HP Universal CMDB

per sistemi operativi Windows e Linux

Versione software: 9.02

Guida di riferimento per lo sviluppatore

Data di rilascio del documento: ottobre 2010
Data di rilascio del software: ottobre 2010

Note legali

Garanzia

Le sole garanzie per i prodotti e i servizi HP sono esplicitate nelle dichiarazioni di garanzia espressa che accompagnano tali prodotti e servizi. Nulla contenuto nel presente documento deve essere interpretato come garanzia aggiuntiva. HP non è responsabile di errori tecnici ed editoriali o di omissioni contenuti nel presente documento.

Le informazioni contenute nel presente documento sono soggette a modifica senza preavviso.

Legenda diritti limitati

Software riservato. Licenza valida rilasciata da HP richiesta per il possesso, l'uso e la copia. Conformemente alle disposizioni di FAR 12.211 e 12.212, il software per computer commerciale, la documentazione sul software per computer e i dati tecnici per articoli commerciali sono concessi al governo degli Stati Uniti con licenza commerciale standard del fornitore.

Note sul copyright

Note sui marchi

Adobe® e Acrobat® sono marchi di Adobe Systems Incorporated.

AMD e il simbolo AMD Arrow sono marchi di Advanced Micro Devices, Inc.

Google™ e Google Maps™ sono marchi di Google Inc.

Intel®, Itanium®, Pentium® e Intel® Xeon® sono marchi di Intel Corporation negli Stati Uniti e negli altri paesi.

Java™ è un marchio statunitense di Sun Microsystems, Inc.

Microsoft®, Windows®, Windows NT®, Windows® XP e Windows Vista® sono marchi registrati negli Stati Uniti di Microsoft Corporation.

Oracle è un marchio registrato di Oracle Corporation e/o sue affiliate.

UNIX® è un marchio registrato di The Open Group.

Riconoscimenti

- Questo prodotto include software sviluppato da Apache Software Foundation (http://www.apache.org/licenses).
- Questo prodotto include il codice OpenLDAP di OpenLDAP Foundation (http://www.openldap.org/foundation/).
- Questo prodotto include il codice GNU di Free Software Foundation, Inc. (http://www.fsf.org/).
- Questo prodotto include il codice JiBX di Dennis M. Sosnoski.
- Questo prodotto include il parser XPP3 XMLPull incluso nella distribuzione e utilizzato in JiBX di Extreme! Lab, Indiana University.
- Questo prodotto include la licenza Office Look and Feels di Robert Futrell (http://sourceforge.net/projects/officelnfs).
- Questo prodotto include il codice JEP - Java Expression Parser di Netaphor Software, Inc. (http://www.netaphor.com/home.asp).

Aggiornamenti alla documentazione

La pagina del titolo di questo documento contiene le seguenti informazioni identificative:

- Numero di versione del software che indica la versione del software.
- Data di rilascio del documento che cambia ogni volta che il documento viene aggiornato.
- Data di rilascio del software che indica la data di rilascio della versione del software.

Per cercare aggiornamenti recenti o verificare che il documento utilizzato sia il più recente, passare alla pagina:

http://h20230.www2.hp.com/selfsolve/manuals

Il sito richiede la registrazione a HP Passport e l'accesso. Per la registrazione a HP Passport, passare alla pagina:

http://h20229.www2.hp.com/passport-registration.html

Oppure fare clic sul collegamento **New users - please register** nella pagina di accesso HP Passport.

L'utente riceverà inoltre le edizioni aggiornate o nuove se si registra al relativo servizio di assistenza del prodotto. Rivolgersi al proprio rappresentante HP per i dettagli.

Supporto

Visitare il sito HP Software Support all'indirizzo:

http://www.hp.com/go/hpsoftwaresupport

Questo sito fornisce le informazioni di contatto e i dettagli sui prodotti, i servizi e l'assistenza offerti da HP Software.

L'assistenza online HP Software fornisce al cliente la possibilità di utilizzare soluzioni implementabili dall'utente mediante un accesso veloce ed efficiente agli strumenti di assistenza tecnica interattiva necessari per gestire la propria attività. In qualità di cliente qualificato dell'assistenza è possibile usufruire del sito Web di assistenza per le operazioni seguenti:

- Ricerca di documenti di interesse sulle caratteristiche
- Invio e traccia dei casi di assistenza e delle richieste di ampliamento
- Download delle patch del software
- Gestione dei contratti di assistenza
- Ricerca dei contatti di assistenza HP
- Riesame delle informazioni sui servizi disponibili
- Partecipazione alle discussioni con altri clienti software
- Ricerca e iscrizione alla formazione sul software

La maggior parte delle aree di assistenza richiedono la registrazione e l'accesso come utente di HP Passport. Molte inoltre richiedono un contratto di assistenza. Per la registrazione di un ID utente di HP Passport, visitare la pagina:

http://h20229.www2.hp.com/passport-registration.html

Per maggiori informazioni sui livelli di accesso, visitare la pagina:

http://h20230.www2.hp.com/new_access_levels.jsp

Sommario

Introduzione alla guida

In questa guida viene spiegato come creare e gestire gli adattatori che consentono di inviare e ricevere dati da repository di dati esterni e altri CMDB.

Il capitolo si suddivide in:

Organizzazione della guida

Questa guida contiene i seguenti capitoli:

Parte I Creazione di adattatori di individuazione e integrazione

Descrive come creare gli adattatori.

Parte II Utilizzo delle API

Descrive come utilizzare le API per estrarre i dati di configurazione da HP Universal CMDB.

Destinatari della guida

La guida è rivolta agli utenti seguenti di HP Universal CMDB:

- Amministratori di HP Universal CMDB
- Amministratori della piattaforma di HP Universal CMDB
- Amministratori delle applicazioni di HP Universal CMDB
- Amministratori della gestione dei dati di HP Universal CMDB

I lettori di questa guida devono avere una buona conoscenza dell'amministrazione del sistema enterprise, avere familiarità con i concetti ITIL e avere una buona conoscenza di HP Universal CMDB.

Documentazione online di HP Universal CMDB

HP Universal CMDB comprende la documentazione online seguente:

Leggimi. Fornisce un elenco delle limitazioni delle versioni e degli ultimi aggiornamenti. Dalla directory principale del DVD di HP Universal CMDB, fare doppio clic su **readme.html**. È anche possibile accedere alla versione più aggiornata del file Leggimi dal sito Web HP Software Support.

Novità. Fornisce un elenco delle nuove funzionalità e degli elementi di rilievo delle versioni. In HP Universal CMDB, selezionare **Guida** > **Novità**.

Documentazione per la stampa. Selezionare **Guida** > **Guida di UCMDB**. Le seguenti guide sono pubblicate solo in formato PDF:

- *Guida alla distribuzione di HP Universal CMDB* in PDF. Illustra i requisiti hardware e software necessari per impostare HP Universal CMDB, la procedura di installazione o di aggiornamento di HP Universal CMDB, la procedura per la protezione avanzata del sistema e la procedura per effettuare l'accesso all'applicazione.
- *Guida al database di HP Universal CMDB* in PDF. Illustra come impostare il database (MS SQL Server o Oracle) richiesto da HP Universal CMDB.

- *HP Universal CMDB Discovery and Integration Content Guide* in PDF. Illustra come eseguire l'individuazione per individuare le applicazioni, i sistemi operativi e i componenti di rete nel proprio sistema. Illustra inoltre come individuare i dati su altri repository di dati tramite l'integrazione.

La guida in linea di HP Universal CMDB comprende:

- **Modellazione**. Consente di gestire il contenuto del proprio modello Universo IT.
- **Gestione flusso di dati**. Consente di integrare HP Universal CMDB con altri repository di dati e come impostare HP Universal CMDB per individuare i componenti di rete.
- **Amministrazione di UCMDB**. Illustra come utilizzare HP Universal CMDB.
- **Riferimento per lo sviluppatore**. Per gli utenti con una conoscenza avanzata di HP Universal CMDB. Illustra come definire e utilizzare gli adattatori e come utilizzare le API per l'accesso ai dati.

La Guida in linea è disponibile dalle finestre specifiche di HP Universal CMDB facendo clic nella finestra e sul pulsante **Guida**.

È possibile visualizzare e stampare le documentazioni in linea utilizzando Adobe Reader, che può essere scaricato dal sito Web Adobe (www.adobe.com).

Tipi di argomenti

All'interno della guida, ciascuna area di interesse è organizzata in argomenti. Un argomento contiene un modulo distinto di informazioni per un oggetto. In genere gli argomenti sono classificati in base al tipo di informazioni che contengono.

Questa struttura è stata progettata per creare un accesso più semplice a informazioni specifiche, suddividendo la documentazione in tipi diversi di informazioni che possono essere necessarie di volta in volta.

Sono utilizzati tre tipi di argomenti principali: **Concetti**, **Compiti** e **Riferimenti**. I tipi di argomenti sono stati differenziati visivamente tramite l'uso di icone.

Tipo di argomento	Descrizione	Utilizzo
Concetti 	Informazioni di background, descrittive o concettuali.	Conoscere informazioni generali su una funzionalità.
Compiti 	**Compiti di istruzione.** Guida dettagliata per l'utilizzo dell'applicazione e il completamento degli obiettivi. Alcuni passaggi dei compiti comprendono esempi con l'utilizzo di dati campione. I passaggi dei compiti possono essere numerati o non numerati: ➤ **Passaggi numerati.** Compiti che vengono eseguiti seguendo ogni passaggio in ordine consecutivo. ➤ **Passaggi non numerati.** Un elenco di operazioni complete che si possono eseguire in qualsiasi ordine.	➤ Imparare il workflow completo di un compito. ➤ Seguire i passaggi elencati in un compito numerato per completare un compito. ➤ Eseguire operazioni indipendenti completando i passaggi in un compito non numerato.
	Compiti di uno scenario nei casi di utilizzo. Esempi di come eseguire un compito per una situazione specifica.	Imparare come potrebbe essere eseguito un compito in uno scenario realistico.

Tipo di argomento	Descrizione	Utilizzo
Riferimenti 	**Riferimenti generali**. Elenchi e spiegazioni dettagliati di materiale orientato ai riferimenti.	Cercare una parte specifica di informazioni sui riferimenti rilevanti per un contesto particolare.
	Riferimenti dell'interfaccia utente. Argomenti su riferimenti specializzati che descrivono una determinata interfaccia utente nel dettaglio. Selezionando **Guida per questa pagina** dal menu della Guida nel prodotto in genere si aprono gli argomenti dell'interfaccia utente.	Cercare informazioni specifiche su cosa inserire o come utilizzare uno o più elementi specifici dell'interfaccia utente, come ad esempio una finestra, una finestra di dialogo o una procedura guidata.
Risoluzione dei problemi e limitazioni 	**Risoluzione dei problemi e limitazioni**. Argomenti su riferimenti specializzati che descrivono i problemi comunemente incontrati e le relative soluzioni e che elencano le limitazioni di una funzionalità o di un'area di prodotto.	Aumentare il riconoscimento di problemi importanti prima di utilizzare una funzionalità oppure se si incontrano problemi di usabilità nel software.

Risorse aggiuntive online

Risoluzione dei problemi & Knowledge Base consente di accedere alla pagina Risoluzione dei problemi sul sito Web HP Software Support dove è possibile effettuare ricerche nella Knowledge Base con soluzioni implementabili dall'utente. Selezionare **Guida** > **Risoluzione dei problemi & Knowledge Base**. L'URL del sito Web è http://h20230.www2.hp.com/troubleshooting.jsp.

HP Software Support consente di accedere al sito Web HP Software Support. Il sito consente di esplorare la knowledge base con soluzioni implementabili dall'utente. È inoltre possibile registrare un messaggio o cercare forum di discussione tra utenti, inoltrare richieste di assistenza, scaricare patch e documentazione aggiornata e così via. Selezionare **Guida** > **HP Software Support**. L'URL del sito Web è www.hp.com/go/hpsoftwaresupport.

La maggior parte delle aree di assistenza richiedono la registrazione e l'accesso come utente di HP Passport. Molte inoltre richiedono un contratto di assistenza.

Per maggiori informazioni sui livelli di accesso, visitare la pagina:

http://h20230.www2.hp.com/new_access_levels.jsp

Per la registrazione di un ID utente di HP Passport, visitare la pagina:

http://h20229.www2.hp.com/passport-registration.html

Il sito Web HP Software consente di accedere al sito Web HP Software. In questo sito sono presenti le informazioni più aggiornate sui prodotti software HP. Sono compresi anche nuove versioni software, seminari, fiere, assistenza clienti e così via. Selezionare **Guida** > **Sito Web HP Software**. L'URL del sito Web è www.hp.com/go/software.

Aggiornamenti della documentazione

HP Software aggiorna costantemente la documentazione sui prodotti aggiungendo nuove informazioni.

Per cercare aggiornamenti recenti o verificare che il documento utilizzato sia il più recente, visitare il sito Web sui manuali dei prodotti software HP (http://h20230.www2.hp.com/selfsolve/manuals).

Parte I

Creazione di adattatori di individuazione e integrazione

1

Sviluppo e scrittura dell'adattatore

Questo capitolo comprende:

Concetti

Compiti

Concetti

Scrittura e sviluppo dell'adattatore - Panoramica

Prima di iniziare la pianificazione effettiva per lo sviluppo di nuovi adattatori, è importante comprendere i processi e le interazioni comunemente associati a questo sviluppo.

Le sezioni seguenti posso aiutare a capire cosa è necessario sapere e fare per gestire ed eseguire correttamente un progetto di sviluppo di individuazione.

Questo capitolo:

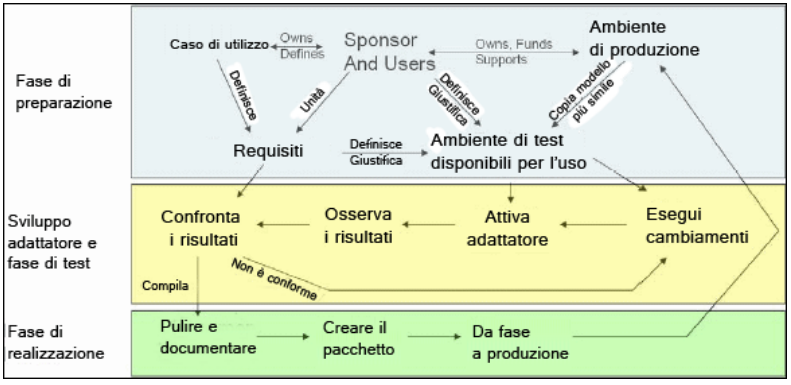
- Presuppone una conoscenza del funzionamento di HP Universal CMDB e una dimestichezza di base con gli elementi del sistema. Il suo scopo è quello di assistere durante il processo di apprendimento e non intende rappresentare una guida completa.
- Tratta le fasi di pianificazione, ricerca e implementazione del nuovo contenuto di individuazione per HP Universal CMDB insieme alle linee guida e alle considerazioni da tenere a mente.
- Fornisce informazioni sulle API principali del Framework di Gestione flusso di dati. Per la documentazione completa sulle API disponibili consultare *Riferimento API di Gestione flusso di dati di HP Universal CMDB*. (Esistono altre API non formali che, sebbene vengano utilizzate su adattatori predefiniti, potrebbero essere soggette a modifiche.)

Creazione del contenuto

In questa sezione vengono trattati i seguenti argomenti:

Ciclo di sviluppo dell'adattatore

L'illustrazione seguente mostra un diagramma di flusso per la scrittura dell'adattatore. La sezione centrale è quella che richiede più tempo poiché si tratta di un ciclo iterativo di sviluppo e test.



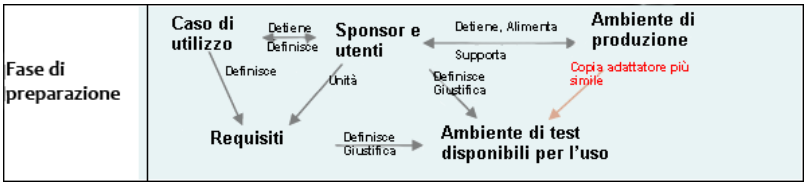
Ciascuna fase di sviluppo dell'adattatore si basa sulla precedente.

Dopo aver trovato l'aspetto e il funzionamento adeguati, è possibile creare il pacchetto. Utilizzando Gestione pacchetti di UCMDB o l'esportazione manuale dei componenti, creare un file *.zip del pacchetto. Come procedura consigliata, è necessario distribuire e testare questo pacchetto su un altro sistema UCMDB prima di rilasciarlo alla produzione, per accertarsi che tutti i componenti vengano controllati e che la creazione del relativo pacchetto sia corretta. Per i dettagli sulla creazione del pacchetto consultare "Gestione pacchetti" nella *Guida all'amministrazione di HP Universal CMDB*.

Le sezioni seguenti trattano ciascuna delle fasi e indicano i passaggi più importanti e le procedure consigliate:

- Fase di ricerca e preparazione
- Sviluppo e test dell'adattatore
- Creazione del pacchetto e realizzazione dell'adattatore

Fase di ricerca e preparazione



La fase di Ricerca e preparazione comprende i casi di utilizzo e le esigenze aziendali principali nonché i controlli per la protezione degli impianti necessari per sviluppare e testare l'adattatore.

1. Quando si intende modificare un adattatore esistente, il primo passaggio tecnico è quello di eseguire una copia di backup di quell'adattatore accertandosi che sia possibile ripristinarne lo stato originale. Se si prevede di creare un nuovo adattatore, copiare l'adattatore più simile e salvarlo con un nome appropriato. Per i dettagli consultare "Riquadro Risorse" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.
2. Cercare la modalità di raccolta dati dell'adattatore:
   - Utilizzare i protocolli/strumenti esterni per ottenere i dati
   - Sviluppare la modalità di creazione di CI basati sui dati da parte dell'adattatore
   - Adesso è possibile avere un'idea di quale potrebbe essere l'aspetto di un adattatore simile
3. Determinare l'adattatore simile in base a:
   - Stessi CI creati
   - Stessi protocolli utilizzati (SNMP)
   - Stesso tipo di destinazioni (per tipo di sistema operativo, versioni ecc.)

