namespace="xs:anyURI"/> *
    <expression>xs:string</expression>
  </xpathSelector> ?)
</contentSelector>
```

6.3.1 contentSelector

The use of the `<contentSelector>` element affects the contents of the matching instances in the response as follows:

- `<contentSelector />` (empty element)

  The instances matching this template are returned with no record content in the response. This may be useful if all that is required is the instanceId of instances matching this template.

6.3.1.1 contentSelector/selectedRecordType

If `<selectedRecordType>` is used without any `<selectedProperty>` child elements, all properties (child elements) of all records of the selected type are returned in the response.

At the discretion of the query service, the response may contain a record type that is an extension (as described in 8.2.2.3) of the selected record type. For example, the following query limits the response to records with a record type with `namespace="http://example.com/models"` and `localName="Computer"`.

```
<query>
  <itemTemplate id="computers">
    <contentSelector>
      <selectedRecordType namespace="http://example.com/models"
                          localName="Computer">
      </selectedRecordType>
    </contentSelector>
  </itemTemplate>
</query>
```

A valid response to this query could contain records with a record type of `namespace="http://example.com/models"` and `localName="LinuxComputer"`, as long as the record type with `localName="LinuxComputer"` is defined as an extension of the record type with `localName="Computer"` using the mechanism described in 8.2.2.3.

6.3.1.1.1 contentSelector/selectedRecordType/selectedProperty

If `<selectedProperty>` elements are included in a `<selectedRecordType>` element, only the selected properties of the selected record types are returned in the response.

EXAMPLE: 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>`.

```
<query>
  <itemTemplate id="user">
    <contentSelector>
      <selectedRecordType namespace="http://example.com/models"
                          localName="people">
        <selectedProperty namespace="http://example.com/models/people"
                          localName="name"/>
        <selectedProperty namespace="http://example.com/models/people"
                          localName="telephone"/>
      </selectedRecordType>
    </contentSelector>
    ...
  </itemTemplate>
</query>
```

Whether or not individual properties are selected, the contents of an item or relationship in the response are always in the form of `<record>` elements, as follows, or in a `<propertySet>`element, which is described in 6.6.1:

```
<record>
  <recordTypeQName>
    <propertyQName>xs:any</propertyQName> *
  </recordTypeQName>
  <recordMetadata>
    <recordId>xs:any</recordId>
    ...
  </recordMetadata>
</record> *
```

A record type may extend multiple record types, as shown in the example on the right hand side of Figure 6 in 8.2.2.3. For each record of an item, regardless of how many record types may describe a subset of the record properties and regardless of how many `<contentSelector>/<selectedRecordType>` elements select all or part of this record, the query response shall contain at most one record or property set (see 6.6.1 for a description of a property set). The record type of the returned record or property set shall be a record type that contains all the properties to be returned. Using the same example on the right hand side of Figure 6, a query that selects the faxNumber property of FaxMachine could be satisfied by returning either a FaxMachine or MultiFunctionPrinter record or property set.

6.3.1.2 contentSelector/xpathSelector

The use of the `<xpathSelector>` element may be used to selects parts of complex models or for complex selection criteria. For example, an item template has matched an item with the following record:

```
<record>
  <ex:ComputerSystem xmlns:ex=”http://www.example.org/cs”>
    ...
    <ex:NetworkInterfaces>
      <ex:ip>1.2.3.4</ex:ip>
      <ex:ip>2.3.4.5</ex:ip>
    </ex:NetworkInterfaces>
    ...
  </ex:ComputerSystem>
  ...
</record>
```

If the `<xpathSelector>` is as follows:

```
<xpathSelector
  dialect=”http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1”>
  <prefixMapping prefix=”ex” namespace=”http://www.example.org/cs” />
  <expression>
    /ex:ComputerSystem/ex:NetworkInterfaces/ex:ip
  </expression>
</xpathSelector>
```

The record returned would be:

```
<record>
  <ex:ip>1.2.3.4</ex:ip>
  <ex:ip>2.3.4.5</ex:ip>
</record>
```

6.3.1.2.1 contentSelector/xpathSelector/@dialect

The dialect corresponds to a particular version or profile of XPath represented by the URI value. See 6.5 for more information on XPath dialects.

6.3.1.2.2 contentSelector/xpathSelector/prefixMapping

Each `<prefixMapping>` child element of the `<xpathConstraint>` 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>/@namespace` attribute. These prefix-namespace pairings shall be added to the namespace declarations of the XPath processor.

6.3.1.2.3 contentSelector/xpathSelector/expression

The `<expression>` element contains an XPath expression to be evaluated according to the chosen dialect against each `<record>` element contained in an item or relationship that has satisfied all of the constraints. The evaluation result is then transformed and normalized into a single DOM node according to the mechanism prescribed by the dialect. See 6.5 for more information on XPath normalization.

If that response DOM node has any children, then the record is selected and those children are appended to the `<record>` element.

6.4 Constraints

Constraints are used to restrict the instances returned based on properties of the instances and associated records.

6.4.1 instanceIdConstraint

The `<instanceIdConstraint>` element is used to point to specific instances by instance ID. The pseudo-schema of this element is as follows:

```
<instanceIdConstraint>
  <instanceId>cmdbf:MdrScopedIdType</instanceId> +
</instanceIdConstraint>
```

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

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 MDRs that have content about the instance, plus possibly an additional instance ID for the instance assigned by the federating CMDB itself.

The constraint is satisfied if one of the known instance IDs for the instance matches one of the requested values (that is, if both the `<mdrId>` and the `<localId>` match using string comparison).

6.4.2 recordConstraint

The `<recordConstraint>` element is used to point to specific record types and related properties to be evaluated.

The pseudo-schema of this element is as follows:

```
<recordConstraint>
  <recordType namespace="xs:anyURI"
    localName="xs:NCName"/> *
  <propertyValue> ... </propertyValue> *
  <xpathConstraint> ... </xpathConstraint> ?
  xs:any
</recordConstraint>
```

The `<recordConstraint>` element can appear any number of times inside an `<itemTemplate>` or a `<relationshipTemplate>`.

6.4.2.1 recordConstraint/recordType

The `<recordType>` element can appear any number of times inside a `<recordConstraint>` element.

One way for this constraint to be satisfied is if the instance has a record of that type. More specifically, if the instance contains a record element that has, as the first child element, an element in the namespace corresponding to the value of the `<recordType>/@namespace` attribute and where the local name of that first child element is the value of the `<recordType>/@localName` attribute. The constraint could also be satisfied by an instance with a record that is an extension of that QName, as described in 8.2.2.3. (For example, comp:Linux might be defined as an extension of comp:OperatingSystem.)

6.4.2.2 recordConstraint/propertyValue

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 `<propertyValue>` 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.

The `<propertyValue>` element is not applicable to properties that are defined as a complex type.

The pseudo-schema of this element is as follows:

```
<propertyValue namespace="xs:anyURI"
      localName="xs:NCName"
      recordMetadata="xs:boolean" ?
      matchAny="xs:boolean" ? >
  <equal caseSensitive="xs:boolean"? negate="xs:boolean"? >
    xs:anySimpleType
  </equal> *
  <less negate="xs:boolean"? >xs:anySimpleType</less> ?
  <lessOrEqual negate="xs:boolean"? >xs:anySimpleType</lessOrEqual> ?
  <greater negate="xs:boolean"? >xs:anySimpleType</greater> ?
  <greaterOrEqual negate="xs:boolean"?>
    xs:anySimpleType
  </greaterOrEqual> ?
  <contains caseSensitive="xs:boolean"? negate="xs:boolean"? >
    xs:string
  </contains> *
  <like caseSensitive="xs:boolean"? negate="xs:boolean"? >
    xs:string
  </like> *
  <isNull negate="xs:boolean"? /> ?
  xs:any
</propertyValue>
```

The `<propertyValue>` element can appear any number of times in `<recordConstraint>`. Its namespace and localName attributes define the QName of the property being tested. If there are one or more `<recordType>` elements in the enclosing `<recordConstraint>`, they define the record types against which to evaluate the constraint. If there are no `<recordType>` elements, the `<propertyValue>` element is evaluated against all record types.

The child elements of `<propertyValue>` are called operators. A `<propertyValue>` constraint is considered to be satisfied if the operators return a positive (true) result for one or more records associated with the instance (logical OR across the records).

The operators are largely defined in terms of 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 in 6.4.2.3.

@recordMetadata – The value of this attribute indicates that the property to be evaluated is in the `<recordMetadata>` element of the record.

@matchAny – The value of this attribute defines whether the operators inside that element are logically AND-ed or OR-ed. The default value for the matchAny attribute is false. If the value of the matchAny attribute is false, the constraint 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 constraint (logical AND). If the value of the matchAny attribute is true, the constraint 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 constraint (logical OR).

6.4.2.2.1 recordConstraint/propertyValue/equal

This operator is defined in terms of the XPath 2.0 value comparison operator "eq". To evaluate, the operand on the left is the property value from the record and the operand on the right is the value of the constraint from the query. The type of the value of the constraint shall 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 XPath 2.0, "Appendix B.2 – Operator Mappings".

6.4.2.2.2 recordConstraint/propertyValue/less, recordConstraint/propertyValue/lessOrEqual, recordConstraint/propertyValue/greater, and recordConstraint/propertyValue/greaterOrEqual

These operators are defined in terms of the XPath 2.0 value comparison operators "lt", "le", "gt", and "ge", respectively. To evaluate, the operand on the left is the property value from the record and the operand on the right is the value of the constraint from the query. The type of the value of the constraint shall be interpreted to be of the same type as the value from the property in the record. These operators are valid only for properties that are numerals, dates, and strings. A list of comparison behaviors is available in 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 is a string, then "2000-01-01T00:00:00" is interpreted as a string and compared with the property value using string comparison.

6.4.2.2.3 recordConstraint/propertyValue/contains

This operator is mapped to the XPath 2.0 function fn:contains(). It is valid only for properties of type string and used to test whether the property value contains the specified string as a substring. The result of the contains operator is as if the fn:contains( ) function were executed with the first parameter being the property value and the second parameter being the string specified.

6.4.2.2.4 recordConstraint/propertyValue/like

This operator is similar in functionality to the SQL LIKE clause. The operator works like the equal operator with the inclusion of the following two special characters:

- The underscore character ("_ ") acts as a wild card for any single character.
- 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 ("\").

6.4.2.2.5 recordConstraint/propertyValue/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.

6.4.2.2.6 Additional Attributes

The following additional attributes are defined for operator elements:

@caseSensitive – This is an optional attribute for the equal, contains, and like operators. The default value is true. If the property value of the record is an instance of xs:string and the caseSensitive attribute 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 – This is an optional attribute for all operators. The default value is false. When the negate attribute is true, the result of the comparison is negated.

Table 2 summarizes which operators are supported for the various XSD built-in datatypes. Unless explicitly specified, the caseSensitive attribute is not supported.

Table 2 – Operators Supported for XSD Built-in Datatypes

Built-in Datatypes	equal	isNull	less, lessOrEqual, greater, greaterOrEqual	contains	like
"String-related types" (String, anyURI, and types derived from string)	Yes, including the optional caseSensitive attribute	Yes	Yes	Yes, including the optional caseSensitive attribute	Yes, including the 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

If more than one property uses the same QName, the comparison has to hold true for only one of the property values.

EXAMPLE 1: Consider the following example for a computer with three IP addresses:

```
<comp:ComputerConfig xmlns:comp="http://example.com/computers">
  ...
  <comp:ip>1.2.3.4</comp:ip>
  <comp:ip>1.2.3.5</comp:ip>
  <comp:ip>1.2.3.6</comp:ip>
  ...
</comp:ComputerConfig>
```

The following property constraint would return a positive result:

```
<recordConstraint>
  <propertyValue namespace="http://example.com/computers"
    localName="ip">
    <equal>1.2.3.5</equal>
  </propertyValue>
</recordConstraint>
```

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.

EXAMPLE 2: For example, on the same computer with three IP addresses:

```
<recordConstraint>
  <propertyValue namespace="http://example.com/computers"
    localName="ip">
    <equal negate="true">1.2.3.5</equal>
  </propertyValue>
</recordConstraint>
```

The property constraint would remove the item above from the result set because the equality comparison matches one IP address in the list.

Similarly, `<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.

EXAMPLE 3: The following is a simple example of using `<propertyValue>`. "Manufacturer" is a property defined in the "http://example.com/Computer" namespace. The constraint is testing whether the instance has a record containing this property and where the value of the property is "HP".

```
<recordConstraint>
  <propertyValue namespace="http://example.com/Computer"
                     localName="Manufacturer" >
    <equal>HP</equal>
  </propertyValue>
</recordConstraint>
```

EXAMPLE 4: The following is a more complex example. The `<itemTemplate>` matches 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 letters), and for which the OSName property also contains either "ubuntu" or "debian" (irrespective of case).

```
<itemTemplate id="linuxMachine">
  <recordConstraint>
    <propertyValue namespace="http://example.com/computers"
                       localName="CPUCount">
      <greaterOrEqual>2</greaterOrEqual>
    </propertyValue>
    <propertyValue namespace="http://example.com/computers"
                       localName="OSName">
      <contains>Linux</contains>
    </propertyValue>
    <propertyValue namespace="http://example.com/computers"
```

```
                         localName="OSName"
                         matchAny="true">
      <contains caseSensitive="false">ubuntu</contains>
      <contains caseSensitive="false">debian</contains>
    </propertyValue>
  <recordConstraint/>
</itemTemplate>
```

6.4.2.3 recordConstraint/xpathConstraint

The `<xpathConstraint>` element provides an alternate mechanism to constrain items and relationships. The pseudo-schema of this element is as follows:

```
<xpathConstraint dialect="xs:anyURI">
  <prefixMapping prefix="xs:NCName" namespace="xs:anyURI"/> *
  <expression>xs:string</expression>
</xpathConstraint>
```

The `<xpathConstraint>` element may appear once inside a `<recordConstraint>` inside an `<itemTemplate>` or `<relationshipTemplate>` element. It can only be used in conjunction with a `<propertyValue>` constraint if the `<propertyValue>` constraint in question applies to record metadata. In other words, if a `<recordConstraint>` contains a `<xpathConstraint>` then it can only contain `<propertyValue>` elements, which have the `recordMetadata` attribute set to true. When such metadata-related `<propertyValue>` elements are used together with a `<xpathConstraint>` element, they are all ANDed together: to be selected, an item or relationship shall have a record for which the metadata meets all the constraints in the `<propertyValue>` elements and the record content satisfies the XPath constraint.

6.4.2.3.1 recordConstraint/xpathConstraint/@dialect

The dialect corresponds to a particular version or profile of XPath represented by the URI value. See 6.5 for more information on XPath dialects.

6.4.2.3.2 recordConstraint/xpathConstraint /prefixMapping

Each `<prefixMapping>` child element of the `<xpathConstraint>` 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>/@namespace` attribute. These prefix-namespace pairings shall be added to the namespace declarations of the XPath processor.

6.4.2.3.3 recordConstraint/xpathConstraint/expression

The `<expression>` element contains an XPath expression to be evaluated according to the specified dialect.

The `<xpathConstraint>` is satisfied if the evaluation result's boolean value is true. The boolean value of the evaluation result is the same result as running the XPath 1 function boolean() on the results of a XPath 1 evaluation or the XPath 2 function fn:boolean() on the results of a XPath 2 evaluation.

EXAMPLE: In the following example, "name" is a property defined in the "http://example.com/people" namespace. The constraint tests whether the instance has a record containing this property where the value of the property is "Pete the Lab Tech". In this example, no metadata is selected by the expression.

```
<itemTemplate>
  <recordConstraint>
    <xpathConstraint
```

```
      dialect=" http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1">
      <prefixMapping prefix="hr" value="http://example.com/people"/>
      <expression>/hr:ContactInfo[hr:name = "Pete the Lab Tech"]
      </expression>
    </xpathConstraint>
  </recordConstraint>
</itemTemplate>
```

6.5 XPath Expressions and Normalization

XPath may be used as a more flexible way to constrain what items/relationships are matched in a query and/or to select the record content returned for selected items/relationships. When used as a selector and a constraint, the client and server need to have a common understanding of how they will interpret and process the XPath expression. This is done through specifying an XPath dialects and a corresponding URI. This specification defines two dialects that may be used as either a selector or as a constraint:

- "http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1" indicates that the expression corresponds to an XPath 1.0 expression.
- "http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath2" indicates that the expression corresponds to an XPath 2.0 expression.

Other dialects may be defined in future versions of this specification or in other specifications.

Implementations are free to provide its own URI for a dialect that is not defined in the specification.

To enable serialization and to simplify the processing of the XPath selector, the XPath selector evaluation result is run through a transformation and then a normalization process. The transformation process transforms attribute nodes into element nodes; this allows them to be serialized later on. Next, this result is run through the normalization process which creates a single DOM node with the selection result nodes as children.

The normalization process shall throw a cmdbf:XPathSerializationFault fault if there is unsupported serialization input from the transformation process. For the XPath 1.0 normalization process, the serialization input shall either be a simple value or a nodeset made up of only element nodes. For the XPath 2.0 normalization process, the serialization input shall not contain any namespace, comment, or processing instruction nodes.

6.5.1 XPath 1.0 Dialect

This dialect indicated by the URI of http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1 is specified for XPath 1.0 support, subject to the conditions described in 6.5.3 and 6.5.4.

The XPath expression is evaluated in the following context:

Component	**Value**
Context Node	The first child of the `<record>` element
Context Position	1
Context Size	1
Variable Binding	None
Function Libraries	Core function library
Namespace Declarations	Prefixes bound via `<prefixMapping>` element

6.5.2 XPath 2.0 Dialect

This dialect indicated by the URI of http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath2 is specified for XPath 2.0 support, subject to the conditions described in 6.5.3 and 6.5.5.

The XPath expression is evaluated in the following context:

Component	Value
XPath 1.0 Compatibility Mode	False
Statically known namespaces	Prefixes bound via `<prefixMapping>` element
Default element/type namespace	None
Default function namespace	None
In-scope variables	None
Context item static type	element([namespace of this specification], record)
Function signatures	Functions defined in XQuery 1.0 and XPath 2.0 Functions and Operators
Context item	The first child of the `<record>` element
Context position	1
Context size	1
Current date and time	Time on server when request was made

6.5.3 XPath Selector Transformation

The transformation allows for selecting XML attributes. This is done through mapping an XML attribute to a `<attributeNode>` element:

- The XML attribute value is mapped to the @value of the `<attributeNode>`.
- The XML attribute local name is mapped to the @localName of the `<attributeNode>`.
- The XML attribute namespace is mapped to the @namespace of the `<attributeNode>`.

The pseudo schem of `<attributeNode>` looks like:

```
<cmdbf:attributeNode namespace="xs:anyUri"
  localName="xs:NCName" value="xs:anySimpleType" />
```

The result is as if the following XSLT template was matched to the selection result:

```
<xsl:template match="@*">
  <cmdbf:attributeNode>
    <xsl:attribute name="namespace">
      <xsl:value-of select="namespace-uri(.)" /></xsl:attribute>
    <xsl:attribute name="localName">
      <xsl:value-of select="local-name(.)" /></xsl:attribute>
    <xsl:attribute name="value">
      <xsl:value-of select="." /></xsl:attribute>
  </cmdbf:attributeNode>
</xsl:template>
```

The “xsl” prefix is bound to XSL 1.0 or 2.0 depending on whether an XPath 1 or XPath 2 evaluation result was input.

Here's an example of how an attribute would be mapped. If the record is:

```
<hr:ContactInfo xmlns:hr="http://example.com/hr" changeby="jsmith">
...
</hr:ContactInfo>  <cmdbf:attributeNode>
```

The result of the content selector with an XPath selector with the expression "hr:ContactInfo/@changeby" would be:

```
<cmdbf:attributeNode namespace=""
                     localName="changeby"
                     value="jsmith" />
```

6.5.4 XPath 1.0 Normalization

The selection evaluation result set for XPath 1.0 is then normalized:

Create a new sequence S.

If the result set is empty, then add a zero length string to the sequence S. If the result set contains a string, a number, or a boolean, run the XPath string() on the item to get the string value and add this string value to the sequence S. If the result set is a node set and contains any node other then a element node, throw a cmdbf:XPathSerializationFault; if the result is a node set and only contains nodes of type element, then add these nodes to the sequence S.

Create a new DocumentFragment named DF. For each item in S, if the item is a string, create a text node and add the text node to DF. Or, if the item is an element node, add the element node to DF.

The result of this normalization process is a DocumentFragment named DF.

6.5.5 XPath 2.0 Normalization

The selection result set for XPath 2.0 results is then normalized as defined in Section 2 "Sequence Normalization" of the *XSLT 2.0 and XQuery 1.0 Serialization* specification. If the serialization input contains any namespace, comment, or processing instruction nodes, or any other serialization error occurs, cmdbf:XPathSerializationFault shall be thrown. The serialization error definition is from http://www.w3.org/TR/xslt-xquery-serialization/#serial-err.

6.6 GraphQuery Response

The pseudo-schema for the GraphQuery response message is as follows:

```
<queryResult>
    <nodes templateId="xs:ID">
      <item>
         <record>
            xs:any
            |
            <propertySet namespace="xs:anyURI" localName="xs:NCName" >
               xs:any *
            </propertySet>
            <recordMetadata>
              <recordId>...</recordId> ?
              <lastModified>...</lastModified> ?
              <baselineId>...</baselineId> ?
              <snapshotId>...</snapshotId> ?
              xs:any
            </recordMetadata> ?
          </record> *
          <instanceId>
            <mdrId>xs:anyURI</mdrId>
            <localId>xs:anyURI</localId>
          </instanceId> +
          <additionalRecordType namespace="xs:anyURI"
                                localName="xs:NCName"/> *
      </item> +
    </nodes> *
    <edges templateId="xs:ID">
      <relationship>
        <source>
          <mdrId>xs:anyURI</mdrId>
          <localId>xs:anyURI</localId>
        </source>
        <target>
          <mdrId>xs:anyURI</mdrId>
          <localId>xs:anyURI</localId>
        </target>
         <record>
            xs:any
            <recordMetadata>
              <recordId>...</recordId> ?
              <lastModified>...</lastModified> ?
              <baselineId>...</baselineId> ?
              <snapshotId>...</snapshotId> ?
            </recordMetadata> ?
          </record> *
        <instanceId>
          <mdrId>xs:anyURI</mdrId>
          <localId>xs:anyURI</localId>
        </instanceId> +
        <additionalRecordType namespace="xs:anyURI"
                              localName="xs:NCName"/> *
      </relationship> +
    </edges> *
</queryResult>
```

Each time an item matches an `<itemTemplate>`, an `<item>` element appears inside a `<nodes>` element in the `<queryResult>` (unless the itemTemplate has the attribute "suppressFromResults" set to true). Note that for an item to "match" an `<itemTemplate>` it needs to not just meet the conditions inside the `<itemTemplate>` but also any `<relationshipTemplate>` that references the `<itemTemplate>` as described in 6.2.2. The templateId attribute of the response `<nodes>` element containing the item has the same value as the id attribute of the corresponding `<itemTemplate>` in the original request. If the item matches more than one `<itemTemplate>`, the `<item>` will be contained in the `<nodes>` for each `<itemTemplate>` matched by the item that doesn't have the "suppressFromResults" attribute set to true (each `<nodes>` element 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 matches more than one `<relationshipTemplate>`, the `<relationship>` is 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. Each record element contains one or two child elements. The first child is an element whose QName is a recordType supported by the Query Service or a `<propertySet>` element (see 6.6.1), which would contain a subset of the properties of the recordType.. The children of that child are the properties associated with the record. The optional second child is a `<recordMetadata>` element that contains information about the record itself.

Items and relationships shall 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 `<source>` 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 `<target>` 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.

6.6.1 propertySet

A query may use `<contentSelector>/<selectedRecordType>/<selectedProperty>` or `<contentSelector>/<xpathSelector>` to request a subset of the properties of a record type. If the subset omits any mandatory properties, the resulting XML element would not be valid according to its schema. In this case, the query processor shall place the requested properties inside a <propertySet> element to avoid schema violations.

The pseudo-schema of this element is as follows:

```
<propertySet namespace="xs:anyURI" localName="xs:NCName">
  xs:any *
</propertySet>
```

The attributes are:

@namespace – The namespace of the QName of the record type.

@localName – The localName of the QName of the record type.

The child elements of `<propertySet>` are each child elements of the record type whose QName is constructed from the namespace and localName attributes.

6.7 GraphQuery Faults

The faults defined in this section are generated if the condition stated in the preamble is met. Faults are targeted at a destination endpoint according to the fault-handling rules defined by the Web service binding.

The definitions of faults in this section use the following properties:

[Code]	The fault code.
[Subcode]	The fault subcode.
[Reason]	The English language reason element.
[Detail]	The detail element. If absent, no detail element is defined for the fault.

6.7.1 Unknown Template ID

This fault occurs when a `<relationshipTemplate>` includes an ID that refers to a `<sourceTemplate>`, `<targetTemplate>`, or `<intermediateItemTemplate>` that was not included in the query.

The properties are as follows:

[Code]	Sender
[Subcode]	cmdbf:UnknownTemplateIDFault
[Reason]	The graph template ID was not declared.
[Detail]	`<cmdbf:graphId> xs:ID </cdmbf:graphId>`

6.7.2 Property Type Mismatch

This fault occurs when the value in a constraint is invalid for the type of the property as defined by the schema for the property. For example, this fault occurs when the property is a date and the query includes a parameter to compare to the date that is a string that cannot be cast to a date, such as "foobar."

The properties are as follows:

[Code]	Sender
[Subcode]	cmdbf:InvalidPropertyTypeFault
[Reason]	The property value being compared is not valid.
[Detail]	`<cmdbf:propertyName namespace="xs:anyURI" localname="xs:NCName" />`

6.7.3 XPath Processing Error

This fault occurs when the XPath expression processing results in an error. See XPath 2.0 for details on the cmdbf:xpathErrorCode.

The properties are as follows:

[Code]	Sender
[Subcode]	cmdbf:XPathErrorFault
[Reason]	The XPath expression was not processed successfully.

- [Detail] `<cmdbf:expression> xs:string </cmdbf:expression>`
  `<cmdbf:xpathErrorCode> [xpath error code] </cmdbf:xpathErrorCode>`

6.7.4 Unsupported Constraint

A constraint element in the template was specified that is not supported by this MDR.

The properties are as follows:

- [Code] Receiver
- [Subcode] cmdbf:UnsupportedConstraintFault
- [Reason] The constraint specified is unsupported.
- [Detail] `<cmdbf:constraint namespace="xs:anyURI" localname="xs:NCName" />`

6.7.5 Unsupported Selector

A selector element in the template was specified that is not supported by this MDR.

The properties are as follows:

- [Code] Receiver
- [Subcode] cmdbf:UnsupportedSelectorFault
- [Reason] The selector specified is unsupported.
- [Detail] `<cmdbf:selector namespace="xs:anyURI" localname="xs:NCName" />`

6.7.6 Expensive Query Error

The query was valid, but the server determined that the query is too expensive to execute or that it would return a result set that is too large to return. The requestor is invited to retry, using a simpler and/or more constrained query. What constitutes “too expensive” or “too large” is determined by the server.

The properties are as follows:

- [Code] Receiver
- [Subcode] cmdbf:ExpensiveQueryErrorFault
- [Reason] The query in the request is too expensive for the server to process or returns a result set that is too large to return.
- [Detail] `xs:any`

6.7.7 Query Error

The query was valid, but there was an error while performing the query. When the query includes an XPath expression, this error may be used to indicate that the specific XPath dialect is not supported.

The properties are as follows:

- [Code] Receiver
- [Subcode] cmdbf:QueryErrorFault
- [Reason] An error occurred while processing the request.
- [Detail] `xs:any`

7 Registration Service

7.1 Overview

The Registration Service is used in push mode federation, as described in 5.3.2.1.

The fundamentals of push mode federation are:

- The MDR invokes the Register operation for items or relationships that it wishes to register. Each item or relationship shall 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. This 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 presumably does not maintain the registration data and no updates to that data are accepted. For previously accepted data, a return status of declined indicates that the Registration Service no longer wishes to be updated about this item. The client would typically deregister the item's ID or attempt to re-register the item, perhaps with new data.
- This specification does not stipulate what the Registration Service should or shall do with the registered data. The semantics of accepted and declined have meaning only 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 the following actions:
  - Changing existing data, such as a property value
  - 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 of 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.
  - This specification does not stipulate what the Registration Service should or shall do after a Deregister operation.

  EXAMPLE:

  The following examples show how the Registration Service might handle a deregister operation:

  - If the Registration Service 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 the Registration Service 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 the Registration Service 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.

7.2 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.

7.2.1 Register Operation

The pseudo-schema for the Register operation is as follows:

```
<registerRequest>
  <mdrId>xs:anyURI</mdrId>
  <itemList>
    <item>
      <record>
        xs:any
        <recordMetadata>...</recordMetadata> ?
      </record> *
      <instanceId>cmdbf:MdrScopedIdType</instanceId> +
      <additionalRecordType namespace="xs:anyURI"
                            localName="xs:NCName"/> *
    </item> +
  <itemList> ?
  <relationshipList>
    <relationship>
      <source>cmdbf:MdrScopedIdType</source>
      <target>cmdbf:MdrScopedIdType</target>
      <record>
        xs:any
        <recordMetadata>...</recordMetadata> ?
      </record> *
      <instanceId>cmdbf:MdrScopedIdType</instanceId> +
      <additionalRecordType namespace="xs:anyURI"
                            localName="xs:NCName"/> *
    </relationship> +
  <relationshipList> ?
</registerRequest>
```

The following subclauses describe additional constraints on the Register operation pseudo-schema.

7.2.1.1 mdrId

The `<mdrID>` element is the ID of the MDR registering its data. This ID shall be unique among all of the MDRs and federating CMDBs that are federated together.

7.2.1.2 itemList

The `<itemList>` element lists the items being registered. The list contains any number of `<item>` elements. However, if the list contains zero `<item>` elements, including the `<itemList>` element serves no purpose. An `<item>` should not be repeated in the list.

7.2.1.3 itemList/item

The `<item>` element indicates some or all of the contents of an `<item>`.

7.2.1.4 itemList/item/instanceId

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

7.2.1.5 itemList/item/record

Each `<item>` contains any number of `<record>` elements.

The `<record>` element shall contain exactly one child element of unrestricted type, followed by a `<recordMetadata>` element. The namespace and local name of the first child element together are the record type.

The `<record>` type shall 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 ComputerConfiguration and ComputerAsset records, 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.

7.2.1.6 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. In each `<item>` the same record type should not appear in both an `<additionalRecordType>` and a `<record>` element.

EXAMPLE: For queries, the MDR may support 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.

7.2.1.7 relationshipList

The `<relationshipList>` item indicates the list of relationships being registered. The list contains any number of `<relationship>` elements. However, if the list contains zero `<relationship>` elements, including the `<relationshipList>` element serves no purpose.

7.2.1.8 relationshipList/relationship

The `<relationship>` element includes some or all of the contents of a `<relationship>`.

7.2.1.9 relationshipList/relationship/instanceId

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

7.2.1.10 relationshipList/relationship/source

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

7.2.1.11 relationshipList/relationship/target

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

7.2.1.12 relationshipList/relationship/record

Each `<relationship>` contains any number of `<record>` elements. The `<record>` type shall be supported by the Registration Service.

The MDR may support queries for `<record>` types that it chooses not to 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>`.

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.

7.2.1.13 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>`.

7.2.2 Register Response

The pseudo-schema for the response to a Register operation is as follows:

```
<registerResponse>
  <RegisterInstanceResponse>
    <instanceId>cmdbf:MdrScopedIdType</instanceId>
    <accepted>
      <alternateInstanceId>
```

```
        cmdbf:MdrScopedIdType
      </alternateInstanceId> *
    </accepted> ?
    <declined>
      <reason>xs:string</reason> *
    </declined> ?
  <RegisterInstanceResponse> *
</registerResponse>
```

The following subclauses describe additional constraints on the Register response pseudo-schema.

7.2.2.1 registerInstanceResponse

The `<registerInstanceResponse>` element indicates the action taken for one item or relationship in the Register request. There can be any number of `<registerInstanceResponse>` elements. There should be exactly one `<registerInstanceResponse>` element per item or relationship in the Register request.

7.2.2.2 registerInstanceResponse/instanceId

The `<instanceId>` element is one of the elements from the Register request for an item or relationship.

7.2.2.3 registerInstanceResponse/accepted

The `<accepted>` element indicates that the item or relationship instance was accepted.

Exactly one of either the `<accepted>` or `<declined>` elements shall be present.

7.2.2.4 registerInstanceResponse/accepted/alternateInstanceId

The `<alternateInstanceId>` element indicates zero or more elements 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.

7.2.2.5 registerInstanceResponse/declined

The `<declined>` element indicates that the item or relationship instance was declined.

Exactly one of either the `<accepted>` or `<declined>` elements shall be present.

7.2.2.6 registerInstanceResponse/declined/reason

The `<reason>` element is zero or more strings that contain the reasons why the registration was declined.

7.2.3 Register Operation Faults

The faults defined in this section are generated if the condition stated in the preamble is met. Faults are targeted at a destination endpoint according to the fault-handling rules defined by the Web service binding.

The definitions of faults in this section use the following properties:

• [Code]	The fault code.
• [Subcode]	The fault subcode.

• [Reason]	The English language reason element.
• [Detail]	The detail element. If absent, no detail element is defined for the fault.

7.2.3.1 Invalid Record

The record does not correspond to the schema specifying the data model. This fault occurs when a required property does not exist, an extension property is used when the data model does not allow for extensions, and so on.

The properties are as follows:

• [Code]	Sender
• [Subcode]	cmdbf:InvalidRecordFault
• [Reason]	The record is invalid.
• [Detail]	`<cmdbf:recordId> xs:anyURI </cmdbf:recordId>`

7.2.3.2 Unsupported Record Type

A record of an unsupported record type was attempted to be registered.

The properties are as follows:

• [Code]	Sender
• [Subcode]	cmdbf:UnsupportedRecordTypeFault
• [Reason]	The record type is not supported.
• [Detail]	`<cmdbf:recordType namespace="xs:anyURI" localname="xs:NCName" />`

7.2.3.3 Invalid MDR ID

The MDR ID specified on an item is not recognized.

The properties are as follows:

• [Code]	Sender
• [Subcode]	cmdbf:InvalidMDRFault
• [Reason]	The MDR is not registered.
• [Detail]	`<cmdbf:mdrId> xs:anyURI </cmdbf:mdrId>`

7.2.3.4 Registration Error

There was a problem with registering the items or relationships.

The properties are as follows:

• [Code]	Sender
• [Subcode]	cmdbf:RegistrationErrorFault
• [Reason]	An error occurred while registering.
• [Detail]	`<cmdbf:recordId> xs:anyURI </cmdbf:recordId>`

7.3 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. Each item or relationship needs to be deregistered only once, regardless of the number of `<instanceId>` elements provided in the register request.

7.3.1 Deregister Operation

The pseudo-schema for the Deregister operation is as follows:

```
<deregisterRequest>
  <mdrId>xs:anyURI</mdrId>
  <itemIdList>
    <instanceId>cmdbf:MdrScopedIdType</instanceId> *
  <itemIdList> ?
  <relationshipIdList>
    <instanceId>cmdbf:MdrScopedIdType</instanceId> *
  <relationshipIdList> ?
</deregisterRequest>
```

The following subclauses describe additional constraints on the Deregister operation pseudo-schema.

7.3.1.1 mdrId

The `<mdrId>` is the ID of the MDR deregistering its data. This ID shall be the ID used when the data was registered using the Register request.

7.3.1.2 itemIdList

The `<itemIdList>` element lists items being deregistered. The list contains any number of `<instanceId>` elements. However, if the list contains zero `<instanceId>` elements, including the `<itemIdList>` element serves no purpose.

7.3.1.3 itemIdList/instanceId

The `<instanceId>` serves as a key for the `<item>`. The `<instanceId>` shall be either the `<instanceId>` from the Register request or an `<alternateInstanceId>` from a `<registerResponse>`. An `<instanceId>` should not be repeated in the list.

7.3.1.4 relationshipIdList

The `<relationshipIdList>` element lists the relationships being deregistered. The list contains any number of `<instanceId>` elements. However, if the list contains zero `<instanceId>` elements, including the `<relationshipIdList>` element serves no purpose.

7.3.1.5 relationshipIdList/instanceId

The `<instanceId>` serves as a key for the `<relationship>`. The `<instanceId>` shall be either the `<instanceId>` from the Register request or an `<alternateInstanceId>` from a `<registerResponse>`. An `<instanceId>` should not be repeated in the list.

7.3.2 Deregister Response

The pseudo-schema for the response to a Deregister operation is as follows:

```
<deregisterResponse>
  <deregisterInstanceResponse>
    <instanceId>cmdbf:MdrScopedIdType</instanceId>
    <accepted /> ?
    <declined>
      <reason>xs:string</reason> *
    </declined> ?
  <deregisterInstanceResponse> *
</deregisterResponse>
```

The following subclauses describe additional constraints on the Deregister response pseudo-schema.

7.3.2.1 deregisterInstanceResponse

The `<deregisterInstanceResponse>` element indicates the action taken for one item or relationship in the Deregister request. There can be any number of `<deregisterInstanceResponse>` elements. There should be exactly one `<deregisterInstanceResponse>` element per item or relationship in the Register request.

7.3.2.2 deregisterInstanceResponse/instanceId

The `<instanceId>` element provides the ID from the Deregister request for an item or relationship.

7.3.2.3 deregisterInstanceResponse/accepted

The `<accepted>` element indicates that the item or relationship instance was accepted.

Exactly one of either the `<accepted>` or `<declined>` elements shall be present.

7.3.2.4 deregisterInstanceResponse/declined

The `<declined>` element 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 the `<accepted>` or `<declined>` elements shall be present.

7.3.2.5 deregisterInstanceResponse/declined/reason

The `<reason>` element includes zero or more strings that contain the reasons that the deregistration was declined.

7.3.3 Deregister Operation Faults

The faults defined in this section are generated if the condition stated in the preamble is met. Faults are targeted at a destination endpoint according to the fault-handling rules defined by the Web service binding.

The definitions of faults in this section use the following properties:

• [Code]	The fault code.
• [Subcode]	The fault subcode.
• [Reason]	The English language reason element.
• [Detail]	The detail element. If absent, no detail element is defined for the fault.

7.3.3.1 Invalid MDR Id

The MDR ID specified on an item is not recognized.

The properties are as follows:

- [Code] Sender
- [Subcode] cmdbf:InvalidMDRFault
- [Reason] The MDR is not registered.
- [Detail] `<cmdbf:mdrId> xs:anyURI </cmdbf:mdrId>`

7.3.3.2 Deregistration Error

There was a problem with deregistering the items or relationships.

The properties are as follows:

- [Code] Sender
- [Subcode] cmdbf:DeregistrationErrorFault
- [Reason] An error occurred while deregistering.
- [Detail] `<cmdbf:recordId> xs:anyURI </cmdbf:recordId>`

8 Service Metadata

8.1 Overview

The register and query operations defined in this specification have a set of optional features that may be supported by a particular implementation. There are also a number of extensibility points in the specification that allow for the anticipated variability in implementations. One key point of variation is the data model or models supported for record types at a given MDR. Prior to sending register or query messages to an MDR, it may be necessary to inspect the capabilities and data models supported by that particular MDR.

The schema defined in this section includes two elements, `<queryServiceMetadata>` and `<registrationServiceMetadata>`, that can be used to indicate which optional features and data models (or record types) are supported by a particular implementation. It is recommended that each MDR implementation include an instance of the appropriate `<queryServiceMetadata>` and/or `<registrationServiceMetadata>` elements as part of the policies describing the implementation.

An example of how these elements can be incorporated into a WS-Policy `<policy>` element and then associated with the implementation's WSDL binding is provided in ANNEX F.

The subclauses in this section describe the service metadata schema elements `<queryServiceMetadata>` and `<registrationServiceMetadata>` and their contents.

Any MDR supporting the GraphQuery operation shall support an `<itemTemplate>` with `<instanceIdConstraint>` query at a minimum. Other query capabilities are optional. The service metadata for the MDR should indicate which optional query capabilities are supported.

8.2 Common Service Metadata Elements

Both `<queryServiceMetadata>` and `<registrationServiceMetadata>` elements have common `<serviceDescription>` and `<recordTypeList>` child elements to describe the service and list the record types supported by the service. These are described here for later reference.

8.2.1 serviceDescription

The required `<serviceDescription>` element is used to associate the service metadata with the MDR that is implementing this service. The `<mdrId>` is the only required element in the `<serviceDescription>`. The other optional elements in the `<serviceDescription>`, including an extensibility element, allow for further description of the service implementation.

The pseudo-schema of the contents of a `<serviceDescription>` element is as follows:

```
<serviceDescription>
  <mdrId>xs:anyURI</mdrId>
  <serviceId>xs:anyURI</serviceId> ?
  <description xml:lang="xs:language" xs:string</description> *
  xs:any *
</serviceDescription>
```

8.2.1.1 serviceDescription/mdrId

The required `<mdrId>` is the ID of the MDR that is providing this service.

8.2.1.2 serviceDescription/serviceId

`<serviceId>` is optional if there is only one instance of this service type (possible service types are query or registration) for each MDR ID. If there is more than one instance of a service type for an MDR ID, `<serviceId>` is mandatory so metadata can be correctly associated with the instance.

8.2.1.3 serviceDescription/description

The optional `<description>` element(s) may be used to describe the service in the languages of choice for human consumption. The xml:lang attribute is required. If there are multiple `<description>` elements, it is expected that each will have a different value for xml:lang.

8.2.2 recordTypeList

The `<recordTypeList>` is used to enumerate the elements that are considered valid for use as records in the implementation of this service. This list of supported record types may change over time and should be kept current by the implementation.

The pseudo-schema of the contents of a `<recordTypeList>` element is as follows:

```
<recordTypeList>
  <recordTypes namespace="xs:anyURI" schemaLocation="xs:anyURI" ? >
    <recordType localName="xs:NCName" appliesTo="xs:string">
      <superType namespace="..." localName="..."/> *
      xs:any *
    </recordType> *
  </recordTypes> *
</recordTypeList>
```

8.2.2.1 recordTypeList/recordTypes

For each different namespace that contains record types supported by the implementation, a `<recordTypes>` element should be included in the metadata that includes the namespace, schemaLocation if appropriate, and the list of the element names from that namespace which are supported by the implementation as `<recordType>` elements.

@namespace – This mandatory attribute gives the namespace of the data model that includes XML elements that correspond to record types supported by the implementation.

@schemaLocation – This optional attribute should be included when there is a URI that can be resolved to an XML schema representation of the elements belonging to the namespace listed in the namespace attribute.

8.2.2.2 recordTypeList/recordTypes/recordType

A `<recordType>` element identifies an element that is supported as a record type in the implementation. Each `<recordType>` element shall be from the namespace identified in the containing `<recordTypes>` element.

@localName – The value of this attribute corresponds to the localName of a supported XML element that is a valid record type for the implementation.

@appliesTo – This attribute shall be one of three values indicating whether this element is valid as a record in a relationship, item, or both. The values for this attribute are from the enumeration, "relationship", "item", or "both".

8.2.2.3 recordTypeList/recordTypes/recordType/superType

Record types are often extensions of other record types. A record type is an extension of another record type if it has all the properties of the other record type or is the source or target of a relationship that does not apply to the other record type. Figure 6 shows two examples of extensions.

In the left example LinuxComputerSystem is an extension of ComputerSystem, which in turn is an extension of ManagedElement. LinuxComputerSystem has all the properties of ComputerSystem plus adds some other properties specific to Linux. Alternatively or in addition, LinuxComputerSystem could be the source or target of a relationship that does not apply to all ComputerSystem instances.

In the right example MultiFunctionPrinter is an extension of both FaxMachine and Printer because it has all the properties of FaxMachine and Printer. FaxMachine and Printer are both extensions of IODevice because they both have the one property in IODevice.

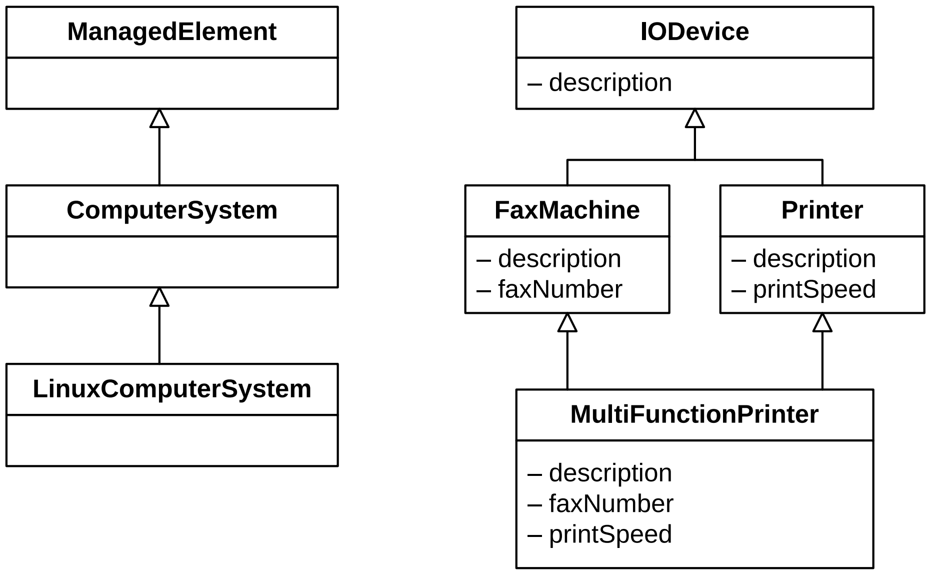


Figure 6 – Record Type Extension Examples

The response to a query for a record type X may contain instances of X or instances of any subtype of X, i.e., any type that declares X to be a super type. A record type is considered a subtype of another record type if all the following are true:

- its definition contains all the properties of the super type, and each of these is identically named and typed,
- it is valid as the source or target of any relationship that is valid for the super type,
- the characterization of the super type applies to the subtype.

A subtype may contain other properties. A record type may have multiple super types.

The `<superType>` element may be used to indicate an extension relationship between record types.

The attributes are:

@namespace – The namespace of the QName of the super type.

@localName – The localName of the QName of the super type.

8.3 queryServiceMetadata

An instance of the `<queryServiceMetadata>` includes the description of the MDR, including the ID of the MDR implementing the Query Service, the supported query capabilities and the supported records, or data model, for the given implementation being modeled.

The pseudo-schema of the contents of a `<queryServiceMetadata>` element is as follows:

```
<queryServiceMetadata>
  <serviceDescription> ... </serviceDescription>
  <supportedOptionSet>xs:anyURI</supportedOptionSet> *
  <queryCapabilities>
    <relationshipTemplateSupport depthLimit="xs:boolean"
                                 minimumMaximum="xs:boolean"
                                 xs:anyAttribute /> ?
    <contentSelectorSupport recordTypeSelector="xs:boolean"
                            propertySelector="xs:boolean"
                            xs:anyAttribute /> ?
    <recordConstraintSupport ...> ... </recordConstraintSupport> ?
    <xpathSupport>
      <dialect>xs:anyURI</dialect>*
    </xpathSupport> ?
    xs:any *
  </queryCapabilities> ?
  <recordTypeList> ... </recordTypeList>
  xs:any *
</queryServiceMetadata>
```

8.3.1 queryServiceMetadata/serviceDescription

The required `<serviceDescription>` element is used to identify this implementation of the Query Service, as previously described.

8.3.2 queryServiceMetadata/supportedOptionSet

An option set is a predefined set of query capabilities supported by the service. Each option set is identified by a URI. Listing an option set URI in a `<supportedOptionSet>` element means that the service supports all the capabilities that are part of this option set. It doesn't imply that the service does not support additional capabilities, just that those that are part of the option set are guaranteed to be supported.

If the `<queryServiceMetadata>` element also contains a `<queryCapabilities>` section, the content of the `<queryCapabilities>` should list a superset of all the capabilities in all the advertised option sets. However, the mere presence of a `<supportedOptionSet>` element is sufficient to advertise the corresponding capabilities, even if a follow-on `<queryCapabilities>` element fails to list them.

In other words, the set of capabilities advertised by the query service is the union of all the capabilities that are part of all the listed option sets (using `<supportedOptionSet>`) and all the capabilities listed in the `<queryCapabilities>` section.

This specification only defines two option sets, described below.

8.3.2.1 Complete Option Set

The URI for this option set is http://schemas.dmtf.org/cmdbf/1/optionSet/query-complete.

The complete option set indicate that all query features described in this specification are supported. It is equivalent to the following `<queryCapabilities>` element:

```
<queryCapabilities>
  <relationshipTemplateSupport depthLimit="true"
     minimumMaximum="true" />
  <contentSelectorSupport recordTypeSelector="true"
     propertySelector="true" />
  <recordConstraintSupport recordTypeConstraint="true"
               propertyValueConstraint="true">
    <propertyValuesOperators equal="true" less="true"
```

```
              lessOrEqual="true" greater="true"
              greaterOrEqual="true" contains="true"
              like="true" isNull="true" />
  </recordConstraintSupport>
  <xpathSupport>
    <dialect>http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1</dialect>
    <dialect>http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath2</dialect>
  </xpathSupport>
</queryCapabilities>
```

8.3.2.2 Base Option Set

The URI for this option set is http://schemas.dmtf.org/cmdbf/1/optionSet/query-basic.

The base option set indicates that all features listed in this specification are supported with the following exceptions:

- The @depthLimit attribute is not supported on relationship templates (relationships need to be traversed hop by hop).
- The @minimum and @maximum attributes on relationship template are not supported.
- Xpath constraints on item templates and relationship templates are not supported.

This option set is equivalent to the following `<queryCapabilities>` element:

```
<queryCapabilities>
  <relationshipTemplateSupport depthLimit="false"
     minimumMaximum="false" />
  <contentSelectorSupport recordTypeSelector="true"
     propertySelector="true" />
  <recordConstraintSupport recordTypeConstraint="true"
              propertyValueConstraint="true">
    <propertyValuesOperators equal="true" less="true"
              lessOrEqual="true" greater="true"
              greaterOrEqual="true" contains="true"
              like="true" isNull="true" />
  </recordConstraintSupport>
  <xpathSupport/>
</queryCapabilities>
```

8.3.3 queryServiceMetadata/queryCapabilities

The `<queryCapabilities>` element indicates which query techniques described in this specification are supported by this particular implementation of the query operation. The `<queryCapabilities>` element includes an extensibility element for representing that query extensions beyond the scope of this specification are supported by the implementation.

8.3.4 queryServiceMetadata/queryCapabilities/relationshipTemplateSupport

When present, the `<relationshipTemplateSupport>` element indicates that the query operation of the implementation supports queries that include `<relationshipTemplate>` elements.

@depthLimit – The Boolean value of this attribute indicates whether the Query Service implementation will process queries with a `<depthLimit>` element in a `<relationshipTemplate>`.

@minimumMaximum – The Boolean value of this attribute indicates whether the Query Service implementation will process queries based on the cardinality of relationships as specified by a @minimum or @maximum attribute on a `<sourceTemplate>` or `<targetTemplate>` element of a `<relationshipTemplate>`.

8.3.5 queryServiceMetadata/queryCapabilities/contentSelectorSupport

When present, the `<contentSelectorSupport>` element indicates that the query operation of the implementation supports queries that include `<contentSelector>` elements.

@recordTypeSelector – The Boolean value of this attribute indicates whether the Query Service implementation will process queries with `<selectedRecordType>` specified in the `<contentSelector>` of an `<itemTemplate>` or `<relationshipTemplate>`.

@propertyTypeSelector – The Boolean value of this attribute indicates whether the Query Service implementation will process queries with `<selectedProperty>` specified in the `<contentSelector>` of an `<itemTemplate>` or `<relationshipTemplate>`.

8.3.6 queryServiceMetadata/queryCapabilities/recordConstraintSupport

The `<recordConstraintSupport>` element indicates whether the query implementation will process queries that use constraints in the `<itemTemplate>` or `<relationshipTemplate>`. The complete pseudo-schema of this element is as follows:

```
<recordConstraintSupport recordTypeConstraint="xs:boolean"
   propertyValueConstraint="xs:boolean" xs:anyAttribute >
   <propertyValuesOperators equal="xs:boolean" less="xs:boolean"
      lessOrEqual="xs:boolean" greater="xs:boolean"
      greaterOrEqual="xs:boolean" contains="xs:boolean"
      like="xs:boolean" isNull="xs:boolean" xs:anyAttribute />?
</recordConstraintSupport>
```

@recordTypeConstraint – The Boolean value of this attribute indicates whether the Query Service implementation will process queries with `<recordType>` constraints in an `<itemTemplate>` or `<relationshipTemplate>`.

@propertyValueConstraint – The Boolean value of this attribute indicates whether the Query Service implementation will process queries with `<propertyValue>` constraints in an `<itemTemplate>` or `<relationshipTemplate>`. When `<propertyValue>` constraints are supported the metadata should also indicate which operators are supported by including the `<propertyValueOperators>` element.

8.3.7 recordConstraintSupport/propertyValueOperators

The `<propertyValueOperators>` element is used to indicate which operators are supported by the query implementation. There is a mandatory attribute for each operator defined by this specification and an extensibility attribute for other operators not defined by this specification.

The Boolean value of each of the following attributes indicates whether the Query Service implementation will process queries with a property value operator of the same name as the attribute: @equal, @less, @lessOrEqual, @greater, @greaterOrEqual, @contains, @like, and @isNull.

8.3.8 queryServiceMetadata/queryCapabilities/xpathSupport

The `<xpathSupport>` element is used to indicate that the query implementation supports the dialects of XPath represented by the contained `<dialect>` elements.

8.3.9 queryServiceMetadata/queryCapabilities/xpathSupport/dialect

The `<dialect>` elements indicate which dialects of XPath will be processed by the query implementation. The URI used as the value of the dialect should be either of the following:

- one of the URIs listed in this specification for XPath dialects
- a URI defined by another specification to represent an XPath dialect appropriate for use in the query operation defined in this specification

8.3.10 queryServiceMetadata/recordTypeList

The `<recordTypeList>` is used to list the record types that can be returned by the Query Service, as previously described.

8.4 registrationServiceMetadata

An instance of the `<registrationServiceMetadata>` includes the description of the MDR implementing the Registration Service, including the ID of the MDR, and the supported records, or data model, for the given implementation being modeled.

The pseudo-schema for the contents of a `<registrationServiceMetadata>` element is as follows:

```
<registrationServiceMetadata>
  <serviceDescription> ... </serviceDescription>
  <recordTypeList> ... </recordTypeList>
  xs:any *
</registrationServiceMetadata>
```

8.4.1 registrationServiceMetadata/serviceDescription

The required `<serviceDescription>` element is used to identify this implementation of the Registration Service, as previously described.

8.4.2 registrationServiceMetadata/recordTypeList

The `<recordTypeList>` is used to list the record types that can be accepted by the Registration Service, as previously described.

ANNEX A
(normative)

URIs and XML Namespaces

This annex lists the XML namespaces and other URIs defined in this specification.

URI	Description
http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1	Represents an XPath 1 dialect that can be used in queries (see 6.5.1).
http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath2	Represents an XPath 2 dialect that can be used in queries (see 6.5.2).
http://schemas.dmtf.org/cmdbf/1/optionSet/query-complete	Represents the set of query service options that contains all possible capabilities (see 8.3.2.1).
http://schemas.dmtf.org/cmdbf/1/optionSet/query-basic	Represents a set of query service options that provide basic functionality for a variety of query expressions (see 8.3.2.2).
http://schemas.dmtf.org/cmdbf/1/action/fault	Represents an action in the SOAP binding for faults.
http://schemas.dmtf.org/cmdbf/1/tns/serviceData	Represents the target namespace of the XML schema used by the CMDBf Query and Registration services.
http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata	Represents the target namespace of the CMDBf Service Description XML schema.
http://schemas.dmtf.org/cmdbf/1/tns/query	Represents the target namespace in the WSDL for the query service.
http://schemas.dmtf.org/cmdbf/1/tns/registration	Represents the target namespace in the WSDL for the registration service.

ANNEX B
(normative)

CMDB Federation XSD and WSDL

Normative copies of the XML schemas for this version of this specification may be retrieved by resolving the URLs below.

```
http://schemas.dmtf.org/cmdbf/1/tns/serviceData/dsp8040_1.0.0.xsd
http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata/dsp8041_1.0.0.xsd
```

Normative copies of the XML schemas for the current version of this specification (which is this version unless it is superseded) may be retrieved by resolving the URLs below.

```
http://schemas.dmtf.org/cmdbf/1/tns/serviceData/dsp8040.xsd
http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata/dsp8041.xsd
```

Any `xs:documentation` content in XML schemas for this specification is informative and provided only for convenience.

Normative copies of the WSDL for the query and registration services described in this version of this specification may be retrieved by resolving the URLs below.

```
http://schemas.dmtf.org/cmdbf/1/tns/query/dsp8043_1.0.0.wsdl
http://schemas.dmtf.org/cmdbf/1/tns/registration/dsp8042_1.0.0.wsdl
```

Normative copies of the WSDL for the query and registration services described in the current version of this specification (which is this version unless it is superseded) may be retrieved by resolving the URLs below.

```
http://schemas.dmtf.org/cmdbf/1/tns/query/dsp8043.wsdl
http://schemas.dmtf.org/cmdbf/1/tns/registration/dsp8042.wsdl
```

ANNEX C (normative)

Fault Binding to SOAP

Faults may be generated for any CMDBf operation. The bindings of faults for both SOAP 1.1 and SOAP 1.2 are described in this annex.

The definitions of faults use the following properties:

• [Code]	The fault code.
• [Subcode]	The fault subcode.
• [Reason]	A language-localized readable description of the error.
• [Detail]	Optional detail elements. If more than one detail element is defined for a fault, implementations shall include the elements in the order that they are specified.

Services that generate CMDBf faults shall set the [Code] property to either "Sender" or "Receiver". These properties are serialized into text XML as shown in Table C-1.

Table C-1 – [Code] Properties

SOAP Version	Sender	Receiver
SOAP 1.1	S11:Client	S11:Server
SOAP 1.2	S:Sender	S:Receiver

The properties in Table C-1 bind to a SOAP 1.2 fault as follows:

```
<S:Envelope>
 <S:Header>
   <wsa:Action>
     http://schemas.dmtf.org/cmdbf/1/action/fault
   </wsa:Action>
   <!-- Headers elided for brevity. -->
 </S:Header>
 <S:Body>
  <S:Fault>
   <S:Code>
     <S:Value> [Code] </S:Value>
     <S:Subcode>
      <S:Value> [Subcode] </S:Value>
     </S:Subcode>
   </S:Code>
   <S:Reason>
     <S:Text xml:lang="en"> [Reason] </S:Text>
   </S:Reason>
   <S:Detail>
     [Detail]
     ...
   </S:Detail>
  </S:Fault>
 </S:Body>
</S:Envelope>
```

The properties in Table C-1 bind to a SOAP 1.1 fault as follows when the fault is generated as a result of processing a CMDBf request message:

```
<S11:Envelope>
 <S11:Header>
   <cmdbf:fault>
     <cmdbf:faultCode> [Subcode] </cmdbf:faultCode>
     <cmdbf:detail> [Detail] </cmdbf:detail>
     ...
   </cmdbf:fault>
   <!-- Headers elided for brevity. -->
 </S11:Header>
 <S11:Body>
  <S11:Fault>
   <S11:faultcode> [Code] </S11:faultcode>
   <S11:faultstring> [Reason] </S11:faultstring>
  </S11:Fault>
 </S11:Body>
</S11:Envelope>
```

When binding to a CMDBf operation that supports WS-Addressing, the fault message shall include the following action URI as the [action] property:

```
    http://schemas.dmtf.org/cmdbf/1/action/fault
```

Fault handling rules for operations using WS-Addressing are defined in section 6 of *WS-Addressing SOAP Binding*.

ANNEX D (informative)

Query Examples

This annex contains two extended GraphQuery examples.

D.1 GraphQuery Example 1

Let us assume that an MDR contains two types of items (people and computers) and one type of relationship (a person "uses" a computer). The following simple query request selects all computers that are used by a person located in California:

```
<query>
  <itemTemplate id="user">
    <recordConstraint>
      <recordType namespace="http://example.com/people"
                  localName="person"/>
      <propertyValue namespace="http://example.com/people"
                     localName="state">
        <equal>CA</equal>
      </propertyValue>
    </recordConstraint>
  </itemTemplate>

