BMC Software Inc.

Technical Disclosure Publication Document
CMDBf Registration Service Data Storage and Access

Authors

Vince Kowalski
Ken Huang

Posted: December 2010

Overview

This document describes a solution so that data registered using the standard CMDBf Registration Service can be stored and accessed in the CMDB.

Background

The problem that is solved by this invention is how to store and access data that is registered by a Management Data Repository (MDR) to a Federating CMDB. In order to be compliant with the CMDBf standard (DMTF DSP0252), the MDR must register such data using the Registration Service defined in the CMDBf standard. The Registration Service defines operations for pushing data into a CMDB, but it does not specify where that data goes, how it is stored, or how it is subsequently accessed by the Federating CMDB. This Technical Disclosure Publication Document teaches the means to do these things.

The CMDBf standard defines two architectural modes for access data stored in an MDR:

- Push Mode
- Pull Mode

In push mode, the MDR initiates the federation. Typically, the MDR is configured by selecting some data types to federate. The MDR uses the Registration Service any time this data is added, updated, or deleted to register the changed data with the CMDB. In order to do this, the mdrId, the instanceId's of each of the items and relationships registered and stored into the Federating CMDB must be preserved so that the subsequent updates or deletions from the MDR can refer to the same instanceId's. (See the pseudo-shema of the <registerRequest> element below) The registered data may be limited to identification data, but it may also include other properties as well.

```
<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>
```

In addition, the “registrationServiceMetadata” (see pseudo-schema below; details explained in chap. 8 of DSP0252_1.0.0.pdf) which is made available to the Federating CMDB at the time the “push-mode” MDR is configured, must be saved in the Federating CMDB so that the items and relationships registered can be interpreted accordingly when queries are made against the registered data.

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

In pull mode, the federating CMDB initiates the federation. Typically, the federating CMDB is configured by selecting the MDR data types that will be federated. The federating CMDB queries MDRs for instances of this data. The difference between push and pull mode is that pull mode data is retrieved only when there is a query made of the CMDB; while in push mode data is pushed to the CMDB at some arbitrary time and the data is then available for subsequent queries.

Explanation

The problem is solved by creating an embedded MDR (Management Data Repository) that is internal to the Federating CMDB. This embedded MDR is then accessed using the Query Service defined in the CMDBf Standard to access and integrate the data into the Federating CMDB. In other words, this approach keeps the data in an embedded MDR, which is accessed using standard Pull-Mode Federation concepts and components.

The advantage of this approach is that it simplifies the overall architecture. Instead of having to treat push-mode and pull-mode data as different entities, once the registration service is executed by an MDR, the data is treated as if it were pull-mode data.