4 Copiare l'intero pacchetto.

5 Eseguire la decompressione nello spazio di lavoro e rinominare i file dell'adattatore (XML) e Jython (.py).

Sviluppo e test dell'adattatore

La fase di sviluppo e test dell'adattatore è un processo altamente iterativo. Quando l'adattatore comincia a prendere forma, iniziare il test rispetto ai casi di utilizzo finali, apportare modifiche, eseguire nuovamente il test e ripetere questo processo fino a quando l'adattatore non sia conforme ai requisiti.

Avvio e preparazione della copia

- Modificare tutte le parti XML dell'adattatore: Nome (id) nella riga 1, Tipi CI creati e nome script Jython chiamato.
- Eseguire la copia con risultati identici all'adattatore originale.
- Convertire in commento la maggior parte del codice, specialmente il codice che produce i risultati principali.

Sviluppo e test

- Utilizzare un altro codice di esempio per sviluppare i cambiamenti
- Testare l'adattatore eseguendolo
- Utilizzare una vista dedicata per convalidare risultati complessi e cercare di convalidare i risultati semplici

Creazione del pacchetto e realizzazione dell'adattatore

La fase di creazione del pacchetto e realizzazione dell'adattatore riguarda l'ultima fase di sviluppo. Come procedura consigliata, è necessario un passo finale per rimuovere gli scarti di debug, i documenti e i commenti e per eventuali considerazioni sulla protezione e così via prima di passare alla creazione del pacchetto. È sempre necessario avere un documento Leggimi per spiegare i funzionamenti interni dell'adattatore. Qualcuno (forse anche voi), in futuro, potrebbe avere bisogno di questo adattatore e anche la minima documentazione potrebbe essere di grande aiuto.

Pulire e documentare

- Rimuovere il debug
- Commentare tutte le funzioni e aggiungere alcuni commenti di apertura nella sezione principale
- Creare una TQL e una vista di esempio da far testare all'utente

Creare il pacchetto

- Esportare gli adattatori, la TQL e così via con Gestione pacchetti. Per i dettagli consultare "Gestione pacchetti" nella *Guida all'amministrazione di HP Universal CMDB*.
- Controllare eventuali dipendenze tra il pacchetto interessato e altri pacchetti, ad esempio se i CI creati da questi pacchetti sono CI di input dell'adattatore.
- Utilizzare Gestione pacchetti per creare uno zip del pacchetto. Per i dettagli consultare "Gestione pacchetti" nella *Guida all'amministrazione di HP Universal CMDB*.
- Testare la distribuzione rimuovendo parti del nuovo contenuto e rieseguendo la distribuzione o effettuando la distribuzione su un altro sistema di prova.

Gestione flusso di dati e integrazione

Gli adattatori GFD possono essere integrati con altri prodotti. Tenere presenti le seguenti definizioni:

- GFD raccoglie il contenuto specifico da molte destinazioni.
- L'integrazione raccoglie diversi tipi di contenuto da un sistema.

Tenere presente che queste definizioni non distinguono tra un metodo di raccolta e un altro. Neanche GFD. Il processo di sviluppo di un nuovo adattatore è lo stesso processo di sviluppo di una nuova integrazione. È possibile eseguire le stesse ricerche, effettuare le stesse scelte sia per gli adattatori nuovi sia per quelli esistenti, scrivere gli adattatori allo stesso modo e così via. Ci sono solo alcune piccole differenze:

- La pianificazione dell'adattatore finale. Gli adattatori di integrazione possono essere eseguiti più frequentemente rispetto all'individuazione, ma ciò dipende dai casi di utilizzo:
- CI di input:
  - Integrazione: CI non trigger da eseguire senza input: un nome file o un'origine viene passato attraverso il parametro dell'adattatore.
  - Individuazione: utilizza CI normali di CMDB per l'input.

Per i progetti di integrazione, è necessario riutilizzare quasi sempre un adattatore esistente. La direzione di integrazione (da HP Universal CMDB a un altro prodotto o da un altro prodotto a HP Universal CMDB) potrebbe influenzare l'approccio allo sviluppo. Ci sono pacchetti pronti/modello che è possibile copiare per uso personale, utilizzando tecniche comprovate.

Da HP Universal CMDB a un altro progetto:

- Creare una TQL che produca i CI e le relazioni da esportare.
- Utilizzare un adattatore wrapper generico per eseguire la TQL e scrivere i risultati in un file XML per il prodotto esterno da leggere.

Nota: per gli esempi di pacchetti pronti/modello, rivolgersi a HP Software Support.

Per integrare un altro prodotto in HP Universal CMDB: a seconda della modalità di esposizione dei dati dell'altro prodotto, l'adattatore di integrazione agisce in modo differente:

Tipo di integrazione	Esempio di riferimento da riutilizzare
Accedere direttamente al database del prodotto	HP ED
Memorizzare un file csv o xml prodotto da un'esportazione	HP ServiceCenter
Accedere all'API di un prodotto	BMC Atrium/Remedy

Associazione del valore aziendale con lo sviluppo di individuazione

Il caso di utilizzo per lo sviluppo di un nuovo contenuto di individuazione deve essere guidato da un caso aziendale e produrre un valore aziendale. Ovvero, l'obiettivo della mappatura dei componenti di sistema su CI e della loro aggiunta a CMDB è quello di fornire valore aziendale.

Non è sempre possibile utilizzare il contenuto per la mappatura dell'applicazione, sebbene ciò rappresenti un passaggio intermedio comune a molti casi di utilizzo. Indipendentemente dall'utilizzo finale del contenuto, la pianificazione deve rispondere alle seguenti domande di questo approccio:

- Chi è il consumatore? Come deve agire il consumatore in base alle informazioni fornite dai CI (e dalle relazioni tra questi)? Qual è il contesto aziendale in cui i CI e le relazioni devono essere visti? Il consumatore di questi CI è una persona, un prodotto o entrambi?
- Una volta stabilita la perfetta combinazione di CI e relazioni in CMDB, come devo utilizzarli per produrre valore aziendale?
- Quale dovrebbe essere l'aspetto ideale della mappatura?
  - Quale termine descrive meglio le relazioni tra ciascun CI?
  - Quale tipo di CI sarebbe il più importante da includere?
  - Qual è l'utilizzo finale e l'utente finale della mappa?
- Qual è il layout perfetto del report?

Una volta stabilita la giustificazione aziendale, il passaggio successivo è quello di integrare il valore aziendale in un documento. Ciò significa tracciare la mappa perfetta con uno strumento di disegno e comprendere l'impatto e le dipendenze tra CI, i report, come i cambiamenti vengono monitorati, quale cambiamento è importante, il monitoraggio, la conformità e il valore aziendale aggiunto come richiesto dai casi di utilizzo.

Questo disegno (o modello) viene denominato **progetto**.

Ad esempio, se è fondamentale che l'applicazione conosca quando un determinato file di configurazione viene cambiato, il file deve essere mappato e collegato al CI adeguato (al quale è correlato) nella mappa disegnata.

Collaborare con uno SME (Subject Matter Expert, esperto dell'argomento) dell'area, che è l'utente finale del contenuto sviluppato. Per fornire valore aziendale, l'esperto deve indicare le entità fondamentali (CI con attributi e relazioni) che devono esistere nel CMDB.

Un metodo potrebbe essere quello di fornire un questionario al proprietario dell'applicazione (alche allo SME, in questo caso). Il proprietario deve essere in grado di specificare il progetto e gli obiettivi suddetti. Il proprietario deve fornire almeno un'architettura corrente dell'applicazione.

È necessario mappare esclusivamente i dati fondamentali e non quelli non necessari: è sempre possibile ampliare l'adattatore in un secondo momento. L'obiettivo deve essere quello di configurare un'individuazione limitata che funzioni e fornisca valore. La mappatura di grandi quantità di dati fornisce mappe più imponenti ma il relativo sviluppo può essere fuorviante e richiedere molto tempo.

Una volta che il modello e il valore aziendale sono chiari, procedere al passaggio successivo. Questa fase può essere rivisitata quando vengono fornite informazioni più concrete dalle fasi successive.

Ricerca dei requisiti di integrazione

Il prerequisito di questa fase è un **progetto** dei CI e delle relazioni da far individuare a GFD, il quale deve includere gli attributi da individuare. Per i dettagli consultare "Scrittura e sviluppo dell'adattatore - Panoramica" a pag. 20.

In questa sezione vengono trattati i seguenti argomenti:

Modifica di un adattatore esistente

È possibile modificare un adattatore esistente quando esiste un adattatore pronto o predefinito, tuttavia:

- non individua gli attributi specifici necessari
- un tipo specifico di destinazione (OS) non viene individuato o viene individuato in modo errato
- una relazione specifica non viene individuata o creata

Se un adattatore esistente esegue lo stesso, ma non tutto, il processo, il primo approccio deve essere quello di valutare gli adattatori esistenti e verificare se uno di questi esegue almeno ciò che è necessario; se sì, è possibile modificare l'adattatore esistente.

È necessario valutare se è disponibile un adattatore pronto esistente. Gli adattatori pronti sono adattatori di individuazione disponibili ma non predefiniti. Rivolgersi a HP Software Support per ricevere l'elenco aggiornato degli adattatori pronti.

Scrittura di un nuovo adattatore

È necessario sviluppare un nuovo adattatore:

- Quando è più rapido scrivere un adattatore anziché inserire manualmente le informazioni nel CMDB (generalmente, da circa 50 a 100 CI e relazioni) oppure non si tratta di un'operazione isolata.
- Quando la necessità giustifica lo sforzo.
- Se nono sono disponibili adattatori predefiniti o pronti.
- Se non è possibile riutilizzare i risultati.
- Quando l'ambiente di destinazione o i relativi dati sono disponibili (non è possibile individuare ciò che non si vede).

Ricerca del modello

- Esplorare il modello di classe di UCMDB (Gestione tipo CI) e creare una corrispondenza tra le entità e le relazioni del proprio **progetto** e i CIT esistenti. Si consiglia di attenersi al modello corrente per evitare possibili complicazioni durante l'aggiornamento della versione. Se è necessario estendere il modello, creare nuovi CIT poiché l'aggiornamento potrebbe sovrascrivere i CIT predefiniti.
- Se alcuni attributi, entità o relazioni mancano dal modello corrente, è necessario crearli. È preferibile creare un pacchetto con questi CIT (che, anche successivamente, conterranno tutta l'individuazione, le viste e altri elementi correlati a questo pacchetto) poiché è necessario poter distribuire questi CIT sull'installazione di HP Universal CMDB.

Ricerca della tecnologia

Una volta verificato che il CMDB contiene i CI pertinenti, la fase successiva è decidere come recuperare questi dati dai sistemi correlati.

Il recupero dei dati include solitamente l'utilizzo di un protocollo per accedere alla parte di gestione dell'applicazione, ai dati effettivi dell'applicazione o ai file o database di configurazione correlati all'applicazione. Qualsiasi origine dati in grado di fornire informazioni su un sistema è preziosa. La ricerca della tecnologia richiede una profonda conoscenza del sistema in questione e talvolta creatività.

Per le applicazioni "fatte in casa", potrebbe essere utile fornire un modulo di questionario al proprietario dell'applicazione. In questo modulo, il proprietario deve elencare tutte le aree dell'applicazione in grado di fornire le informazioni necessarie per il progetto e i valori aziendali. Queste informazioni devono includere (ma non devono essere limitate a) i database di gestione, i file di configurazione, i file di registro, le interfacce di gestione, i programmi di amministrazione, i servizi Web, i messaggi o gli eventi inviati e così via.

Per i prodotti predefiniti, è necessario focalizzare l'attenzione sulla documentazione, sui forum o sull'assistenza del prodotto. Cercare guide di amministrazione, guide di integrazione e plug-in, guide di gestione e così via. Se ci sono ancora dati mancanti dalle interfacce di gestione, leggere le informazioni sui file di configurazione dell'applicazione, voci di registro, file di registro, registri eventi NT di eventuali elementi dell'applicazione che ne controllano il corretto funzionamento.

Linee guida per la scelta delle modalità di accesso ai dati

Pertinenza: selezionare le origini o una combinazione di origini che forniscono la maggior parte dei dati. Se una singola origine fornisce la maggior parte delle informazioni mentre l'accesso al resto delle informazioni è sporadico o difficile, tentare di valutare il valore delle informazioni restanti confrontandolo allo sforzo o al rischio necessari per ottenerle. Talvolta, è possibile decidere di ridurre il progetto se il valore o il costo non garantisce lo sforzo investito.

Riutilizzo: se HP Universal CMDB include già un supporto del protocollo di connessione specifico è una buona idea utilizzarlo. Ciò significa che il framework GFD è in grado di fornire un client già pronto e una configurazione per la connessione. In caso contrario, potrebbe essere necessario investire nello sviluppo dell'infrastruttura. È possibile visualizzare i protocolli di connessione attualmente supportati di HP Universal CMDB: **riquadro Individuazione > Imposta Discovery Probe > Domini e sonde**. Per i dettagli consultare "Riquadro Domini e sonde" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

È possibile aggiungere nuovi protocolli aggiungendo nuovi CI al modello. Per i dettagli rivolgersi a HP Software Support.

Nota: per accedere ai dati del registro di sistema di Windows, è possibile utilizzare WMI o NTCmd.

Protezione: l'accesso alle informazioni richiede solitamente le credenziali (nome utente, password), immesse nel CMDB e protette in tutto il prodotto. Se possibile e se l'aggiunta di protezione non entra in conflitto con gli altri principi impostati, scegliere la credenziale o il protocollo meno vulnerabile che risponde ancora ai requisiti di accesso. Ad esempio, se le informazioni sono disponibili sia tramite JMX (interfaccia di amministrazione standard, limitata) sia tramite Telnet, è preferibile utilizzare JMX poiché offre implicitamente accesso limitato e (solitamente) nessun accesso alla piattaforma sottostante.

Praticità: alcune interfacce di gestione possono includere funzioni più avanzate. Ad esempio, potrebbe essere più facile eseguire query (SQL, WMI) che esplorare strutture di informazioni o creare espressioni regolari per l'analisi.

Gruppo di destinatari dello sviluppatore: le persone che infine svilupperanno gli adattatori possono avere un'inclinazione a favore di una determinata tecnologia. Ciò può essere preso in considerazione se due tecnologie forniscono quasi le stesse informazioni a un costo uguale in altri fattori.

Riepilogo

Il risultato di questa fase è un documento che descrive i metodi di accesso e le relative informazioni che è possibile estrarre da ciascun metodo. Il documento deve inoltre contenere una mappatura da ciascuna origine a ciascun dato del progetto pertinente.

Ogni metodo di accesso deve essere contrassegnato secondo le istruzioni suddette. Infine, è necessario disporre di un piano per definire quali origini individuare e quali informazioni estrarre da ciascuna origine nel modello del progetto (che da questo momento è stato mappato sul modello UCMDB corrispondente).

Sviluppo del contenuto di integrazione

Prima di creare una nuova integrazione, è necessario comprendere quali siano i requisiti dell'integrazione:

- L'integrazione deve copiare i dati nel CMDB? I dati devono essere monitorati tramite cronologia? L'origine è inaffidabile?

  Popolamento necessario.

- L'integrazione deve federare i dati in tempo reale per le viste e le query TQL? La precisione dei cambiamenti apportati ai dati è fondamentale? La quantità dei dati da copiare nel CMDB è troppo grande ma la quantità dei dati richiesta è solitamente piccola?

  Federazione necessaria.

- L'integrazione deve inviare i dati alle origini dati remote?

  Invio dati necessario.

Nota: i flussi di federazione e popolamento possono essere configurati per la stessa integrazione, per il massimo livello di flessibilità.

Per i dettagli sui diversi tipi di integrazione consultare "Studio di integrazione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

Per la creazione degli adattatori di integrazione sono disponibili quattro diverse opzioni:

- Adattatore Jython
  - La sequenza di individuazione classica
  - Scritto in Jython
  - Utilizzato per il popolamento

  Per i dettagli consultare "Sviluppo degli adattatori Jython" a pag. 63.

- Adattatore Java
  - Un adattatore che implementa una delle interfacce dell'adattatore nel Framework SDK di federazione.
  - Può essere utilizzato per una o più federazioni, popolamenti o invii dati (a seconda dell'implementazione richiesta).
  - Scritto da zero in Java, consente la scrittura del codice che si connetterà a un'eventuale origine o destinazione.
  - Adatto per processi ciascuno dei quali si connette a una singola origine o destinazione dati.

  Per i dettagli consultare "Sviluppo degli adattatori Java" a pag. 221.
- Adattatore DB generico
  - Un adattatore astratto basato sull'adattatore Java e che utilizza il Framework SDK di federazione.
  - Consente la creazione di adattatori che si connettono a repository di dati esterni.
  - Supporta sia la federazione sia il popolamento (con un plug-in Java implementato per il supporto dei cambiamenti).
  - Relativamente facile da definire poiché si basa principalmente su file XML e di configurazione proprietà.
  - La configurazione principale è basata su un file **orm.xml** che esegue la mappatura tra le colonne del database e le classi di UCMDB.
  - Adatto per processi ciascuno dei quali si connette a una singola origine dati.

  Per i dettagli consultare "Sviluppo degli adattatori generici del database" a pag. 127.
- Adattatore Push generico
  - Un adattatore astratto basato sull'adattatore Java (il Framework SDK di federazione) e sull'adattatore Jython.
  - Consente la creazione di adattatori che inviano dati a destinazioni remote.

- Relativamente facile da definire poiché è necessario definire esclusivamente la mappatura tra le classi di UCMDB e l'XML e uno script Jython che invia i dati alla destinazione.
- Adatto per processi ciascuno dei quali si connette a una singola destinazione dati.
- Utilizzato per l'invio dati.

Per i dettagli consultare "Sviluppo degli adattatori Push" a pag. 261.