  <itemTemplate id="computer">
    <recordConstraint>
      <recordType namespace="http://example.com/computer"
                  localName="computer"/>
    </recordConstraint>
  </itemTemplate>

  <relationshipTemplate id="usage">
    <recordConstraint>
      <recordType namespace="http://example.com/computer"
                  localName="uses"/>
    </recordConstraint>
    <sourceTemplate ref="user"/>
    <targetTemplate ref="computer"/>
  </relationshipTemplate>

</query>
```

The detailed syntax and semantics of the XML elements were described in the body of this specification, but the following summary describes the items and relationships that are returned by this query:

The `<itemTemplate>` called "user" (line 02) matches 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")

- are the source of a relationship that matches the `<relationshipTemplate>` called "usage" (line 11)

The `<itemTemplate>` called "computer" (line 08) matches all items that:

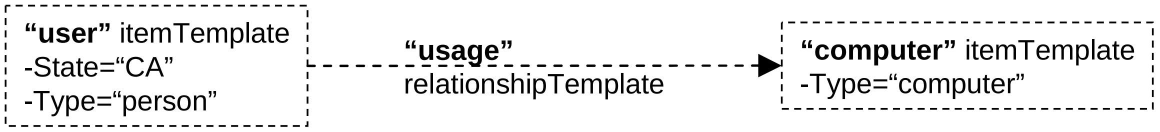
- have a record named "computer" (defined in the namespace "http://example.com/computer")
- are the target of a relationship that matches the `<relationshipTemplate>` called "usage" (line 11)

The `<relationshipTemplate>` called "usage" (line 11) matches all relationships that:

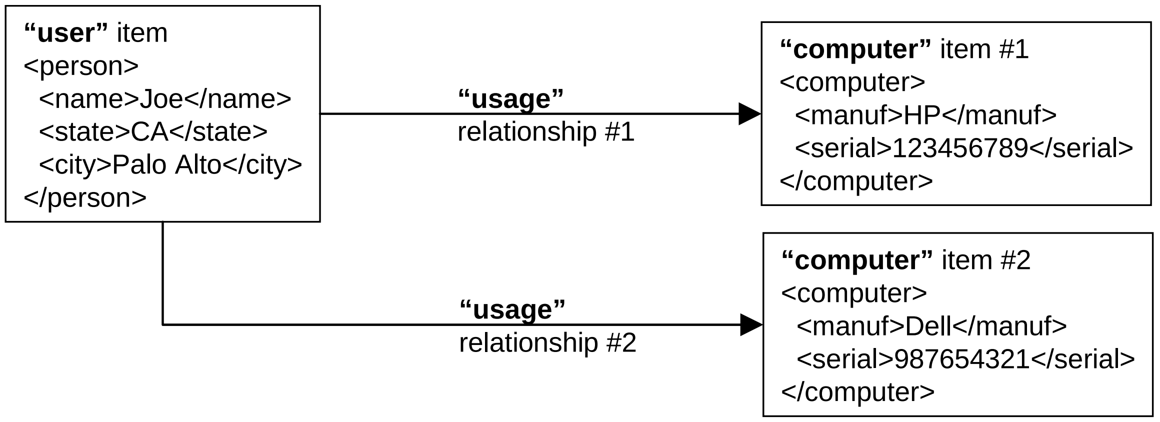
- have a record named "uses" (defined in the namespace "http://example.com/computer")
- have a source that matches the `<itemTemplate>` called "user" (line 02)
- have a target that matches the `<itemTemplate>` called "computer" (line 08)

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

The following is a graphical representation of the query:



A user in California who happens to "use" two computers is represented in the response by three items (one for the user and one for each computer) and two relationships (from the user to each of his or her computers). The following 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 (for example, if the second computer were instead "used" by another user also located in California).

If the `<relationshipTemplate>` element (line 11) were 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 of these relationships would have no bearing on what items are returned.

The GraphQuery operation can also use relationships to qualify instances, even when the result of the query does not include relationships. In the previous example, suppose that we are interested only in the computers used by people in California, not the users themselves. We can add suppressFromResult=true to the "user" and "usage" templates in the previous query. The query result is simply the two computers listed above.

```
<query>
  <itemTemplate id="user" suppressFromResult="true">
    <recordConstraint>
      <recordType namespace="http://example.com/people"
                  localName="person"/>
      <propertyValue namespace="http://example.com/people"
                     localName="state">
        <equal>CA</equal>
      </propertyValue>
    </recordConstraint>
  </itemTemplate>
  <itemTemplate id="computer">
    <recordConstraint>
      <recordType namespace="http://example.com/computer"
                  localName="computer"/>
    </recordConstraint>
  </itemTemplate>
  <relationshipTemplate id="usage" suppressFromResult="true">
    <recordConstraint>
      <recordType namespace="http://example.com/computer"
                  localName="uses"/>
    </recordConstraint>
    <sourceTemplate ref="user"/>
    <targetTemplate ref="computer"/>
  </relationshipTemplate>
</query>
```

D.2 GraphQuery Example 2

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 that are administered by "Pete the Lab Tech" and returns all items and relationships involved in this graph. The response shows two computers administrated by one user.

The data the query is executed against are as follows:

Table D-1 – "User (ContactInfo)" Data

Name	Phone	employeeNumber
Pete the Lab Tech	111-111-1111	109
Joe the Manager	111-111-4567	12
Frank the CEO	111-111-9999	1

Table D-2 – "Computer (ComputerConfig)" 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

Table D-3 – "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

The following example involves a relationship traversal:

```
<query>
  <itemTemplate id="user">
    <recordConstraint>
      <recordType namespace=http://example.com/people
          localName="ContactInfo"/>
      <propertyValue namespace=http://example.com/people
          localName="name">
        <equal>Pete the Lab Tech</equal>
      </propertyValue>
    </recordConstraint>
  </itemTemplate>
  <itemTemplate id="computer">
    <recordConstraint>
      <recordType
          namespace=http://example.com/computerModel
          localName="ComputerConfig"/>
    </recordConstraint>
  </itemTemplate>
  <relationshipTemplate id="administers">
    <recordConstraint>
      <recordType
          namespace=http://example.com/computerModel
          localName="administers"/>
    </recordConstraint>
    <sourceTemplate ref="user"/>
    <targetTemplate ref="computer"/>
  </relationshipTemplate>
</query>
```

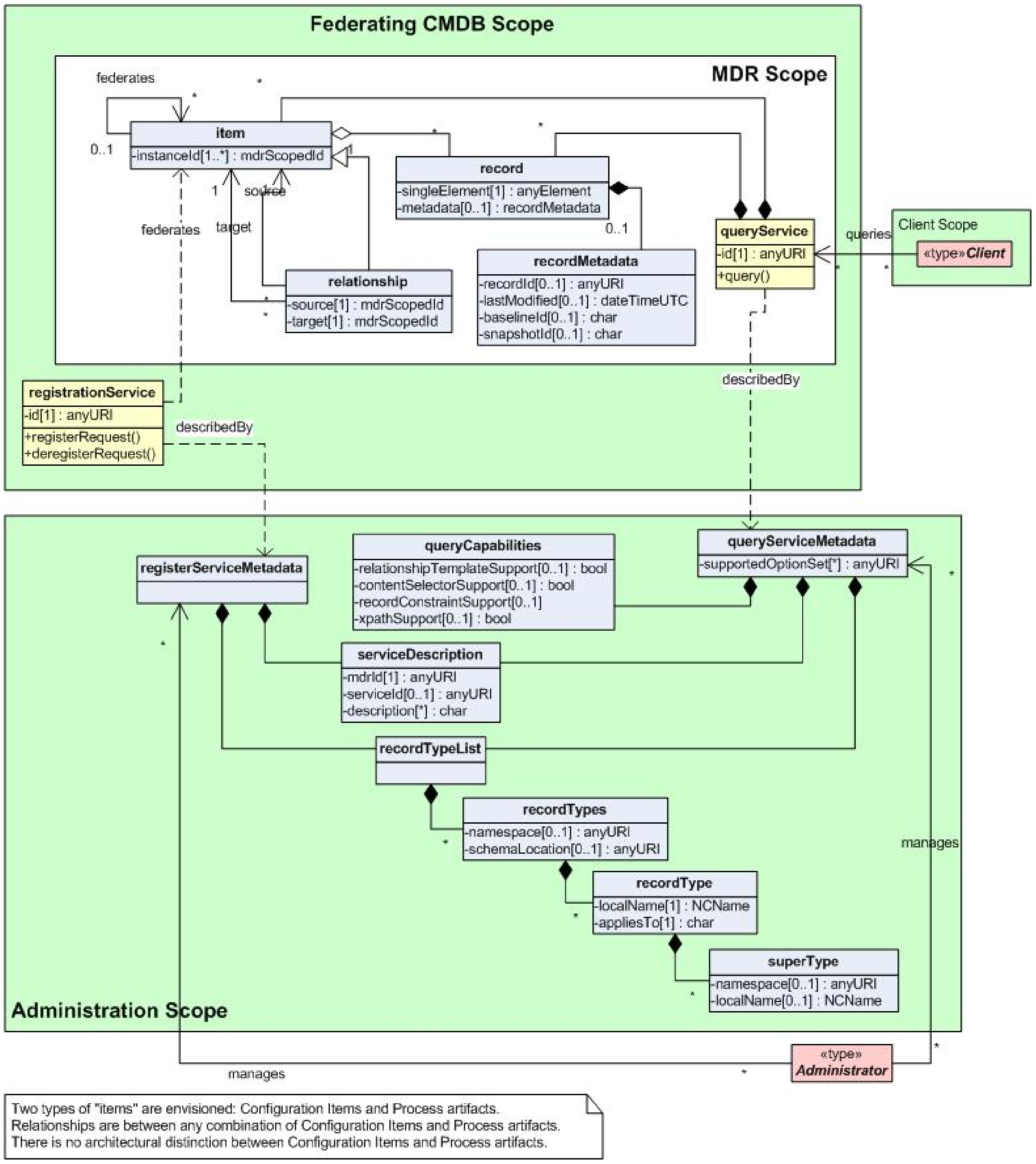
The following is a response to the GraphQuery:

```
<queryResult>
    <nodes templateId="user">
      <item>
        <record xmlns:hr="http://example.com/people">
          <hr:ContactInfo>
            <hr:name>Pete the Lab Tech</hr:name>
            <hr:phone>111-111-1111</hr:phone>
            <hr:employeeNumber>109</hr:employeeNumber>
          </hr:ContactInfo>
          <recordMetadata>
            <recordId>http://example.com/109/Current</recordId>
          </recordMetadata>
        </record>
        <instanceId>
          <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
          <localId>http://example.com/PeteTheLabTech</localId>
        </instanceId>
      </item>
    </nodes>
    <nodes templateId="computer">
      <item>
        <record xmlns:comp="http://example.com/computerModel">
          <comp:ComputerConfig>
            <comp:CPUType>AMD Athlon 64</comp:CPUType>
            <comp:assetTag>XYZ9753</comp:assetTag>
            <comp:primaryMACAddress>
              00A4B49D2F41
            </comp:primaryMACAddress>
            <comp:name>LabMachineA</comp:name>
            ...
          </comp:ComputerConfig>
          <recordMetadata>
            <recordId>
              http://example.com/machines/XYZ9753/scanned
            </recordId>
          </recordMetadata>
        </record>
        <instanceId>
          <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
          <localId>http://example.com/machines/XYZ9753</localId>
        </instanceId>
      </item>
      <item>
        <record xmlns:comp="http://example.com/computerModel">
          <comp:ComputerConfig>
            <comp:CPUType>AMD Athlon 64</comp:CPUType>
            <comp:assetTag>XYZ9876</comp:assetTag>
            <comp:primaryMACAddress>
```

```
                00A4B49D2F42
              </comp:primaryMACAddress>
              <comp:name>LabMachineB</comp:name>
              ...
            </comp:ComputerConfig>
            <recordMetadata>
              <recordId>
                http://example.com/machines/XYZ9876/scanned
              </recordId>
            </recordMetadata>
          </record>
          <instanceId>
            <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
            <localId>http://example.com/machines/XYZ9876</localId>
          </instanceId>
        </item>
      </nodes>
      <edges templateId="administers">
        <relationship>
          <source>
            <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
            <localId>http://example.com/PeteTheLabTech</localId>
          </source>
          <target>
            <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
            <localId>http://example.com/machines/XYZ9876</localId>
          </target>
          <record xmlns:foo="http://example.com/computerModel">
            <foo:administers>
              <foo:adminSupportHours>
                business hours only
              </foo:adminSupportHours>
            </foo:administers>
            <recordMetadata>
              <recordId>adm10001</recordId>
            </recordMetadata>
          </record>
          <instanceId>
            <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
            <localId>
          http://example.com/administers/PeteTheLabTechToLabMachineB
            </localId>
          </instanceId>
        </relationship>
        <relationship>
          <source>
            <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
            <localId>http://example.com/PeteTheLabTech</localId>
          </source>
          <target>
            <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
            <localId>http://example.com/machines/XYZ9753</localId>
          </target>
          <record xmlns:foo="http://example.com/computerModel">
            <foo:administers>
              <foo:adminSupportHours>24/7</foo:adminSupportHours>
            </foo:administers>
            <recordMetadata>
              <recordId>adm10002</recordId>
            </recordMetadata>
          </record>
          <instanceId>
            <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
            <localId>
          http://example.com/administers/PeteTheLabTechToLabMachineA
            </localId>
          </instanceId>
        </relationship>
      </edges>
  </queryResult>
```

ANNEX E
(informative)

Detailed UML Class Diagrams



Two types of "items" are envisioned: Configuration Items and Process artifacts.
Relationships are between any combination of Configuration Items and Process artifacts.
There is no architectural distinction between Configuration Items and Process artifacts.

Figure E-1 – UML Class Diagrams

ANNEX F (informative)

Sample WSDL Binding

The following example illustrates how the interfaces defined in this specification should be described in a Web service binding that implements the interfaces. This example also illustrates how the CMDBf service metadata should be associated with a particular implementation of a CMDBf interface.

As shown below, this query implementation uses SOAP 1.1 over HTTP as the protocol and supports the use of WS-Addressing if the message sender uses WS-Addressing for an asynchronous request/response. Because this specification does not define specific WS-Addressing actions, the action header values for WS-Addressing are determined according to the defaults described in the *WS-Addressing 1.0 – WSDL Binding* specification.

The queryServiceMetadata element is included in a WS-Policy expression which is included by reference in the WSDL binding to the query port type. This particular sample is of a Query Service that supports the complete set of record constraint and selector operators defined in the specification. The metadata in the sample also shows that XPath1 and XPath 2 are supported by the service.

The metadata for the service also includes the two record types that may be queried at this service, an "R_ComputerSystem" data type, and a "CIM_CommonDatabase" data type.

The approach to including metadata as a policy in the WSDL is a recommended approach to creating the WSDL documentation for the binding implementation as it allows for the file containing the WSDL binding to completely describe the interface to the service and the options allowed by this specification.

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

<!--

Copyright © 2009 Distributed Management Task Force, Inc. (DMTF). All rights reserved.
DMTF is a not-for-profit association of industry members dedicated to promoting
enterprise and systems management and interoperability. Members and non-members may
reproduce DMTF specifications and documents provided that correct attribution is
given. As DMTF specifications may be revised from time to time, the particular version
and release date should always be noted. Implementation of certain elements of this
standard or proposed standard may be subject to third party patent rights, including
provisional patent rights (herein "patent rights"). DMTF makes no representations to
users of the standard as to the existence of such rights, and is not responsible to
recognize, disclose, or identify any or all such third party patent right, owners or
claimants, nor for any incomplete or inaccurate identification or disclosure of such
rights, owners or claimants. DMTF shall have no liability to any party, in any manner
or circumstance, under any legal theory whatsoever, for failure to recognize,
disclose, or identify any such third party patent rights, or for such partyâ€™s
reliance on the standard or incorporation thereof in its product, protocols or testing
procedures. DMTF shall have no liability to any party implementing such standard,
whether such implementation is foreseeable or not, nor to any patent owner or
claimant, and shall have no liability or responsibility for costs or losses incurred
if a standard is withdrawn or modified after publication, and shall be indemnified and
held harmless by any party implementing the standard from any and all claims of
infringement by a patent owner for such implementations. For information about patents
held by third-parties which have notified the DMTF that, in their opinion, such patent
may relate to or impact implementations of DMTF standards, visit
http://www.dmtf.org/about/policies/disclosures.php.

-->


<wsdl:definitions
```

```
targetNamespace="http://schemas.dmtf.org/cmdbf/1/tns/query"
xmlns:cmdbfPort="http://schemas.dmtf.org/cmdbf/1/tns/query"
xmlns:cmdbfMetadata="http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

<wsdl:import location="query.wsdl"
  namespace="http://schemas.dmtf.org/cmdbf/1/tns/query">
</wsdl:import>

<!-- Subject supports WS-Addressing -->
<wsp:Policy xml:Id="SupportsWSAddressing">
  <wsam:Addressing wsp:Optional="true">
    <wsp:Policy />
  </wsam:Addressing>
</wsp:Policy>


<!-- Subject supports the referenced data model in the operations -->
<wsp:Policy xml:Id="SupportedMetadata">
  <queryServiceMetadata
            xmlns="http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata">
    <serviceDescription>
      <mdrId>CMDBf12345</mdrId>
    </serviceDescription>
    <queryCapabilities>
      <contentSelectorSupport propertySelector="true"
                              recordTypeSelector="true" />
      <recordConstraintSupport recordTypeConstraint="true"
                               propertyValueConstraint="true">
        <propertyValueOperators equal="true" less="true"
                                greater="true" lessOrEqual="true"
                                greaterOrEqual="true"
                                contains="true"
                                like="false"
                                isNull="false" />
      </recordConstraintSupport>
      <xpathSupport>
        <dialect>
          http://www.w3.org/TR/1999/REC-xpath-19991116
        </dialect>
        <dialect>
          http://www.w3.org/TR/2007/REC-xpath-20070123
        </dialect>
      </xpathSupport>
    </queryCapabilities>
```

```

      <recordTypeList>
          <recordTypes namespace="http://cmdbf.org"
                  schemaLocation="http://cmdbf.org/common_schemas/R_ComputerSystem.xsd">
             <recordType localName="R_ComputerSystem" />
         </recordTypes>
             <recordTypes
   namespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_CommonDatabase"
   schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_CommonDatabase.xsd">
                <recordType localName="CIM_CommonDatabase" />
             </recordTypes>
      </recordTypeList>

    </queryServiceMetadata>
  </wsp:Policy>

  <!-- Sample Binding for SOAP 1.1 with WS-Addressing support
  -->
  <wsdl:binding name="QueryBinding" type="cmdbfPort:QueryPortType">
    <soap:binding style="document"
       transport="http://schemas.xmlsoap.org/soap/http" />
    <wsp:PolicyReference URI="SupportsWSAddressing" />
    <wsp:PolicyReference URI="SupportedMetadata" />
    <wsdl:operation name="GraphQuery">
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
      <wsdl:fault name="UnknownTemplateID">
        <soap:fault name="UnknownTemplateID" use="literal" />
      </wsdl:fault>
      <wsdl:fault name="InvalidPropertyType">
        <soap:fault name="InvalidPropertyType" use="literal" />
      </wsdl:fault>
      <wsdl:fault name="XPathError">
        <soap:fault name="XPathError" use="literal" />
      </wsdl:fault>
      <wsdl:fault name="UnsupportedConstraint">
        <soap:fault name="UnsupportedConstraint" use="literal" />
      </wsdl:fault>
      <wsdl:fault name="UnsupportedSelector">
        <soap:fault name="UnsupportedSelector" use="literal" />
      </wsdl:fault>
      <wsdl:fault name="QueryError">
        <soap:fault name="QueryError" use="literal" />
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

</wsdl:definitions>
```

ANNEX G
(informative)

Change Log

Version	Date	Description
1.0.0	2009-06-22	DMTF Standard Release
1.0.1	2010-04-22	DMTF Standard Release – Fixed errors in sections 6.6.1 and 6.7.1

Bibliography

W3C, *Web Services Addressing (WS-Addressing) 1.0: Core*, May 2006, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/

W3C, *Web Services Addressing 1.0 – SOAP Binding*, May 2006, http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509

W3C, *Web Services Addressing 1.0 – WSDL Binding*, May 2006, http://www.w3.org/TR/ws-addr-wsdl/