La tabella seguente visualizza le capacità di ciascun adattatore:

Flusso/ Adattatore	Adattatore Jython	Adattatore Java	Adattatore GDB	Adattatore Push
Popolamento	X	X	X	
Federazione		X	X	
Invio dati		X		X

Sviluppo del contenuto di individuazione

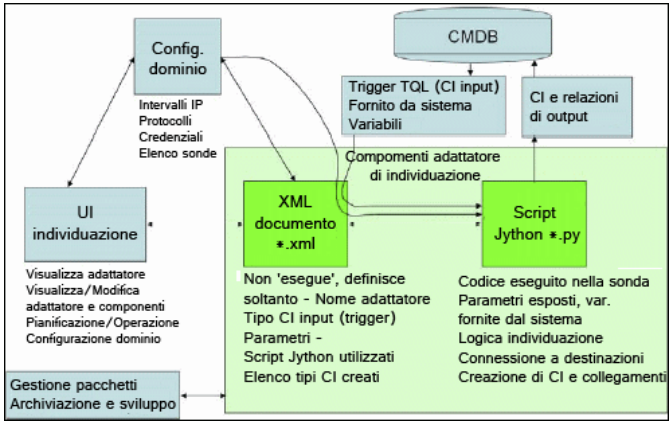
In questa sezione vengono trattati i seguenti argomenti:

➤ "Adattatori di individuazione e componenti correlati" a pag. 35

➤ "Separazione di adattatori" a pag. 37

Adattatori di individuazione e componenti correlati

Il diagramma seguente mostra i componenti di un adattatore e i componenti con cui interagiscono per eseguire l'individuazione. I componenti in verde sono gli adattatori effettivi e i componenti in blu sono i componenti che interagiscono con gli adattatori.



Tenere presente che la nozione minima di un adattatore è due file: un documento XML e uno script Jython. Il framework di individuazione che include i CI di input, le credenziali e le librerie fornite dall'utente è esposto all'adattatore durante l'esecuzione. Entrambi i componenti dell'adattatore di individuazione sono gestiti tramite Gestione flusso di dati. Sono archiviati in base all'operazione nel CMDB stesso; sebbene il pacchetto esterno rimanga, non viene comunque utilizzato per l'operazione. Gestione pacchetti consente di conservare la nuova individuazione e la capacità del contenuto di integrazione.

I CI di input per l'adattatore vengono forniti da una TQL e sono presentati allo script dell'adattatore in variabili fornite dal sistema. I parametri dell'adattatore vengono anche forniti come dati di destinazione, pertanto è possibile configurare l'operazione dell'adattatore in base a una funzione specifica dell'adattatore.

L'applicazione GFD viene utilizzata per creare ed eseguire il test dei nuovi adattatori. Utilizzare le pagine Pannello di controllo dell´individuazione, Gestione adattatore e Impostazione della sonda del flusso di dati durante la scrittura dell'adattatore.

Gli adattatori vengono archiviati e trasportati come pacchetti. L'applicazione Gestione pacchetti e la console JMX sono utilizzate per creare pacchetti dai nuovi adattatori e per distribuire gli adattatori sui nuovi sistemi.

Separazione di adattatori

Tecnicamente, è possibile definire un'intera individuazione in un singolo adattatore. Un buon progetto richiede tuttavia che un sistema complesso sia separato in componenti più semplici e gestibili.

Di seguito sono indicate le linee guida e le procedure consigliate per dividere il processo dell'adattatore:

- L'individuazione deve essere eseguita in fasi. Ciascuna fase deve essere rappresentata da un adattatore che deve mappare un'area o un livello del sistema. Gli adattatori, per continuare l'individuazione del sistema, devono basarsi sulla fase o livello precedente da individuare. Ad esempio, l'adattatore A viene è attivato da un risultato TQL del server applicazioni ed esegue la mappatura del livello del server applicazioni. Come parte di questa mappatura, viene mappato un componente di connessione JDBC. L'adattatore A registra un componente di connessione JDBC come TQL trigger e utilizza i risultati dell'adattatore A per accedere al livello del database (ad esempio tramite l'attributo JDBC URL) ed esegue la mappatura del livello del database.
- **Il paradigma di connessione a due fasi**: la maggior parte dei sistemi richiede credenziali per accedere ai relativi dati. Ciò significa che è necessario tentare una combinazione utente/password rispetto a questi sistemi. L'amministratore GFD fornisce le informazioni sulle credenziali in modo sicuro al sistema e può attribuire diverse credenziali di accesso con priorità. Ciò viene denominato **Dizionario di protocollo**. Se il sistema non è accessibile (per un motivo qualsiasi) non c'è motivo di proseguire l'individuazione. Se la connessione è corretta, ci deve essere un modo per indicare quale insieme di credenziali è stato utilizzato correttamente per l'eventuale accesso futuro all'individuazione.

Queste due fasi portano a una separazione dei due adattatori nei casi seguenti:

- **Adattatore di connessione**: questo è un adattatore che accetta un trigger iniziale e cerca l'esistenza di un agente remoto su quel trigger. Esegue questa operazione provando tutte le voci nel Dizionario di protocollo corrispondenti a questo tipo di agente. Se corretta, questo adattatore fornisce come suo risultato un CI agente remoto (SNMP, WMI e così via) che indica inoltre la voce corretta nel Dizionario di protocollo per le connessioni future. Questo CI agente fa quindi parte di un trigger per l'adattatore di contenuto.
- **Adattatore di contenuto**: la condizione preliminare di questo adattatore è la connessione corretta dell'adattatore precedente (condizioni preliminari specificate dalle TQL). Questi tipi di adattatori non devono più esplorare tutto il Dizionario di protocollo poiché hanno un modo di ottenere le credenziali corrette dal CI agente remoto e le utilizzano per accedere al sistema individuato.

- Considerazioni di pianificazione diverse possono anche influenzare la divisione dell'individuazione. Ad esempio, un sistema potrebbe essere richiesto solo durante le ore di disattivazione, pertanto, anche se avrebbe senso unire l'adattatore allo stesso adattatore che individua un altro sistema, le pianificazioni diverse indicano che è necessario creare due adattatori.
- L'individuazione di tecnologie o interfacce di gestione diverse per individuare lo stesso sistema deve essere dislocata in due adattatori separati. Ciò consente di attivare il metodo di accesso appropriato per ciascun sistema od organizzazione. Ad esempio, alcune organizzazioni dispongono dell'accesso WMI ai computer ma non degli agenti SNMP installati.

Compiti

Implementare un adattatore di individuazione

Un compito GFD ha lo scopo di accedere ai sistemi remoti (o locali), modellando i dati come CI e salvando i CI nel CMDB. Il compito è composto dai seguenti passaggi:

1 **Adattatore GFD.**

È possibile configurare un file adattatore contenente il contesto, i parametri e i tipi di risultato selezionando gli script che fanno parte dell'adattatore. Per i dettagli consultare la sezione seguente .

2 **Processo di individuazione.**

È possibile configurare un processo con le informazioni di pianificazione e una TQL trigger. Per i dettagli consultare "Passaggio 2: Assegnare un processo all'adattatore" a pag. 50.

3 **Codice di individuazione.**

È possibile modificare un codice Jython o Java contenuto nei file dell'adattatore e che si riferisce al Framework GFD. Per i dettagli consultare "Passaggio 3: Creare il codice Jython" a pag. 51.

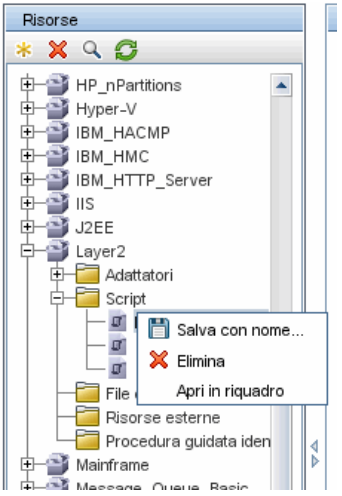
Per scrivere nuovi adattatori, creare ciascuno dei componenti suddetti, ognuno dei quali è legato automaticamente al componente del passaggio precedente. Ad esempio, una volta creato un processo e selezionato l'adattatore pertinente, il file si lega al processo.

Codice adattatore

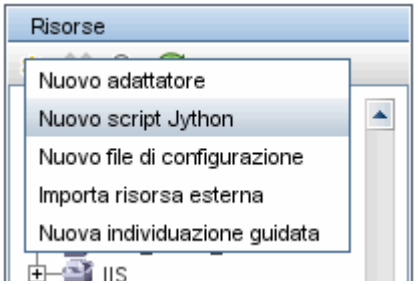
L'implementazione effettiva della connessione al sistema remoto, della richiesta dei relativi dati e relativa mappatura come dati di CMDB viene eseguita dal codice Jython. Ad esempio, il codice contiene la logica per la connessione a un database e per l'estrazione dei dati da quest'ultimo. In tal caso, il codice si aspetta di ricevere un URL JDBC, un nome utente, una password, una porta e così via. Questi parametri sono specifici di ciascuna istanza del database che risponde alla query TQL. È possibile definire queste variabili nell'adattatore (nei dati CI trigger) e, quando il processo viene eseguito, questi dettagli specifici vengono passati al codice per l'esecuzione.

L'adattatore può riferirsi a questo codice tramite un nome classe Java o un nome script Jython. In questa sezione viene trattata la scrittura del codice GFD come script Jython.

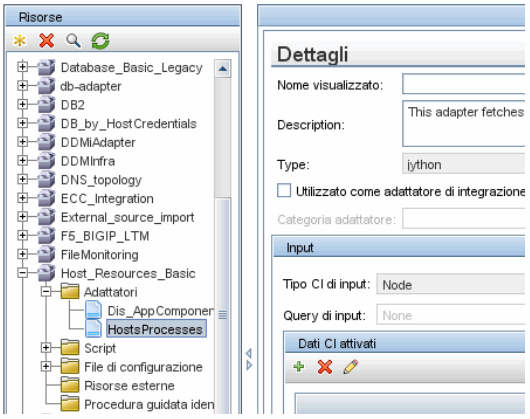
Un adattatore può contenere un elenco di script da utilizzare durante l'esecuzione dell'individuazione. Quando si crea un nuovo adattatore, viene creato solitamente un nuovo script che viene assegnato all'adattatore. Un nuovo script include gli esemplari di base, tuttavia è possibile utilizzare uno degli altri script come esemplare facendo clic su di esso con il pulsante destro del mouse e selezionando **Salva con nome**:



Per i dettagli sulla scrittura di nuovi script Jython consultare "Passaggio 3: Creare il codice Jython" a pag. 51. Aggiungere gli script tramite il riquadro Risorse:



Gli script nell'elenco vengono eseguiti uno dopo l'altro, nella stessa sequenza in cui sono definiti nell'adattatore:



Nota: è necessario specificare uno script sebbene venga utilizzato esclusivamente come libreria da un altro script. In tal caso, lo script libreria deve essere definito prima che venga utilizzato dallo script. In questo esempio, lo script processdbutils.py è una libreria utilizzata dall'ultimo script host_processes.py. Le librerie si distinguono dai normali script eseguibili per la mancanza della funzione DiscoveryMain()

Passaggio 1: Creare un adattatore

Un adattatore può essere considerato come la definizione di una funzione. Questa funzione definisce una definizione di input, esegue la logica sull'input, definisce l'output e fornisce un risultato.

Ciascun adattatore specifica l'input e l'output: sia l'input sia l'output sono CI trigger definiti specificatamente nell'adattatore. L'adattatore estrae i dati dal CI trigger di input e passa questi dati al codice sotto forma di parametri. (I dati dai CI correlati vengono talvolta passati anche al codice. Per i dettagli consultare "Finestra CI correlati" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.) Un codice adattatore è generico, tranne per questi parametri specifici del CI trigger di input che vengono passati al codice.

Per i dettagli sui componenti di input consultare "CI trigger e query trigger" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

In questa sezione vengono trattati i seguenti argomenti:

Definire l'input dell'adattatore (CIT trigger e query di input)

È possibile utilizzare i componenti CIT trigger e Query di input per definire CI specifici come input dell'adattatore:

➤ Il CIT trigger definisce quale CIT viene utilizzato come input per l'adattatore. Ad esempio, per un adattatore che dovrà individuare IP, il CIT di input è Network.

➤ La query di input è una query normale e modificabile che definisce la query rispetto al CMDB. La query di input definisce vincoli aggiuntivi sul CIT (ad esempio, se il compito richiede un attributo hostID o application_ip) e può definire più dati CI, se richiesto dall'adattatore.

Se l'adattatore richiede informazioni aggiuntive dai CI correlati al CI trigger, è possibile aggiungere nodi aggiuntivi alla TQL di input. Per i dettagli consultare "Esempio di definizione della query di input" a pag. 44 nel "Aggiungere nodi query e relazioni a una query TQL" e nella *Guida alla modellazione di HP Universal CMDB*.

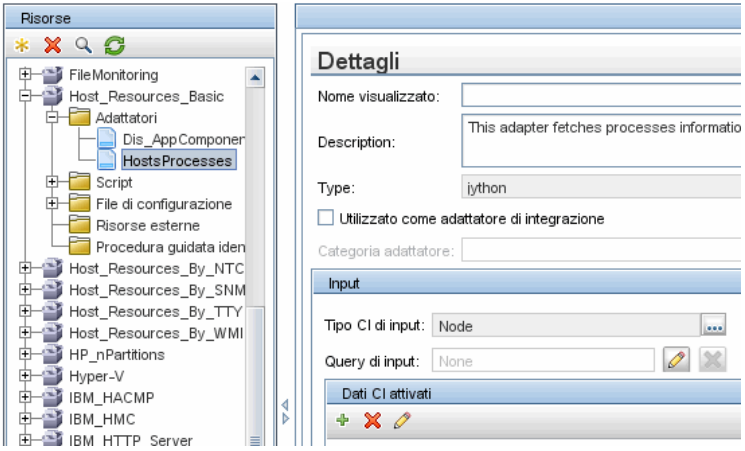
➤ I dati del CI trigger contengono tutte le informazioni richieste sul CI trigger nonché le informazioni da altri nodi nella TQL di input, se definiti. GFD utilizza le variabili per recuperare i dati dai CI. Quando il compito viene scaricato sulla sonda, le variabili dei dati del CI trigger vengono sostituite dai valori effettivi esistenti sugli attributi per le istanze CI reali.

Esempio di definizione del CIT trigger:

In questo esempio, un CIT trigger definisce che nell'adattatore sono consentiti i CI IP.

1. Accedere a **Gestione flusso di dati** > **Gestione adattatore**. Selezionare l'adattatore HostProcesses (**Pacchetti** > **Host_Resources_Basic** > **Adattatori** > **HostProcesses**).
2. Individuare la casella Tipo CI di input. Per i dettagli consultare "Dati CI attivati" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.
3. Fare clic sul pulsante per aprire la finestra di dialogo Scegli classe individuata. Per i dettagli consultare "Finestra di dialogo Scegli classe individuata" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.
4. Selezionare il CIT.

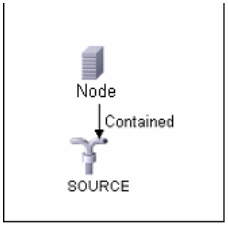
   In questo esempio, il CI IP (Host) è autorizzato nell'adattatore:



Esempio di definizione della query di input

In questo esempio, la query TQL di input definisce che il CI IP (configurato nell'esempio precedente come CIT trigger) deve essere connesso a un CI Host.

1. Accedere a **Gestione flusso di dati** > **Gestione adattatore**. Individuare la casella TQL di input. Fare clic sul pulsante **Modifica** per aprire l'editor TQL di input. Per i dettagli consultare "Finestra Editor di query di input" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.
2. Nell'editor TQL di input, denominare il nodo CI trigger **SOURCE**: fare clic con il pulsante destro del mouse sul nodo e scegliere **Proprietà nodo**. Nella casella **Nome elemento**, cambiare il nome in **SOURCE**.
3. Aggiungere un CI Host e una relazione Contains al CI IP. Per i dettagli sull'utilizzo dell'editor TQL di input, consultare "Finestra Editor di query di input" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.



Il CI **IP** è connesso a un CI **HOST**. La TQL di input è composta da due nodi, **HOST** e **IP**, con un collegamento tra loro. Il CI **IP** è denominato **SOURCE**.

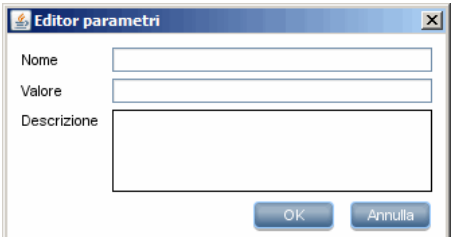
Esempio di aggiunta variabili alla query TQL di input:

In questo esempio, le variabili DIRECTORY e CONFIGURATION_FILE vengono aggiunte alla query TQL di input creata nell'esempio precedente. Queste variabili aiutano a definire cosa deve essere individuato, in questo caso, per trovare i file di configurazione ubicati sugli host collegati agli IP da individuare.

1 Visualizzare la TQL di input creata nell'esempio precedente.

Accedere a **Gestione flusso di dati** > **Gestione adattatore**. Individuare il riquadro Dati CI attivati. Per i dettagli consultare "Dati CI attivati" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

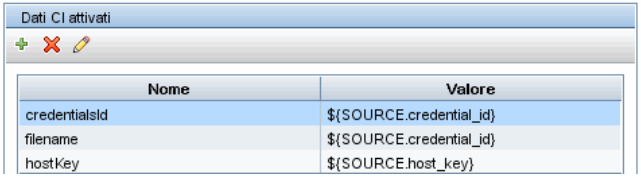
2 Aggiungere variabili alla TQL di input. Per i dettagli, accedere a **Gestione flusso di dati** > **Gestione adattatore**. Individuare il riquadro Dati CI attivati. Per i dettagli consultare il campo Variabili in "Dati CI attivati" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.



Esempio di sostituzione delle variabili con i dati effettivi:

In questo esempio, le variabili sostituiscono i dati del CI IP con i dati effettivi esistenti sulle istanze CI IP reali nel sistema.

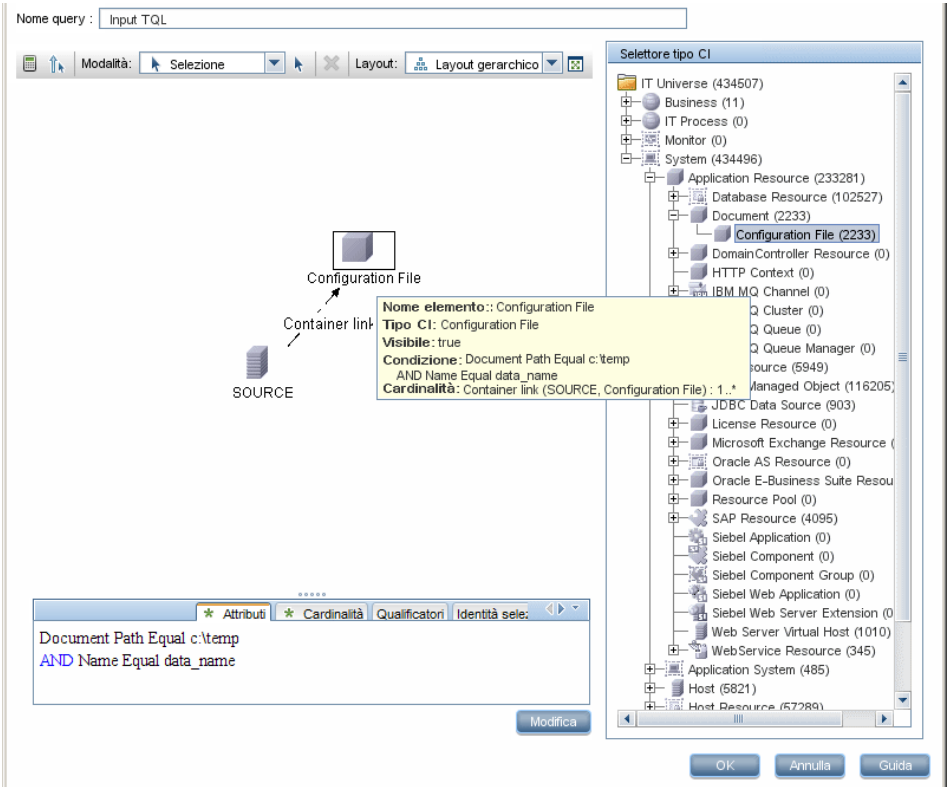
I dati CI attivati per il CI **IP** CI includono una variabile fileName. Questa variabile consente la sostituzione del nodo CONFIGURATION_FILE nella TQL di input con i valori effettivi del file di configurazione ubicato su un host:



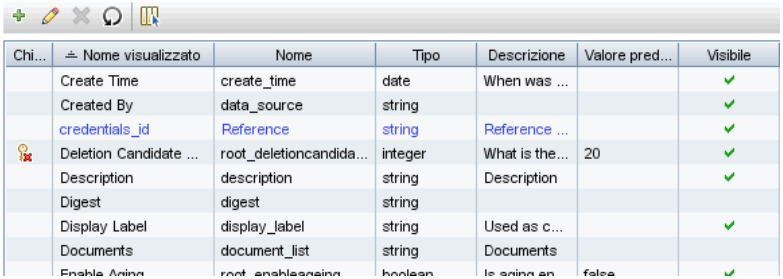
I dati del CI trigger vengono caricati sulla sonda con tutte le variabili sostituite dai valori effettivi. Lo script adattatore include un comando per utilizzare il Framework GFD e recuperare i dati effettivi delle variabili definite:

```
Framework.getTriggerCIData ('ip_address')
```

Le variabili fileName e path utilizzano gli attributi data_name e document_path del nodo del file di configurazione (definito nel TQL di input - vedere l'esempio precedente).



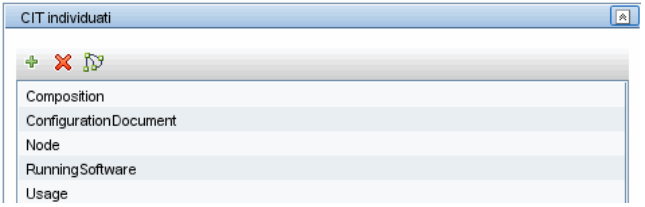
Le variabili Protocol, credentialsId e ip_address utilizzano gli attributi root_class, credentials_id e application_ip:



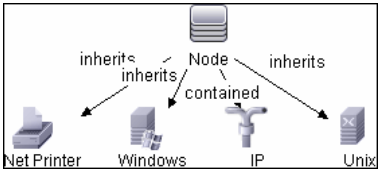
Chi...	Nome visualizzato	Nome	Tipo	Descrizione	Valore pred...	Visibile
	Create Time	create_time	date	When was ...		✔
	Created By	data_source	string			✔
	credentials_id	Reference	string	Reference ...		✔
	Deletion Candidate ...	root_deletioncandida...	integer	What is the...	20	✔
	Description	description	string	Description		✔
	Digest	digest	string			
	Display Label	display_label	string	Used as c...		✔
	Documents	document_list	string	Documents		
	Enable Aging	root_enableageing	boolean	Is aging en...	false	✔

Definire l'output dell'adattatore

L'output dell'adattatore è un elenco dei CI individuati (**Gestione flusso di dati** > **Gestione adattatore** > **Definizione adattatore** > **CIT individuati**) e dei collegamenti tra loro:



È possibile inoltre visualizzare i CI come mappa topologica, ovvero, i componenti e il modo in cui questi sono collegati tra loro (fare clic sul pulsante **Visualizza CIT individuati come mappa**):



I CI individuati vengono restituiti dal codice GFD (ovvero lo script Jython) nel formato ObjectStateHolderVector di UCMDB. Per i dettagli consultare "Generazione di risultati dallo script Jython" a pag. 72.

Esempio di output dell'adattatore:

In questo esempio, vengono definiti quali CI devono far parte dell'output del CI IP.

1. Accedere a **Gestione flusso di dati** > **Gestione adattatore**.
2. Nel riquadro Risorse, selezionare **Rete** > **Adattatori** > **NSLOOKUP_on_Probe**.
3. Nella scheda Definizione adattatore, individuare il riquadro CIT individuati.
4. Vengono elencati i CIT che devono far parte dell'output dell'adattatore. Aggiungere i CIT o rimuoverli dall'elenco. Per i dettagli consultare "Riquadro CIT individuati" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

Sostituire i parametri dell'adattatore

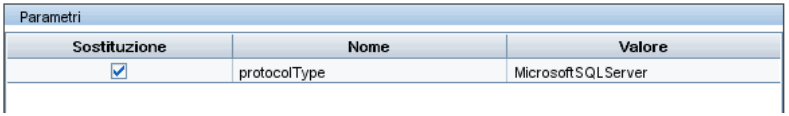
Per configurare un adattatore per più di un processo, è possibile sostituire i parametri dell'adattatore. Ad esempio, l'adattatore SQL_NET_Dis_Connection viene utilizzato da entrambi i processi MSSQL Connection by SQL e Oracle Connection by SQL.

Esempio di sostituzione di un parametro dell'adattatore:

Questo esempio illustra la sostituzione di un parametro dell'adattatore in modo che sia possibile utilizzare un adattatore per individuare entrambi i database Microsoft SQL Server e Oracle.

1. Accedere a **Gestione flusso di dati** > **Gestione adattatore**.
2. Nel riquadro Risorse, selezionare **Database di base** > **Adattatori** > **SQL_NET_Dis_Connection**.
3. Nella scheda Definizione adattatore, individuare il riquadro **Parametri sequenza di individuazione**. Il parametro protocolType ha un valore di **tutti**:

4. Fare clic con il pulsante destro del mouse sull'adattatore **SQL_NET_Dis_Connection_MsSql** e scegliere **Vai a processo di individuazione** > **MSSQL Connection by SQL**.
5. Visualizzare la scheda Proprietà. Individuare il riquadro Parametri:



Il valore tutti viene sovrascritto con il valore MicrosoftSQLServer.

Nota: il processo **Oracle Connection by SQL** include gli stessi parametri ma il valore viene sovrascritto con un valore Oracle.

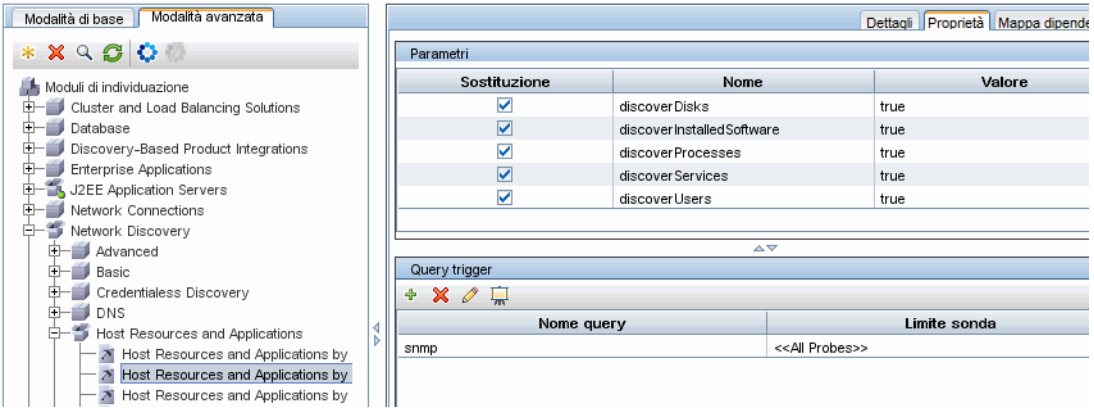
Per i dettagli sull'aggiunta, eliminazione o modifica dei parametri consultare "Riquadro Parametri adattatore" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

GFD inizia a cercare le istanze di Microsoft SQL Server in base a questo parametro.

Passaggio 2: Assegnare un processo all'adattatore

Ciascun adattatore ha uno o più processi associati che definiscono il criterio di esecuzione. I processi consentono la pianificazione dello stesso adattatore in modo differente su insiemi diversi di CI attivati e consentono inoltre la fornitura di parametri diversi per ciascun insieme.

I processi vengono visualizzati nella struttura dei Moduli di individuazione che rappresenta l'entità attivata dall'utente.



TQL trigger

Ciascun processo è associato ai TQL trigger. Questi TQL trigger pubblicano i risultati utilizzati come CI trigger di input per l'adattatore di questo processo.

Una TQL trigger può aggiungere vincoli a una TQL di input. Ad esempio, se i risultati di una TQL trigger sono IP connessi a SNMP, questi risultati possono essere IP connessi a SNMP all'interno dell'intervallo 195.0.0.0-195.0.0.10.

Nota: una TQL trigger deve riferirsi agli stessi oggetti ai quali si riferisce la TQL di input. Ad esempio, se una TQL di input richiede IP con SNMP in esecuzione, non è possibile definire una TQL trigger (per lo stesso processo) per chiedere gli IP connessi a un host, poiché alcuni IP potrebbero non essere connessi a un oggetto SNMP, come richiesto dalla TQL di input.

Pianificazione

Le informazioni di pianificazione per la sonda specificano quando eseguire il codice sui CI trigger. Se la casella di controllo **Richiama immediatamente sui nuovi CI attivati** è selezionata, anche il codice viene eseguito una volta su ciascun CI trigger quando raggiunge la sonda, indipendentemente dalle impostazioni di pianificazione futura.

Per ogni occorrenza pianificata per ciascun processo, la sonda esegue il codice rispetto a tutti i CI trigger accumulati per quel processo. Per i dettagli consultare "Finestra di dialogo Utilità di pianificazione individuazione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

Parametri

Quando si configura un processo, è possibile sostituire i parametri dell'adattatore. Per i dettagli consultare "Sostituire i parametri dell'adattatore" a pag. 48.

Passaggio 3: Creare il codice Jython

HP Universal CMDB utilizza gli script Jython per la scrittura dell'adattatore. Ad esempio, lo script SNMP_Connection.py viene utilizzato dall'adattatore SNMP_NET_Dis_Connection per tentare la connessione ai computer tramite SNMP. Jython è un linguaggio basato su Python e potenziato da Java.

Per i dettagli su come lavorare in Jython, è possibile consultare questi siti Web:

- http://www.jython.org
- http://www.python.org

Per i dettagli consultare "Creare il codice Jython" a pag. 65.

2

Linee guida per la migrazione del contenuto di individuazione

Questo capitolo comprende:

Concetti

Compiti

Riferimenti

Concetti

Linee guida per la migrazione del contenuto di individuazione - Panoramica

In HP Universal CMDB versione 9.0x, il modello di dati si è evoluto significativamente, forzando cambiamenti correlati nel precedente codice contenuto di Discovery and Dependency Mapping (DDM). Di conseguenza, alcuni meccanismi principali del contenuto di DDM sono cambiati. Pertanto, il contenuto sviluppato per UCMDB prima della versione 9.0x deve essere aggiornato per corrispondere al modello di dati 9.0x (BDM: BTO Data Model, Modello di dati BTO). Questa sezione fornisce istruzioni per il processo di adozione del contenuto di DDM e del relativo allineamento con BDM.

Per i dettagli sull'aggiornamento di HP Universal CMDB consultare "Aggiornamento di HP Universal CMDB dalla versione 8.0x alla versione 9.0x" nella *Guida alla distribuzione di HP Universal CMDB* in PDF.

Nuove funzionalità dell'infrastruttura della versione 9.0x

Nota: per i dettagli sull'accesso online alla documentazione di BDM consultare"Accedere alla documentazione online del modello di dati BTO" a pag. 61.

In questa sezione vengono trattati i seguenti argomenti:

➤ "Meccanismo del software" a pag. 56

➤ "Identificazione lato sonda" a pag. 57

➤ "Livello di trasformazione" a pag. 58

Modello di dati BTO (BTO Data Model, BDM)

➤ Per i dettagli sul modello di dati BTO (BDM) consultare il documento Modello di dati concettuale. Questo documento è un mappa dei concetti presenti nel modello nonché l'ambito del modello. Il modello di dati concettuale fornisce un punto di partenza per la comprensione della semantica del dominio sottoposto a modello.

➤ Per i dettagli sulle classi di BDM consultare il documento HP Software BTO Data Model Reference. In questo documento vengono trattate tutte le classi di BDM, inclusi la descrizione e l'attributo delle classi, il qualificatore e le informazioni sulla gerarchia.

Differenze tra il modello di classe di UCMDB 8.0x e il modello di dati di UCMDB 9.0x

I cambiamenti apportati tra il modello di classe di UCMDB versione 8.0x e BDM vengono scaricati nella sonda nel file di configurazione dell'individuazione:
C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles\flat-class-model-changes.xml.

bdm_changes.xml. Questo file XML contiene le informazioni relative ai cambiamenti apportati ai nomi delle classi, nomi degli attributi, classi rimosse, attributi, qualificatori ecc.

➤ Per i dettagli sulla mappatura tra il modello di classe di UCMDB versione 8.0x e BDM, consultare il documento Mappatura di UCMDB 9.0x (Modello di dati BTO) sul modello di classe di UCMDB 8.0x.

➤ Per i dettagli sui cambiamenti apportati al modello di classe tra la versione 8.0x e 9.0x, consultare il documento Report dei cambiamenti del modello di dati di UCMDB.

Nuovo meccanismo di identificazione CIT

Nelle versioni di UCMDB precedenti alla versione 9.0x, i CI vengono identificati utilizzando gli attributi principali. In UCMDB versione 9.0x, questo concetto è stato generalizzato e l'identificazione viene eseguita in un componente server denominato Motore di riconciliazione. Il Motore di riconciliazione è in grado di identificare i CI tramite regole logiche denominate regole DDA (Data Definition Algorithm, Algoritmo di definizione dati).

Questo nuovo meccanismo è soprattutto utile per i CI in cui la topologia correlata è importante per la relativa identificazione (ad esempio, il nodo CIT, host nelle versioni precedenti, è identificato dal relativo nome e dalla topologia correlata, come ad esempio l'indirizzo IP e i CIT di interfaccia). Alcuni CIT sono ancora identificati dagli attributi principali; per questi CIT non è definita una regola DDA.

Per i dettagli sul Motore di riconciliazione consultare "Riconciliazione - Panoramica" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

Meccanismo del software

Il CI dell'**Elemento software** versione 8.0x è denominato **Software** in BDM versione 9.0x. Questo CIT viene identificato nella versione 9.0x da una regola DDA e non dagli attributi principali.

Supponiamo che sia stato aggiunto un CIT personalizzato derivato dal CIT del **Software**. Nelle versioni precedenti, questo CIT personalizzato era identificato dai relativi attributi principali. Tuttavia, nella versione 9.0x questo CIT viene identificato da una regola DDA ereditata, ignorando pertanto gli attributi principali.

Se si aggiunge un CIT derivato, tenere presente quanto segue:

- Per identificare il nuovo CIT tramite la stessa regola DDA come tutti i CIT software, è necessario mantenere la configurazione corrente.
- Per identificare il nuovo CIT tramite gli attributi principali, è necessario creare una regola DDA definendo l'identificazione tramite attributi principali. Di seguito è indicato un esempio per questa regola DDA, definita per il CIT **object**:

```
<identification-config type="object">
    <identification-criteria>
        <identification-criterion targetType="root">
            <key-attributes-condition/>
        </identification-criterion>
</identification-criteria>
</identification-config>
```

Identificazione lato sonda

DDM_ID_ATTRIBUTE. La sonda del flusso di dati versione 9.0x identifica i CI sono tramite i relativi attributi principali (ovvero **ID_ATTRIBUTE**). Se un CIT include una regola DDA (ovvero una regola di riconciliazione), il CIT potrebbe non includere un attributo principale. In tal caso, gli attributi principali del CIT sono contrassegnati da un qualificatore **DDM_ID_ATTRIBUTE**. Pertanto, ai fini dell'identificazione di un CI, la sonda prende in considerazione tutti i qualificatori **DDM_ID_ATTRIBUTE** e **ID_ATTRIBUTE**.

DDM_REQUIRED_TOPOLOGY. Una regola DDA per un CIT specifico può dipendere da CI diversi segnalati nello stesso blocco, insieme al CI esaminato. Ad esempio, l'identificazione del CIT del **J2EE Domain** viene eseguita esclusivamente tramite l'attributo del nome del dominio ma anche tramite il CIT del **J2EE Application Server** a esso collegato mediante un collegamento membro.

Per accertarsi che tutti i CI richiesti vengano segnalati con il CI esaminato, è necessario contrassegnare ciascuno dei CI esaminati con il qualificatore **DDM_REQUIRED_TOPOLOGY** contenente una voce dati che specifichi il tipo di collegamento richiesto. Ad esempio, nell'esempio suddetto, il CIT del **J2EE Domain** è contrassegnato con il qualificatore **DDM_REQUIRED_TOPOLOGY** e con un elemento dati del collegamento **member** in modo che, quando l'Individuazione segnala un dominio J2EE, vengono segnalati anche i server.

Per i dettagli sui qualificatori consultare "Pagina Qualificatori" nella *Guida alla modellazione di HP Universal CMDB*.

Livello di trasformazione

Per garantire la compatibilità con le versioni precedenti, è stato introdotto un nuovo meccanismo di trasformazione nella versione 9.0x sulla sonda. Il nuovo meccanismo è in grado di convertire le topologie della versione 8.0x nelle topologie 9.0x in fase di runtime. Ciò consente alla sonda di proseguire l'esecuzione dei compiti, come ad esempio gli script Jython, che segnalano le topologie compatibili con la versione 8.0x.

Il nuovo meccanismo di trasformazione utilizza i dati contenuti nel file **bdm_changes.xml** ed esegue i cambiamenti richiesti (cambiamenti del nome classe e degli attributi, rimozione di attributi, cambiamenti della gerarchia e così via) per rendere le topologie 8.0x compatibili con il BDM. Attualmente (e indipendentemente dalle topologie segnalate dai compiti eseguiti dalla sonda), il server UCMDB riceve topologie compatibili con BDM.

Utilità di migrazione pacchetto

L'installazione di UCMDB 9.0x include un'utilità di migrazione pacchetto esterna che consente agli sviluppatori di contenuti di convertire un pacchetto di contenuti dal modello di classe 8.0x al modello di dati 9.0x. L'utilità di migrazione pacchetto converte le risorse del pacchetto, sottosistema per sottosistema, in modo che siano compatibili con il nuovo modello di classe. Le definizioni CIT, le query, i processi, gli adattatori e i moduli vengono trasformati in base ai dati contenuti nel file **bdm_changes.xml**. Di conseguenza, possono essere distribuiti e utilizzati da un server UCMDB 9.0x.

Per i dettagli consultare "Aggiornamento dei pacchetti dalla versione 8.04 alla 9.02" nella *Guida alla distribuzione di HP Universal CMDB* in PDF.

Limitazioni dell'utilità di migrazione pacchetto

- Gli script Jython non vengono aggiornati mediante l'utilità di migrazione pacchetto. Per il supporto degli script progettati per corrispondere al modello di classe di UCMDB versione 8.0x , è stato introdotto un nuovo modulo di **livello di trasformazione** in UCMDB 9.0x. Per i dettagli consultare "Livello di trasformazione" a pag. 58.
- Gli adattatori di individuazione di tipo Integrazione non vengono aggiornati mediante l'utilità di migrazione pacchetto e, pertanto, devono essere aggiornati manualmente.
- Il processo di individuazione Topologia livello 2 (e le relative risorse quali l'adattatore di individuazione, TQL e così via) è stato significativamente modificato ed è stato rimosso dall'utilità di migrazione pacchetto invece di essere aggiornato.

Linee guida per lo sviluppo di script di modelli di dati incrociati

Le linee guida seguenti sono applicabili a entrambe le versioni 8.0x e 9.0x.

Libreria API script di individuazione

La libreria API di individuazione è completamente compatibile con le versioni precedenti, pertanto, sono supportate tutte le API e le librerie della versione 8.0x. Per i dettagli consultare "Librerie e utilità Jython" a pag. 114.

L'API 9.0x comprende più elementi e metodi. Ad esempio, uno script Jython segnala ora un codice di errore (numero intero) invece di un messaggio di errore di stringa, abilitando in tal modo i messaggi di errore di individuazione localizzati. Per i dettagli consultare "Convenzioni di scrittura errori" a pag. 121.

Suggerimenti di implementazione

- Utilizzare il modulo di **modellazione** per creare un CIT **Software** o un discendente per il quale è presente il metodo pertinente.
- Utilizzare **HostBuilder** per la creazione di CIT di tipo **Nodo**.
- Utilizzare **modeling.createOshByCmdbIdString** per ripristinare l'OSH tramite il relativo ID.
- Utilizzare l'istanza **ShellUtils** del modulo **shellutils** per tutte le connessioni basate su shell.
- Utilizzare il meccanismo predefinito per recuperare la versione di UCMDB: logger.Version().getVersion(framework). Ad esempio, se viene aggiunto un altro attributo application_ip solo per UCMDB versione 9.0x o successive:

```
versionAsDouble = logger.Version().getVersion(Framework)
if versionAsDouble >= 9:
    appServerOSH.setAttribute('application_ip', ip)
```

- Utilizzare **wmiutils** per creare un'individuazione basata su WMI.
- Utilizzare **snmputils** per creare un'individuazione basata su SNMP.

Compiti

Accedere alla documentazione online del modello di dati BTO

Per accedere alla documentazione BDM:

1. Accedere a HP Universal CMDB.
2. Fare clic su **Guida** > **Guida di UCMDB**.
3. Nella pagina iniziale, fare clic sul collegamento **Modellazione** in **Applicazioni** per accedere al portale di **modellazione**.
4. Fare clic sulla scheda **Modello di dati**.

Riferimenti

Risoluzione dei problemi e limitazioni

- Il valore **ip_address** non viene passato per impostazione predefinita alla sequenza. Deve essere aggiunto esplicitamente alla sequenza come Dati CI trigger.
- Se uno script Jython non predefinito richiede una risorsa o un contenitore esterno nel classpath, deve essere ubicato nel pacchetto pertinente in una sottocartella denominata **discoveryResources**.
- Quando si lavora con attributi di tipo **List** come ad esempio **StringVector** e **IntegerVector** (ereditati da **BaseVector**), non è possibile utilizzare entrambe le operazioni **aggiungi elemento** e **rimuovi elemento** sullo stesso oggetto dell'elenco.

3

Sviluppo degli adattatori Jython

Questo capitolo comprende:

Concetti

Compiti

Riferimenti

Concetti

Riferimento API di Gestione flusso di dati di HP

Per la documentazione completa sulle API disponibili consultare *Riferimento API di Gestione flusso di dati di HP Universal CMDB*. Questi file si trovano nella cartella seguente:

C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\doc_lib\DevRef_guide\DDM_JavaDoc\index.html

Compiti

Creare il codice Jython

HP Universal CMDB utilizza gli script Jython per la scrittura dell'adattatore. Ad esempio, lo script SNMP_Connection.py viene utilizzato dall'adattatore SNMP_NET_Dis_Connection per tentare la connessione ai computer tramite SNMP. Jython è un linguaggio basato su Python e potenziato da Java.

Per i dettagli su come lavorare in Jython, è possibile consultare questi siti Web:

- http://www.jython.org
- http://www.python.org

La sezione seguente descrive la scrittura effettiva del codice Jython all'interno del Framework GFD. Questa sezione tratta nello specifico i punti di contatto tra lo script Jython e il Framework richiamato e descrive inoltre le librerie e le utilità Jython da utilizzare ogni volta che è possibile.

Nota:

- Gli script scritti per GFD devono essere compatibili con Jython versione 2.1.
- Per la documentazione completa sulle API disponibili consultare *Riferimento API di Gestione flusso di dati di HP Universal CMDB*.

In questa sezione vengono trattati i seguenti argomenti:

➤ "Generazione di risultati dallo script Jython" a pag. 72

➤ "Istanza Framework" a pag. 74

➤ "Individuazione delle credenziali corrette (per gli adattatori di connessione)" a pag. 78

➤ "Gestione delle eccezioni da Java" a pag. 79

Utilizzare i file JAR Java esterni all'interno di Jython

Quando si sviluppano nuovi script Jython, sono talvolta necessarie librerie Java esterne (file JAR) o file eseguibili di terze parti come archivi di utilità Java, archivi di connessione quali i file Java dei driver JDBC o file eseguibili (ad esempio, **nmap.exe** è utilizzato per l'individuazione senza credenziali).

Queste risorse devono essere raggruppate nel pacchetto nella cartella Risorse esterne. Qualsiasi risorsa presente in questa cartella viene inviata automaticamente a una sonda che si connette al server HP Universal CMDB.

Inoltre, quando l'individuazione viene avviata, qualsiasi risorsa del file JAR viene caricata nel classpath di Jython, rendendo tutte le classi al suo interno disponibili all'importazione e all'utilizzo.

Esecuzione del codice

Dopo l'attivazione di un processo, viene scaricato sulla sonda un compito con tutte le informazioni richieste.

La sonda avvia l'esecuzione del codice GFD utilizzando le informazioni specificate nel compito.

Il flusso dei codici Jython avvia l'esecuzione da una voce principale all'interno dello script, esegue il codice per individuare i CI e fornisce i risultati di un vettore dei CI individuati.

Modifica degli script predefiniti

Quando si eseguono modifiche di uno script predefinito, apportare soltanto cambiamenti minimi allo script e posizionare i metodi necessari in uno script esterno. È possibile rilevare i cambiamenti in modo più efficiente e, quando si passa a una versione più aggiornata di HP Universal CMDB, il codice non viene sovrascritto.

Ad esempio, la seguente riga singola di codice in un script predefinito chiama un metodo che calcola il nome di un server Web in un modo specifico dell'applicazione:

```
	serverName = iplanet_cspecific.PlugInProcessing(serverName, transportHN,
mam_utils)
```

La logica più complessa che decide come calcolare questo nome è contenuta in uno script esterno:

```
# implement customer specific processing for 'servername' attribute of httpplugin

#

def PlugInProcessing(servername, transportHN, mam_utils_handle):

	# support application-specific HTTP plug-in naming

	if servername == "appsrv_instance":

		# servername is supposed to match up with the j2ee server name,
however some groups do strange things with their

		# iPlanet plug-in files. this is the best work-around we could find. this join
can't be done with IP address:port

		# because multiple apps on a web server share the same IP:port for
multiple websphere applications

		logger.debug('httpcontext_webapplicationserver attribute has been
changed from [' + servername + '] to [' + transportHN[:5] + '] to facilitate websphere
enrichment')

		servername = transportHN[:5]

	return servername
```

Salvare lo script esterno nella cartella Risorse esterne. Per i dettagli consultare "Riquadro Risorse" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*. Se si aggiunge questo script a un pacchetto, è possibile utilizzarlo anche per altri processi. Per i dettagli sull'utilizzo di Gestione pacchetti consultare "Gestione pacchetti" nella *Guida all'amministrazione di HP Universal CMDB*.

Durante l'aggiornamento, il cambiamento apportato alla riga singola del codice viene sovrascritto dalla nuova versione dello script predefinito, pertanto sarà necessario sostituire la riga. Tuttavia, lo script esterno non viene sovrascritto.

Struttura del file Jython

Il file Jython è composto da tre parti in una sequenza specifica:

1. Imports
2. Main Function - DiscoveryMain
3. Functions definitions (facoltativo)

Di seguito viene riportato un esempio di script Jython:

```
#   imports section
from appilog.common.system.types import ObjectStateHolder
from appilog.common.system.types.vectors import ObjectStateHolderVector

# Function definition
def foo:
    # do something

#   Main Function
def DiscoveryMain(Framework):
    OSHVResult = ObjectStateHolderVector()

    ## Write implementation to return new result CIs here...

    return OSHVResult
```

Imports

Le classi Jython vengono distribuite negli spazi dei nomi gerarchici. Nella versione 7.0 o successive, diversamente dalle versioni precedenti, non ci sono importazioni implicite, pertanto ogni classe utilizzata deve essere importata esplicitamente. (Questo cambiamento è stato apportato per motivi di prestazioni e per semplificare la comprensione dello script Jython non nascondendo i dettagli necessari.)

➤ Per importare uno script Jython:

```
import logger
```

➤ Per importare una classe Java:

```
from appilog.collectors.clients import ClientsConsts
```

Main Function – DiscoveryMain

Ciascun file script eseguibile Jython contiene una funzione principale: DiscoveryMain.

La funzione DiscoveryMain è la funzione principale all'interno dello script; è la prima funzione eseguita. La funzione principale può chiamare altre funzioni definite negli script:

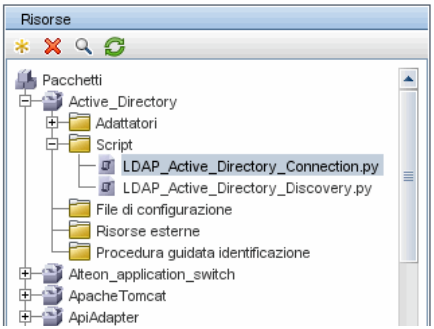
```
def DiscoveryMain(Framework):
```

L'argomento Framework deve essere specificato nella definizione delle funzione principale. Questo argomento viene utilizzato dalla funzione principale per recuperare le informazioni richieste per l'esecuzione degli script (come ad esempio le informazioni sul CI trigger e sui parametri) e può anche essere utilizzato per segnalare eventuali errori durante l'esecuzione dello script.

È possibile creare uno script Jython senza un metodo principale. Tali script vengono utilizzati come script di libreria chiamati da altri script.

Functions Definition

Ciascuno script può contenere funzioni aggiuntive chiamate dal codice principale. Ciascuna di queste funzioni può chiamare un'altra funzione esistente nello script corrente o in un altro script (utilizzare l'istruzione import). Tenere presente che per utilizzare un altro script è necessario aggiungerlo alla sezione Scripts del pacchetto:



Esempio di una funzione che chiama un'altra funzione:

Nell'esempio seguente, il codice principale chiama il metodo doQueryOSUsers(..) che, a sua volta, chiama un metodo interno doOSUserOSH(..):

```
def doOSUserOSH(name):
    sw_obj = ObjectStateHolder('winosuser')

    sw_obj.setAttribute('data_name', name)
    # return the object
    return sw_obj

def doQueryOSUsers(client, OSHVResult):
    _hostObj = modeling.createHostOSH(client.getIpAddress())
    data_name_mib = '1.3.6.1.4.1.77.1.2.25.1.1,1.3.6.1.4.1.77.1.2.25.1.2,string'
    resultSet = client.executeQuery(data_name_mib)
    while resultSet.next():
        UserName = resultSet.getString(2)
        ########## send object ###############
        OSUserOSH = doOSUserOSH(UserName)
        OSUserOSH.setContainer(_hostObj)
        OSHVResult.add(OSUserOSH)

def DiscoveryMain(Framework):
    OSHVResult = ObjectStateHolderVector()
    try:
        client =
Framework.getClientFactory(ClientsConsts.SNMP_PROTOCOL_NAME).createClie
nt()
    except:
        Framework.reportError('Connection failed')
    else:
        doQueryOSUsers(client, OSHVResult)
        client.close()
    return OSHVResult
```

Se lo script è una libreria globale relativa a molti adattatori, è possibile aggiungerlo all'elenco degli script nel file di configurazione jythonGlobalLibs.xml invece di aggiungerlo a ciascun adattatore (**Gestione adattatore** > **Riquadro Risorse** > **AutoDiscoveryContent** > **File di configurazione**).

Generazione di risultati dallo script Jython

Ciascuno script Jython viene eseguito da un CI trigger specifico e termina con i risultati restituiti dal valore di ritorno della funzione DiscoveryMain.

Il risultato dello script è di fatto un gruppo di CI e collegamenti che devono essere inseriti o aggiornati nel CMDB. Lo script restituisce questo gruppo di CI e collegamenti in formato ObjectStateHolderVector.

La classe ObjectStateHolder è un modo di rappresentare un oggetto o un collegamento definito nel CMDB. L'oggetto ObjectStateHolder contiene il nome CIT e un elenco di attributi e relativi valori. ObjectStateHolderVector è un vettore di istanze ObjectStateHolder.

Sintassi ObjectStateHolder

Questa sezione spiega come creare i risultati di GFD in un modello UCMDB.

Esempio di impostazione attributi sui CI:

La classe ObjectStateHolder descrive il grafico dei risultati di GFD. Ciascun CI e ciascun collegamento (relazione) è ubicato all'interno di un'istanza della classe ObjectStateHolder come indicato nel seguente esempio di codice Jython:

```
# siebel application server
1 appServerOSH = ObjectStateHolder('siebelappserver' )
2 appServerOSH.setStringAttribute('data_name', sblsvrName)
3 appServerOSH.setStringAttribute ('application_ip', ip)
4 appServerOSH.setContainer(appServerHostOSH)
```

- La riga 1 crea un CI di tipo **siebelappserver**.
- La riga 2 crea un attributo denominato **data_name** con un valore di **sblsvrName** ovvero una variabile Jython impostata con il valore individuato per il nome server.
- La riga 3 imposta un attributo non chiave aggiornato nel CMDB.
- La riga 4 è la creazione del Containment (il risultato è un grafico). Essa specifica che questo server applicazioni è contenuto all'interno di un host (un'altra classe ObjectStateHolder nell'ambito).

Nota: ciascun CI segnalato dallo script Jython deve includere i valori per tutti gli attributi chiave del tipo CI.

Esempio di relazioni (collegamenti):

L'esempio di collegamento seguente spiega come viene rappresentato il grafico:

```
1 linkOSH = ObjectStateHolder('route')
2 linkOSH.setAttribute('link_end1', gatewayOSH)
3 linkOSH.setAttribute('link_end2', appServerOSH)
```

- La riga 1 crea il collegamento (che è anche della classe ObjectStateHolder. L'unica differenza è che route è un tipo CI di collegamento).
- Le righe 2 e 3 specificano i nodi all'estremità di ciascun collegamento. Ciò viene eseguito utilizzando gli attributi **end1** e **end2** del collegamento da specificare (poiché si tratta degli attributi chiave minimi di ciascun collegamento). I valori dell'attributo sono le istanze ObjectStateHolder. Per i dettagli su End 1 ed End 2 consultare "Collegamento" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

Attenzione: un collegamento è direzionale. È necessario verificare che i nodi End 1 ed End 2 corrispondano ai CIT su ciascuna estremità. Se i nodi non sono validi, l'oggetto risultante non viene convalidato né segnalato correttamente. Per i dettagli consultare "Relazioni del tipo di CI" nella *Guida alla modellazione di HP Universal CMDB*.

Esempio di vettore (raccolta CI):

Dopo la creazione degli oggetti con attributi e dei collegamenti con gli oggetti alla rispettiva fine, è necessario raggrupparli. Eseguire questa operazione aggiungendoli a un'istanza ObjectStateHolderVector come descritto di seguito:

```
oshvMyResult = ObjectStateHolderVector()
oshvMyResult.add(appServerOSH)
oshvMyResult.add(linkOSH)
```

Per i dettagli sulla segnalazione di questo risultato composito al Framework in modo che possa essere inviato al server CMDB consultare il metodo sendObjects.

Una volta assemblato il grafico dei risultati in un'istanza ObjectStateHolderVector, restituirlo al Framework GFD da inserire nel CMDB. Questa operazione viene eseguita restituendo l'istanza ObjectStateHolderVector come risultato della funzione DiscoveryMain().

Nota: per i dettagli sulla creazione di **OSH** per i CIT comuni consultare **modeling.py** in "Librerie e utilità Jython" a pag. 114.

Istanza Framework

L'istanza Framework è l'unico argomento fornito nella funzione principale dello script Jython. Questa è un'interfaccia che può essere utilizzata per recuperare le informazioni necessarie per eseguire lo script (ad esempio, le informazioni sui CI trigger e sui parametri dell'adattatore) e viene utilizzata inoltre per segnalare gli errori durante l'esecuzione dello script. Per i dettagli consultare "Riferimento API di Gestione flusso di dati di HP" a pag. 64.

Questa sezione descrive gli utilizzi più importanti del Framework:

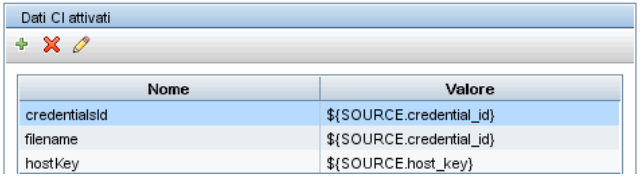
- "Framework.getTriggerCIData(String attributeName)" a pag. 74
- "Framework.createClient(credentialsId, props)" a pag. 75
- "Framework.getParameter (String parameterName)" a pag. 76
- "Framework.reportError(String message) e Framework.reportWarning(String message)" a pag. 77

Framework.getTriggerCIData(String attributeName)

Questa API fornisce il passaggio intermedio tra i dati del CI trigger definiti nell'adattatore e nello script.

Esempio di recupero delle informazioni sulle credenziali:

Richiedere le seguenti informazioni sui dati del CI trigger:



Per recuperare le informazioni sulle credenziali dal compito, utilizzare questa API:

```
credId = Framework.getTriggerCIData('credentialsId')
```

Framework.createClient(credentialsId, props)

È possibile eseguire una connessione a un computer remoto utilizzando un oggetto client ed eseguendo i comandi su questo client. Per creare un client, recuperare la classe ClientFactory. Il metodo getClientFactory() riceve il tipo di protocollo client richiesto. Le costanti del protocollo vengono definite nella classe ClientsConsts. Per i dettagli sulle credenziali e sui protocolli supportati consultare "Riferimenti sulle credenziali del dominio" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

Esempio di creazione di un'istanza client per l'ID delle credenziali:

Per creare un'istanza Client per l'ID delle credenziali:

```
properties = Properties()
codePage = Framework.getCodePage()
properties.put( BaseAgent.ENCODING, codePage)
client = Framework.createClient(credentailsID ,properties)
```

È ora possibile utilizzare l'istanza Client per eseguire la connessione al computer o all'applicazione pertinente.

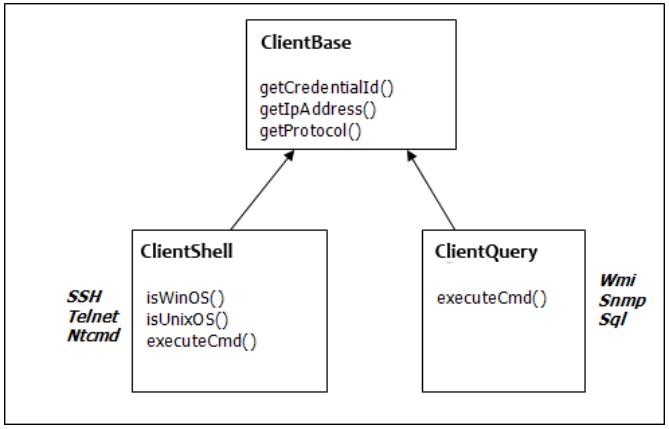
Esempio di creazione di un client WMI e di esecuzione di una query WMI:

Per creare un client WMI ed eseguire una query WMI utilizzando il client:

```
wmiClient = Framework.createClient(credential)
resultSet = wmiClient. executeQuery("SELECT TotalPhysicalMemory
                                    FROMWin32_LogicalMemoryConfiguration")
```

Nota: per far funzionare l'API createClient(), aggiungere il parametro seguente ai parametri dei dati del CI trigger: **credentialsId = ${SOURCE.credentials_id}** nel riquadro Dati CI attivati. Oppure è possibile aggiungere manualmente l'ID delle credenziali quando viene chiamata la funzione:
wmiClient = clientFactory().createClient(credentials_id).

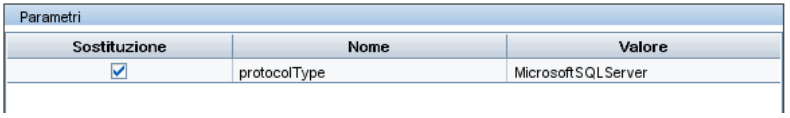
Il diagramma seguente illustra la gerarchia dei client con le rispettive API comunemente supportate:



Per i dettagli sui client e sulle rispettive API supportate consultare BaseClient, ShellClient e QueryClient in *Riferimento API di Gestione flusso di dati di HP Universal CMDB*.

Framework.getParameter (String parameterName)

Oltre a recuperare le informazioni sul CI trigger, è spesso necessario recuperare un valore del parametro dell'adattatore. Ad esempio:



Esempio di recupero del valore del parametro protocolType:

Per recuperare il valore del parametro protocolType dallo script Jython, utilizzare la seguente API:

```
protocolType = Framework.getParameterValue('protocolType')
```

Framework.reportError(String message) e Framework.reportWarning(String message)

Durante l'esecuzione di uno script possono verificarsi alcuni errori (ad esempio, errori di connessione, problemi hardware, timeout). Quando tali errori vengono rilevati, il Framework può segnalare il problema. Il messaggio segnalato raggiunge il server e viene visualizzato per l'utente.

Esempio di segnalazione di un errore e messaggio:

L'esempio seguente illustra l'utilizzo dell'API reportError(<Error Msg>):

```
try:
        client =
Framework.getClientFactory(ClientsConsts.SNMP_PROTOCOL_NAME)
        createClient()
except:
        strException = str(sys.exc_info()[1]).strip()
        Framework. reportError ('Connection failed: %s' % strException)
```

È possibile utilizzare una delle API—Framework.reportError(String message) o Framework.reportWarning(String message)— per segnalare un problema. La differenza tra le due API è che, quando viene segnalato un errore, la sonda salva un file di registro di comunicazione con i parametri dell'intera sessione nel file system. In questo modo, è possibile monitorare le sessione e comprendere meglio l'errore.

Per i dettagli sui messaggi di errore consultare "Messaggi di errore" a pag. 119.

Individuazione delle credenziali corrette (per gli adattatori di connessione)

Un adattatore che tenta di connettersi a un sistema remoto deve tentare tutte le possibili credenziali. Uno dei parametri necessari quando si crea un client (tramite ClientFactory) è l'ID delle credenziali. Lo script di connessione ottiene l'accesso ai set di credenziali possibili e li prova uno alla volta utilizzando il metodo clientFactory.getAvailableProtocols(). Quando un set di credenziali è corretto, l'adattatore segnala un oggetto di connessione CI sull'host di questo CI trigger (con l'ID delle credenziali corrispondente all'IP) al CMDB. Gli adattatori successivi possono utilizzare questo CI dell'oggetto di connessione per connettersi al set di credenziali (ovvero, gli adattatori non devono tentare nuovamente tutte le possibili credenziali).

L'esempio seguente mostra come ottenere tutte le voci del protocollo SNMP. Tenere presente che qui, l'IP viene ottenuto dai dati del CI trigger (# Get the Trigger CI data values).

Lo script di connessione richiede tutte le possibili credenziali del protocollo (# Go over all the protocol credentials) e le prova a rotazione fino a trovare quella corretta (resultVector). Per i dettagli consultare il **paradigma di connessione a due fasi** in "Separazione di adattatori" a pag. 37.

```
import logger
from appilog.collectors.clients import ClientsConsts
from appilog.common.system.types.vectors import ObjectStateHolderVector

        def mainFunction(Framework):
resultVector = ObjectStateHolderVector()

        # Get the Trigger CI data values
        ip_address = Framework.getDestinationAttribute('ip_address')
        ip_domain  = Framework.getDestinationAttribute('ip_domain')

        # Create the client factory for SNMP
        clientFactory = framework.getClientFactory(ClientsConsts.SNMP_PROTOCOL_NAME)
        protocols = clientFactory.getAvailableProtocols(ip_address, ip_domain)
```

```
connected = 0
# Go over all the protocol credentials
for credentials_id in protocols:
    client = None
    try:
        # try to connect to the snmp agent
        client = clientFactory.createClient(credentials_id)

        // Query the agent
        ….

        # connection succeed
        connected = 1
    except:
        if client != None:
            client.close()
if (not connected):
    logger.debug('Failed to connect using all credentials')
else:
    // return the results as OSHV
    return resultVector
```

Gestione delle eccezioni da Java

Alcune classi Java restituiscono un'eccezione quando viene rilevato un errore. Si consiglia di gestire l'eccezione, altrimenti potrebbe interrompere immediatamente l'adattatore.

Per le eccezioni conosciute, nella maggior parte dei casi è necessario stamparne la traccia dello stack nel registro e inviare un messaggio adeguato all'interfaccia utente, ad esempio:

```
try:
        client = Framework.getClientFactory().createClient()
except Exception, msg:
        Framework.reportError('Connection failed')
        logger.debugException('Exception while connecting: %s' % (msg))
        return
```

Se l'eccezione non è irreversibile e lo script può continuare, è necessario omettere la chiamata per il metodo reportError() e consentire la continuazione dello script.

Supportare la localizzazione negli adattatori Jython

La funzione delle impostazioni internazionali multilingue consente a GFD di funzionare con diverse lingue di sistema operativo (OS) e di abilitare le personalizzazioni durante il runtime.

Prima di Content Pack 3.00, GFD utilizzava una codifica specificata statisticamente per trattare l'output da tutte le destinazioni di rete. Tuttavia, questo approccio non è adatto a una rete IT multilingue: per individuare gli host con lingue di sistema operativo diverse, gli amministratori della sonda devono rieseguire manualmente i processi di GFD più volte con parametri di processo diversi ogni volta. Questa procedura ha generato un grave sovraccarico sulla rete e, problema ancora più grave, ha impedito diverse funzioni chiave di GFD, come ad esempio il richiamo del processo intermedio su un CI trigger o l'aggiornamento automatico dei dati in UCMDB da parte della Gestione pianificazione.

Per impostazione predefinita, sono supportate le seguenti lingue internazionali: giapponese, russo e tedesco. L'impostazione internazionale predefinita è inglese.

In questa sezione vengono trattati i seguenti argomenti:

Aggiungere il supporto per una nuova lingua

Questo compito descrive come aggiungere il supporto per una nuova lingua

Questo compito include i passaggi seguenti:

➤ "Aggiungere un pacchetto di risorse (file *.properties)" a pag. 81

➤ "Dichiarare e registrare l'oggetto Language" a pag. 82

1 Aggiungere un pacchetto di risorse (file *.properties)

Aggiungere un pacchetto di risorse in base al processo da eseguire. La tabella seguente elenca i processi di GFD e il pacchetto di risorse utilizzato da ciascun processo:

Processo	Nome base del pacchetto di risorse
File Monitor by Shell	langFileMonitoring
Host Resources and Applications by Shell	langHost_Resources_By_TTY, langTCP
Hosts by Shell using NSLOOKUP in DNS Server	langNetwork
Host Connection by Shell	langNetwork
Collect Network Data by Shell or SNMP	langTCP
Host Resources and Applications by SNMP	langTCP
Microsoft Exchange Connection by NTCMD, Microsoft Exchange Topology by NTCMD	msExchange
MS Cluster by NTCMD	langMsCluster

Per i dettagli sui pacchetti consultare "Utilizzare i pacchetti di risorse" a pag. 87.

2 Dichiarare e registrare l'oggetto Language

Per definire una nuova lingua, aggiungere le seguenti due righe di codice allo script **shellutils.py** che attualmente contiene l'elenco di tutte le lingue supportate. Lo script è incluso nel pacchetto AutoDiscoveryContent. Per visualizzare lo script, accedere alla finestra Gestione adattatore. Per i dettagli consultare "Finestra Gestione adattatore" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

a Dichiarare la lingua come segue:

```
LANG_RUSSIAN = Language(LOCALE_RUSSIAN, 'rus', ('Cp866', 'Cp1251'),
(1049,), 866)
```

Per i dettagli sulla lingua della classe consultare "Riferimento API" a pag. 88. Per i dettagli sull'oggetto Class Locale consultare http://java.sun.com/j2se/1.5.0/docs/api/java/util/Locale.html. È possibile utilizzare un'impostazione internazionale esistente o definirne una nuova.

b Registrare la lingua aggiungendola alla raccolta seguente:

```
LANGUAGES = (LANG_ENGLISH, LANG_GERMAN, LANG_SPANISH,
LANG_RUSSIAN, LANG_JAPANESE)
```

Cambiare la lingua predefinita

Se non è possibile determinare la lingua del sistema operativo, viene utilizzata quella predefinita. La lingua predefinita è specificata nel file **shellutils.py**.

```
#default language for fallback
DEFAULT_LANGUAGE = LANG_ENGLISH
```

Per cambiare la lingua predefinita, inizializzare la variabile DEFAULT_LANGUAGE con una lingua diversa. Per i dettagli consultare "Aggiungere il supporto per una nuova lingua" a pag. 81.

Determinare il set di caratteri per la codifica

Il set di caratteri adatto per l'output comando di decodifica viene determinato durante il runtime. La soluzione multilingue è basata sui seguenti fatti e presupposti:

1. È possibile determinare la lingua del sistema operativo in un modo indipendente dalle impostazioni internazionali, ad esempio, eseguendo il comando **chcp** in Windows o il comando **locale** in Linux.
2. La codifica della lingua della relazione è nota e può essere definita statisticamente. Ad esempio, la lingua russa ha due delle codifiche più conosciute: Cp866 e Windows-1251.
3. È preferibile un set di caratteri per ciascuna lingua, ad esempio, il set di caratteri predefiniti per la lingua russa è Cp866. Ciò significa che la maggior parte dei comandi produce output in questa codifica.
4. La codifica in cui viene fornito l'output comando successivo è imprevedibile, tuttavia è una delle codifiche possibili per una lingua specifica. Ad esempio, quando si lavora con un computer Windows con un'impostazione internazionale russa, il sistema fornisce l'output comando **ver** in Cp866, ma il comando **ipconfig** viene fornito in Windows-1251.

5 Un comando noto produce parole chiave note nel relativo output. Ad esempio, il comando **ipconfig** contiene la forma tradotta della stringa **IP-Address**. Pertanto, l'output comando **ipconfig** contiene **IP-Address** per il sistema operativo inglese, IP-Адрес per il sistema operativo russo, **IP-Adresse** per il sistema operativo tedesco e così via.

Una volta individuato in quale lingua viene prodotto l'output comando (# 1), i possibili set di caratteri vengono limitati a uno o due (# 2). Inoltre, è noto quali parole chiave sono contenute in questo output (# 5).

La soluzione, pertanto, è decodificare l'output comando con una delle possibili codifiche cercando una parole chiave nel risultato. Se la parola chiave viene trovata, il set di caratteri corrente viene considerato quello corretto.

Definire un nuovo processo da utilizzare con i dati localizzati

Questo compito descrive come scrivere un nuovo processo utilizzabile con i dati localizzati.

Gli script Jython eseguono solitamente i comandi e ne analizzano l'output. Per ricevere questo output comando in maniera correttamente decodificata, utilizzare l'API per la classe **ShellUtils**. Per i dettagli consultare "API servizio Web di HP Universal CMDB - Panoramica" a pag. 292.

Questo codice assume solitamente la forma seguente:

```
client = Framework.createClient(protocol, properties)
shellUtils = shellutils.ShellUtils(client)
languageBundle = shellutils.getLanguageBundle ('langNetwork', shellUtils.osLanguage,
Framework)
strWindowsIPAddress = languageBundle.getString('windows_ipconfig_str_ip_address')
ipconfigOutput = shellUtils.executeCommandAndDecode('ipconfig /all',
strWindowsIPAddress)
#Do work with output here
```

1 Creare un client:

```
client = Framework.createClient(protocol, properties)
```

2 Creare un'istanza della classe **ShellUtils** e aggiungervi la lingua del sistema operativo. Se la lingua non viene aggiunta, viene utilizzata la lingua predefinita (solitamente inglese):

```
shellUtils = shellutils.ShellUtils(client)
```

Durante l'inizializzazione dell'oggetto, GFD rileva automaticamente la lingua del computer e imposta la codifica preferibile dall'oggetto Language predefinito. La codifica preferibile è la prima istanza visualizzata nell'elenco delle codifiche.

3 Recuperare il pacchetto delle risorse appropriate da **shellclient** utilizzando il metodo **getLanguageBundle**:

```
languageBundle = shellutils.getLanguageBundle (’langNetwork’,
shellUtils.osLanguage, Framework)
```

4 Recuperare una parola chiave dal pacchetto delle risorse, adatta per un comando specifico:

```
strWindowsIPAddress =
languageBundle.getString(’windows_ipconfig_str_ip_address’)
```

5 Richiamare il metodo **executeCommandAndDecode** e passare a esso la parola chiave sull'oggetto **ShellUtils**:

```
ipconfigOutput = shellUtils.executeCommandAndDecode(’ipconfig /all’,
strWindowsIPAddress)
```

L'oggetto ShellUtils viene anche utilizzato per collegare un utente al riferimento API (in cui questo metodo è descritto nel dettaglio).

6 Analizzare l'output come al solito.

Decodificare i comandi senza una parola chiave

L'approccio corrente per la localizzazione utilizza una parola chiave per decodificare tutto l'output comando. Per i dettagli consultare il passaggio 4 a pag. 85 in "Definire un nuovo processo da utilizzare con i dati localizzati" a pag. 84.

Un altro approccio, tuttavia, utilizza una parola chiave per decodificare solo il primo output comando e, successivamente, decodifica ulteriori comandi con il set di caratteri utilizzato per decodificare il primo comando. Per fare ciò, utilizzare i metodi **getCharsetName** e **useCharset** dell'oggetto **ShellUtils**.

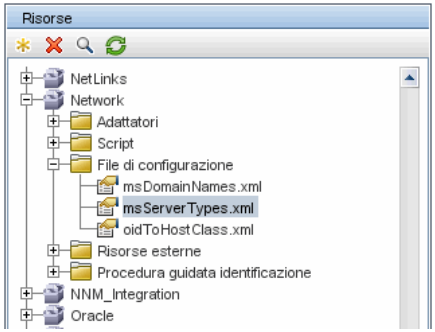
Il caso di utilizzo normale funziona come segue:

1. Richiamare il metodo **executeCommandAndDecode** una volta.
2. Ottenere il nome del set di caratteri utilizzato più di recente tramite il metodo **getCharsetName**.
3. Far utilizzare a **shellUtils** questo set di caratteri come impostazione predefinita richiamando il metodo **useCharset** sull'oggetto **ShellUtils**.
4. Richiamare il metodo **execCmd** di **ShellUtils** una o più volte. L'output viene restituito con il set di caratteri specificato nel passaggio 3. Non si verifica alcuna operazione di decodifica aggiuntiva.

Utilizzare i pacchetti di risorse

Un pacchetto di risorse è un file con estensione properties (***.properties**). Un file properties può essere considerato come un dizionario che archivia i dati in formato chiave = valore. Ciascuna riga in un file properties contiene un'associazione chiave = valore. La funzionalità principale di un pacchetto di risorse è quella di restituire un valore tramite la relativa chiave.

I pacchetti di risorse sono ubicati sul computer sonda: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles**. Vengono scaricati dal server UCMDB come qualsiasi altro file di configurazione. Possono essere modificati, aggiunti o rimossi nella finestra Risorse. Per i dettagli consultare "Riquadro File di configurazione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.



Quando si individua una destinazione, GFD deve solitamente analizzare il testo dall'output comando o dal contenuto del file. Questa analisi è spesso basata su un'espressione regolare. Lingue diverse richiedono espressioni regolari differenti da utilizzare per l'analisi. Per il codice da scrivere una volta per tutte le lingue, è necessario estrarre i dati specifici di tutte le lingue nei pacchetti di risorse. È presente un pacchetto di risorse per ciascuna lingua. (Sebbene sia possibile che un pacchetto di risorse contenga dati per lingue diverse, in GFD, un pacchetto di risorse contiene sempre i dati per una lingua.)

Lo script Jython stesso non include dati hardcoded specifici della lingua (ad esempio, espressioni regolari specifiche della lingua). Lo script determina la lingua del sistema remoto, carica il pacchetto di risorse adeguato e ottiene tutti i dati specifici della lingua tramite una chiave specifica.

In GFD, i pacchetti di risorse hanno un stesso formato specifico: <base_name>_<language_identifier>.properties, ad esempio, langNetwork_spa.properties. (Il pacchetto di risorse predefinito ha il seguente formato: <base_name>.properties, ad esempio, langNetwork.properties.)

Il formato base_name riflette lo scopo previsto di questo pacchetto. Ad esempio, **langMsCluster** indica che il pacchetto di risorse contiene le risorse specifiche della lingua utilizzate dai processi MS Cluster.

Il formato language_identifier è un acronimo di 3 lettere utilizzato per identificare la lingua. Ad esempio, rus indica la lingua russa e ger quella tedesca. Questo identificatore della lingua è incluso nella dichiarazione dell'oggetto Language.

Riferimento API

In questa sezione vengono trattati i seguenti argomenti:

Classe Language

Questa classe contiene tutte le informazioni sulla lingua come ad esempio il suffisso del pacchetto di risorse, la possibile codifica e così via.

Campi

Nome	Descrizione
impostazione internazionale	Oggetto Java che rappresenta l'impostazione internazionale.
bundlePostfix	Suffisso del pacchetto di risorse. Questo suffisso viene utilizzato nei nomi dei file dei pacchetti di risorse per identificare la lingua. Ad esempio, il pacchetto **langNetwork_ger.properties** include un suffisso di pacchetto di risorse **ger**.
charsets	Set di caratteri utilizzati per codificare questa lingua. Ciascuna lingua può avere diversi set di caratteri. Ad esempio, la lingua russa è comunemente codificata con la codifica Cp866 e Windows-1251.
wmiCodes	Elenco dei codici WMI utilizzati dal sistema operativo Microsoft Windows per identificare la lingua. Tutti i possibili codici sono elencati in http://msdn.microsoft.com/en-us/library/aa394239(VS.85).aspx (la sezione OSLanguage). Uno dei metodi per identificare la lingua del sistema operativo è quello di interrogare il sistema operativo della classe WMI riguardo alla proprietà OSLanguage.
codepage	Tabella codici utilizzata con una lingua specifica. Ad esempio, 866 è utilizzato per i computer russi e 437 per quelli inglesi. Uno dei metodi per identificare la lingua del sistema operativo è quello di recuperare la relativa tabella codici predefinita (ad esempio tramite il comando chcp).

Metodo executeCommandAndDecode

Questo metodo è destinato all'utilizzo da parte degli script Jython della logica aziendale. Esso contiene l'operazione di decodifica e restituisce un output comando decodificato.

Argomenti

Nome	Descrizione
cmd	Comando effettivo da eseguire.
keyword	Parole chiave da utilizzare per l'operazione di decodifica.
framework	L'oggetto Framework passato a tutti gli script Jython eseguibili in GFD.
timeout	Timeout del comando.
waitForTimeout	Specifica se il client deve attendere quando il timeout viene superato.
useSudo	Specifica se sudo deve essere utilizzato (relativo solo ai client del computer UNIX).
language	Consente di specificare direttamente la lingua invece di rilevarne automaticamente una.

Metodo getCharsetName

Questo metodo restituisce il nome del set di caratteri utilizzato più di recente.

Metodo useCharset

Questo metodo imposta il set di caratteri sull'istanza ShellUtils che utilizza questo set di caratteri per la decodifica iniziale dei dati.

Argomenti

Nome	Descrizione
charsetName	Il nome del set di caratteri, ad esempio windows-1251 o UTF-8.

Vedere anche "Metodo getCharsetName" a pag. 90.

Metodo getLanguageBundle

Questo metodo deve essere utilizzato per ottenere il pacchetto di risorse corretto. Sostituisce l'API seguente:

```
Framework.getEnvironmentInformation().getBundle(…)
```

Argomenti

Nome	Descrizione
baseName	Il nome del pacchetto senza suffisso della lingua, ad esempio langNetwork.
language	Oggetto della lingua. ShellUtils.osLanguage deve essere passato qui.
framework	Il Framework, l'oggetto comune passato a tutti gli script Jython eseguibili in GFD.

Campo osLanguage

Questo campo contiene un oggetto che rappresenta la lingua.

Utilizzare Discovery Analyzer

Lo strumento Discovery Analyzer è destinato a scopi di debug durante la distribuzione dei pacchetti, degli script e di altri contenuti. Lo strumento esegue un processo rispetto a una destinazione remota e restituisce registri contenenti informazioni, dettagli di avviso e di errore e i risultati dei CI individuati.

Tenere presente che i risultati non sono sempre segnalati all'interfaccia utente. Il motivo è che i risultati vengono segnalati in due modi e solo uno di questi è supportato. Inoltre, il registro di comunicazione non è supportato da Eclipse.

Quando si esegue lo strumento da Eclipse, il file **DiscoveryProbe.properties** (**C:\hp\UCMDB\DataFlowProbe\conf\DiscoveryProbe.properties**) deve contenere il seguente parametro impostato su **true**:

```
appilog.agent.local.discoveryAnalyzerFromEclipse = true
```

Per i dettagli consultare "Eseguire Discovery Analyzer da Eclipse" a pag. 101.

In tutti gli altri casi (quando lo strumento viene eseguito da **cmd** o mentre la sonda è in esecuzione) questo flag deve essere impostato su **false**:

```
appilog.agent.local.discoveryAnalyzerFromEclipse = false
```

Compiti e record

Un file compito contiene i dati relativi a un compito da eseguire. Il compito contiene informazioni quali il nome del processo e i parametri richiesti che definiscono il CI trigger, ad esempio l'indirizzo di destinazione remota.

Un file record contiene informazioni sul compito nonché i risultati di un'esecuzione specifica, ovvero la comunicazione dettagliata (che comprende una risposta) tra la sonda o Discovery Analyzer (a seconda del modulo che ha eseguito il compito) e la distribuzione remota.

Un compito definito da un file compito può essere eseguito rispetto a una destinazione remota mentre un compito definito da un file record (contenente dati aggiuntivi relativi a un'esecuzione specifica) può essere eseguito e anche riprodotto (ovvero, può riprodurre la stessa esecuzione documentata nel file record).

Registri

I registri forniscono informazioni sull'ultima esecuzione, come segue:

- **General Log**. Questo registro include tutti i dati, gli errori e gli avvisi delle informazioni verificatisi durante l'esecuzione.
- **Communication Log**. Questo registro contiene la comunicazione dettagliata tra Discovery Analyzer e la destinazione remota (inclusa la relativa risposta). Dopo l'esecuzione, è possibile salvare il registro come file record.
- **Results Log**. Visualizza un elenco dei CI individuati. Il tempo di visualizzazione di ciascun CI dipende dalla struttura degli adattatori e degli script.

È possibile salvare tutti i registri insieme o separatamente. Quando si salvano tutti i registri, questi vengono salvati insieme sotto un unico nome.

Se si riproduce un file record, gli stessi dati vengono visualizzati nel Communication Log; l'unica differenza è il tempo di esecuzione.

Limitazione: i registri Communication e Results non sono disponibili durante l'esecuzione di Discovery Analyzer tramite Eclipse.

La sezione è suddivisa nei passaggi seguenti:

➤ "Eseguire il compito" a pag. 98

➤ "Inviare un risultato del compito al server" a pag. 99

➤ "Importare le impostazioni" a pag. 99

➤ "Punti di interruzione" a pag. 100

1 Prerequisiti

➤ È necessario installare la sonda. (Discovery Analyzer è installato come parte del processo di installazione della sonda e condivide con essa le risorse.)

➤ Non è necessario eseguire la sonda mentre si utilizza Discovery Analyzer.

Tuttavia, se la sonda è già stata eseguita rispetto a un server UCMDB, tutte le risorse richieste sono già state scaricate sul file system. Se la sonda non viene eseguita, è possibile caricare le risorse necessarie a Discovery Analyzer tramite il menu Impostazioni. Per i dettagli consultare "Importare le impostazioni" a pag. 99.

➤ Non è necessario installare il server CMDB.

2 Accedere a Discovery Analyzer

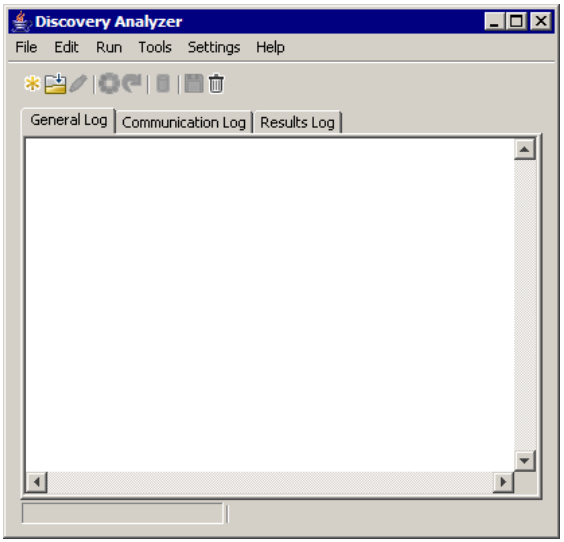
È possibile accedere a Discovery Analyzer:

➤ Quando si utilizza Eclipse.

L'installazione della sonda è dotata di un'area di lavoro Eclipse predefinita ubicata in **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace**. Questa area di lavoro include uno script Jython per avviare Discovery Analyzer (**startDiscoveryAnalyzerScript.py**) nonché un collegamento a tutti gli script GFD. Se si avvia lo strumento in questo modo, è possibile individuare i punti di interruzione all'interno degli script Jython per scopi di debug.

➤ Direttamente, facendo doppio clic sul file nella cartella seguente: **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzer.cmd**. Per i dettagli consultare la sezione seguente.

La finestra di Discovery Analyzer si apre:



3 Definire un compito

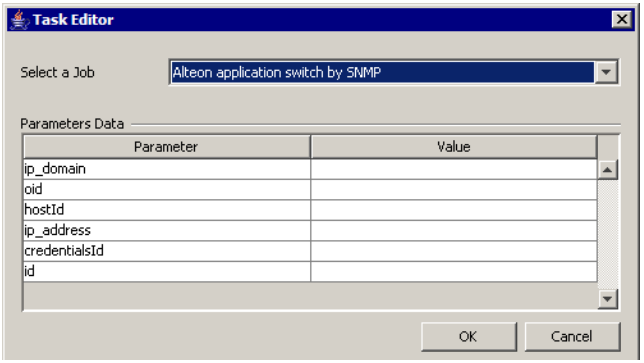
È possibile definire un compito utilizzando uno dei metodi seguenti:

- Definendo un nuovo compito. Per i dettagli consultare "Definire un nuovo compito" a pag. 96.
- Importando un compito da un file record. Per i dettagli consultare "Recuperare un record" a pag. 97.
- Importando un compito salvato da un file compito. Per i dettagli consultare "Aprire un file compito" a pag. 97.
- Recuperando un processo dal database interno della sonda. Per i dettagli consultare "Importare un compito dal database" a pag. 97.

4 Definire un nuovo compito

a Visualizzare l'editor del compito: fare clic sul pulsante **Nuovo compito**.

L'editor del compito visualizza un elenco di processi attualmente esistenti nel file system. Questo elenco viene aggiornato ogni volta che la sonda riceve compiti dal server o quando i pacchetti vengono distribuiti manualmente dal menu Impostazioni.



b Selezionare un processo.

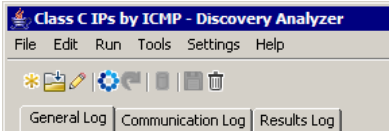
c Immettere i valori per tutti i parametri.

I parametri qui visualizzati sono i parametri dell'adattatore GFD. Possono essere visualizzati nel riquadro Parametri sequenza di individuazione nella scheda Firma sequenza. Per i dettagli consultare "Riquadro Parametri adattatore" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

Tutti i campi sono obbligatori (a meno che uno script del processo non richieda che il campo debba essere vuoto).

Per i parametri che richiedono un ID o un valore di input ID delle credenziali, è possibile utilizzare ID creati casualmente: fare clic con il pulsante destro del mouse sulla casella del valore e selezionare **Generate random CMDB ID** o **Credential Chooser**.

Il compito è ora attivo e il nome del compito aperto viene visualizzato nella barra del titolo:



d Continuare la procedura per definire un compito. Per i dettagli consultare "Salvare il compito e i registri" a pag. 98.

5 Recuperare un record

È possibile definire un compito aprendo un file record contenente i dati relativi a un'esecuzione specifica. Se un compito viene definito in questo modo, è possibile riprodurre l'esecuzione specifica selezionando l'opzione di riproduzione. (Se un compito viene riprodotto, le risposte vengono ricevute dai dati archiviati nel file record e non dalla destinazione remota.)

Selezionare **File > Open Record**. Passare alla cartella in cui è stato salvato il record. Il record è ora attivo e il nome del compito viene visualizzato nella barra del titolo.

Per i dettagli sull'acquisizione di un file record consultare "Registrare il codice GFD" a pag. 112.

6 Aprire un file compito

È possibile definire un compito da un file compito: selezionare **File > Open Task**.

7 Importare un compito dal database

È possibile recuperare un compito dal database della sonda a condizione che la sonda sia già stata eseguita e abbia compiti attivi nel relativo database interno. È possibile utilizzare i valori dei parametri per definire il compito.

a Selezionare **File > Import Task from Probe Database**.

b Nella finestra di dialogo che si apre, selezionare il compito da eseguire e fare clic su **OK**.

c Continuare la procedura per definire un compito. Per i dettagli consultare "Salvare il compito e i registri" a pag. 98.

8 Modificare un compito

Dopo la definizione di un compito, il nome del compito (o del file) viene visualizzato nella barra del titolo. Ora è possibile modificare il file.

a Selezionare **File** > **Edit Task**.

b Apportare eventuali cambiamenti al compito e fare clic su **OK**.

9 Salvare il compito e i registri

È possibile salvare i parametri del compito: Selezionare **File** > **Save Task**.

Le opzioni seguenti sono disponibili solo dopo l'esecuzione di un compito.

- Salvare un record del compito. È possibile salvare i parametri del compito e i risultati del compito eseguito: selezionare **File** > **Save Record**.
- Salvare un registro del compito: selezionare **File** > **Save General Log**.
- salvare i risultati: Selezionare **File** > **Save Results**.

10 Eseguire il compito

Il passaggio successivo della procedura è l'esecuzione del compito creato.

a Importare il file di configurazione credenziali/intervalli. Per i dettagli consultare "Importare le impostazioni" a pag. 99.

b Per eseguire il compito solo rispetto a una destinazione remota, fare clic sul pulsante **Run Task** .

Discovery Analyzer esegue il processo e visualizza le informazioni in tre file di registro: **General**, **Communication** e **Results**.

c È possibile salvare i file di registro insieme o separatamente: selezionare **File** > **Save General Log**, **Save Record**, **Save Results** o **Save All Logs**. Per i dettagli sui file di registro consultare "Registri" a pag. 93.

d Se il compito viene recuperato da un file record, l'esecuzione documentata in questo file può essere riprodotta facendo clic sul pulsante **Playback**. Viene visualizzato lo stesso Communication log ma il tempo di esecuzione viene aggiornato.

11 Inviare un risultato del compito al server

Se l'esecuzione di un compito termina con risultati (ovvero la scheda Results Log visualizza un elenco dei CI individuati), è possibile inviare i risultati al server UCMDB. Ciò risulta è utile se, ad esempio, in precedenza si stava eseguendo il test di uno script mentre il server non era accessibile.

Nota: è possibile inviare risultati solo a un server UCMDB che riceve compiti dalla sonda installata sullo stesso computer di Discovery Analyzer.

---

12 Importare le impostazioni

Per eseguire compiti o riprodurre un file record, è necessario importare il file **domainScopeDocument.bin**. Durante l'importazione, immettere una password.

a Avviare un browser Web e immettere l'URL seguente: **http://localhost:8080/jmx-console**. Potrebbe essere necessario effettuare l'accesso con nome utente e password.

b Fare clic su **UCMDB:service=DiscoveryManager** per aprire la pagina JMX MBEAN View.

c Individuare l'operazione **exportCredentialsAndRangesInformation**. Eseguire l'operazione seguente:

- Immettere l'ID del cliente (l'impostazione predefinita è **1**).
- Immettere un nome per il file esportato.
- Immettere la password.
- Impostare **isEncrypted** su **False**.

d Fare clic su **Invoke** per esportare il file **domainScopeDocument.bin**.

Quando il processo di esportazione viene completato correttamente, il file viene salvato nella posizione seguente: **C:\hp\UCMDB\UCMDBServer\conf\discovery\<customer_dir>**.

e Copiare il file **domainScopeDocument.bin** del file system della sonda del flusso di dati e importarlo selezionando: **Settings** > **Import domainScopeDocument**.

Nota: durante l'importazione del file **domainScopeDocument**, viene richiesto di fornire una password. Questa richiesta viene visualizzata anche dopo ogni avvio di Discovery Analyzer e prima dell'esecuzione del primo compito o record.

---

13 Punti di interruzione

Se si esegue Discovery Analyzer dallo script Python, è possibile aggiungere punti di interruzione allo script.

14 Configurare Eclipse

Per i dettagli sull'esecuzione degli script Jython in modalità di debug consultare"Eseguire Discovery Analyzer da Eclipse" a pag. 101.

Eseguire Discovery Analyzer da Eclipse

Questo compito spiega come configurare Eclipse in modo da eseguire gli script Jython in modalità di debug, consentendo così una migliore visibilità dei thread del processo, dei CI trigger e dei risultati.

La sezione è suddivisa nei passaggi seguenti:

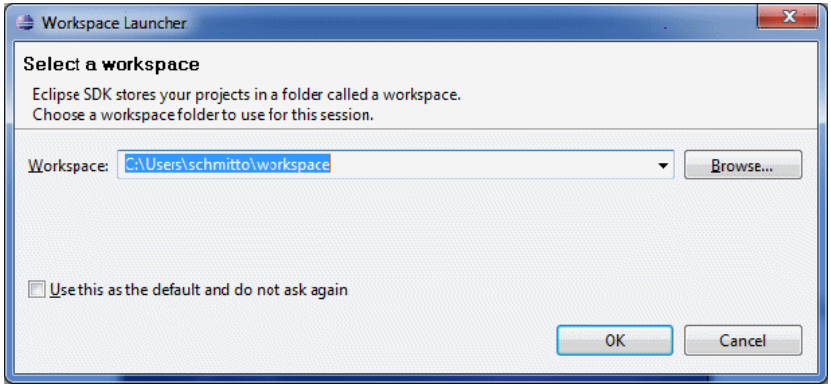
1 Prerequisiti

- Installare sul computer la versione più aggiornata di Eclipse. L'applicazione è disponibile all'indirizzo www.eclipse.org.
- Verificare che la sonda del flusso di dati sia installata sullo stesso computer.
- Verificare che il parametro **appilog.agent.local.discoveryAnalyzerFromEclipse** nel file **DiscoveryProbe.properties** sia impostato su **true**.

2 Decomprimere Eclipse e avviarlo

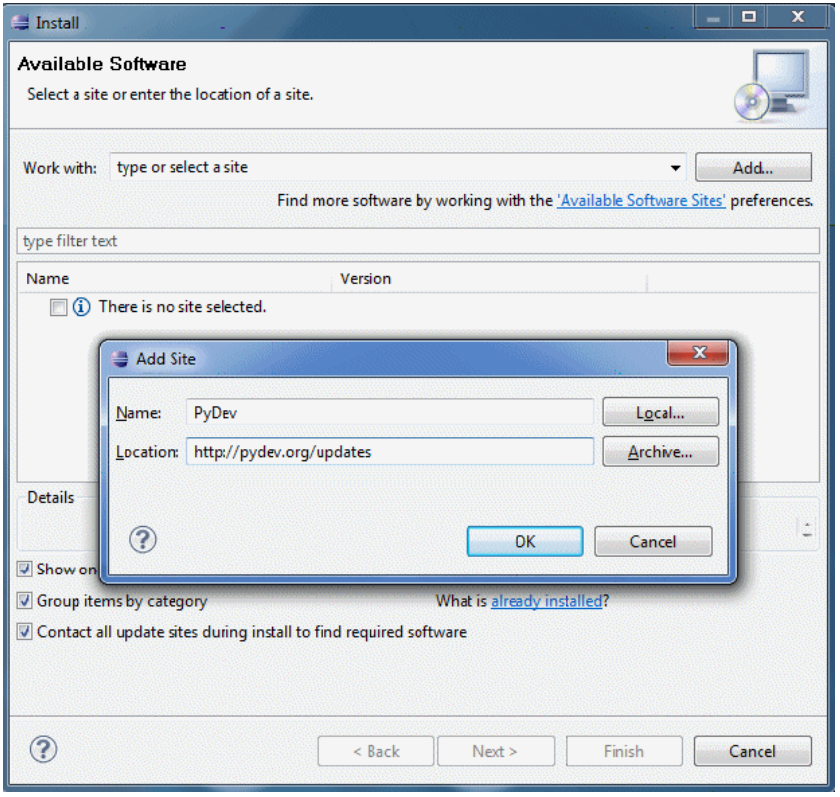
3 Configurare l'area di lavoro predefinita

Configurare l'area di lavoro predefinita in cui Eclipse salva e archivia tutti i progetti e i dati correlati.



4 Configurare le estensioni PyDev

a Accedere a **Help** > **Install New Software**, fare clic su **Add**, digitare un nome per il plug-in PyDev e nel campo Sede e aggiungere l'URL del sito in cui pydev può essere scaricato: **http://pydev.org/updates**. Fare clic su **OK**.



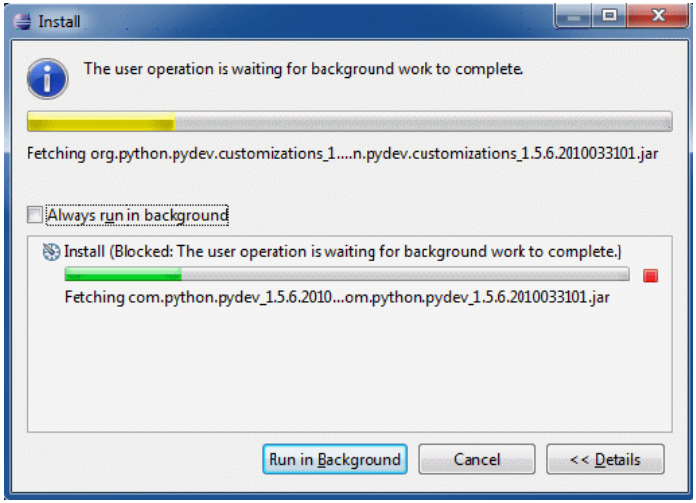
Nota: PyDev e le estensioni e PyDev sono ora unite in un singolo plug-in poiché le estensioni PyDev sono ora open source. Per le informazioni aggiuntive visitare il sito http://pydev.org.

---

b Nella finestra che si apre, selezionare **Pydev**. Il secondo plug-in è un plug-in per l'interfaccia utente focalizzata sul compito. Fare clic su **Next**, controllare i dettagli di installazione e fare nuovamente clic su **Next**.

c Accettare il contratto di licenza e fare clic su **Next**.

d Pydev è installato. Se viene chiesto di installare un contenuto non firmato, confermare facendo clic su **OK**.

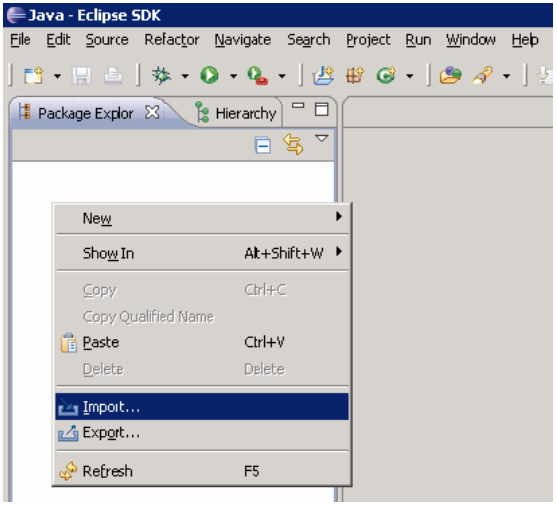


e Riavviare Eclipse.

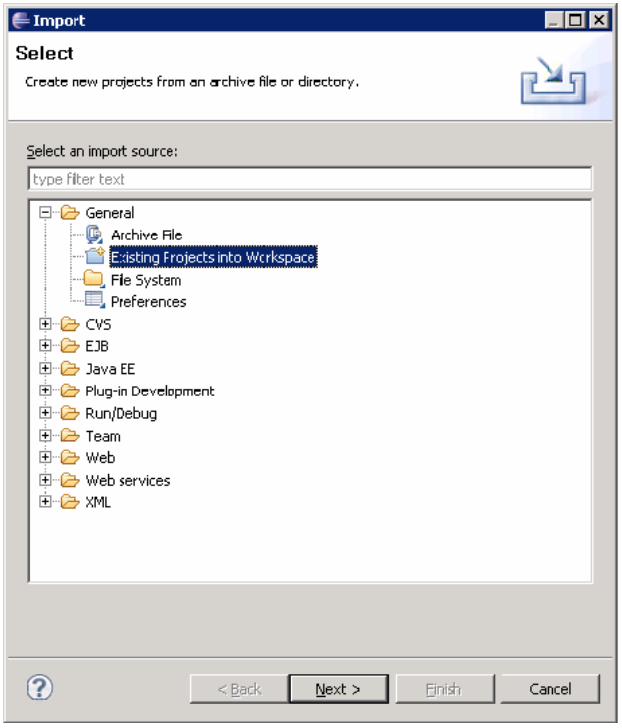
PyDev è ora installato nell'IDE Eclipse. Si dispone di nuove prospettive in Eclipse e l'IDE è in grado di interpretare gli script Python (evidenziazione testo, opzioni di configurazione aggiuntive e così via).

5 Configurare l'area di lavoro di Discovery Analyzer

a Importare i file necessari: fare clic con il pulsante destro del mouse nell'area bianca di esplorazione del pacchetto e fare clic su **Import** per importare **discoveryAnalyzerWorkspace** preconfigurato, in dotazione con l'installazione della sonda.



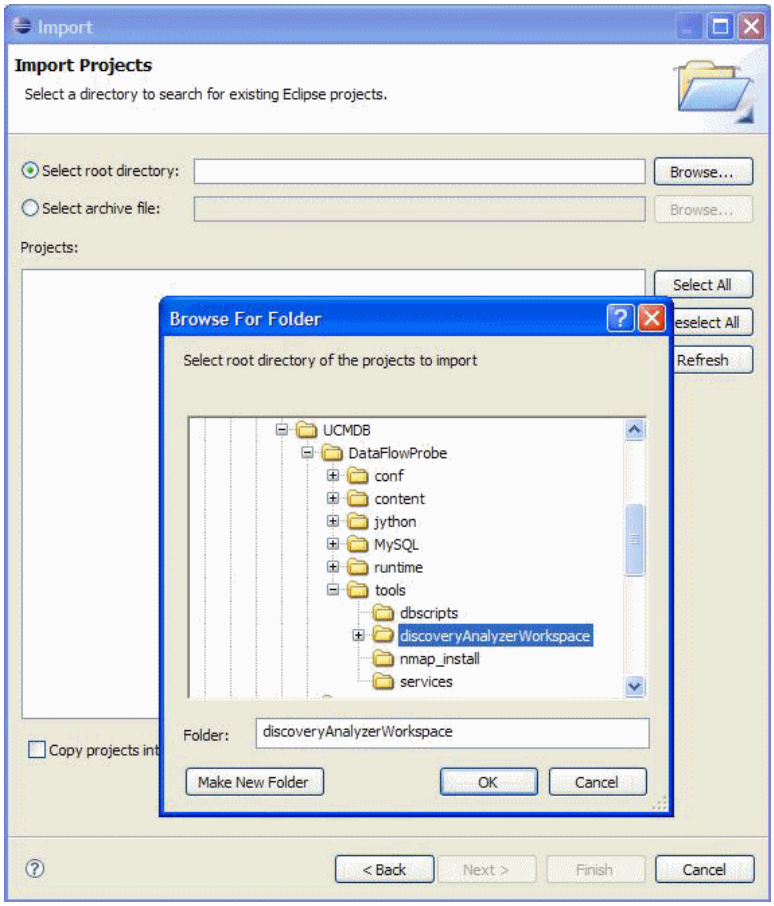
b In **General**, selezionare **Existing projects into Workspace** per importare il progetto nell'area di lavoro di Eclipse.

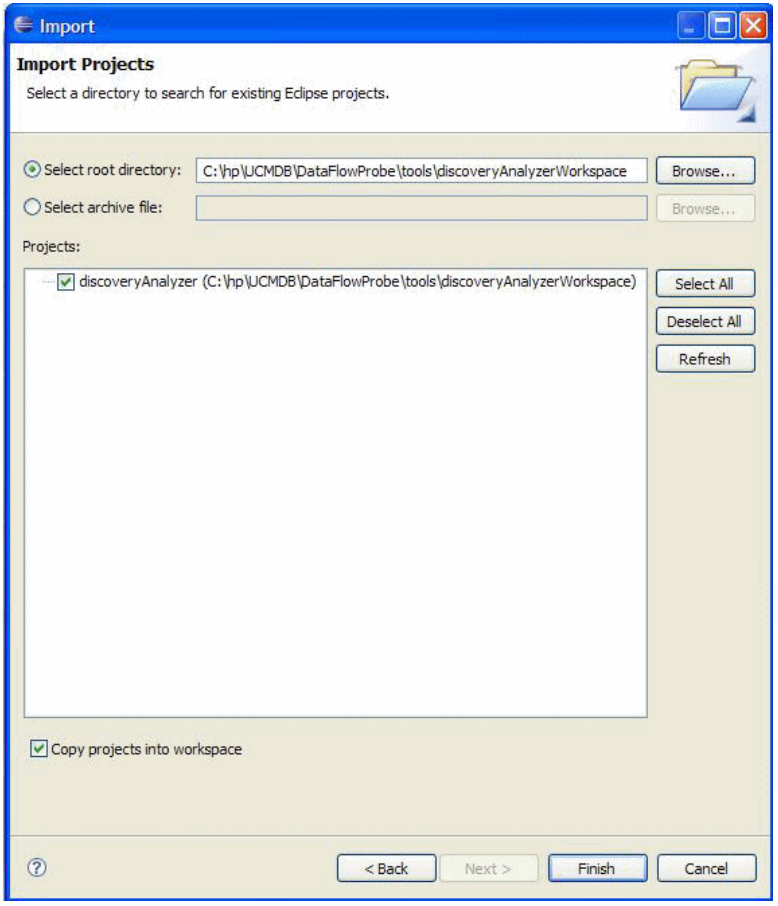


c In **Select root directory**, selezionare l'area di lavoro dell'analizzatore, ubicato solitamente in **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace**.

d Selezionare **Copy projects into workspace** per creare una copia reale dell'area di lavoro esistente. Questo è un passaggio importante: in caso di errore, è possibile reimportare **discoveryAnalyserWorkspace** originale.

e Fare clic su **Finish** per avviare l'importazione.





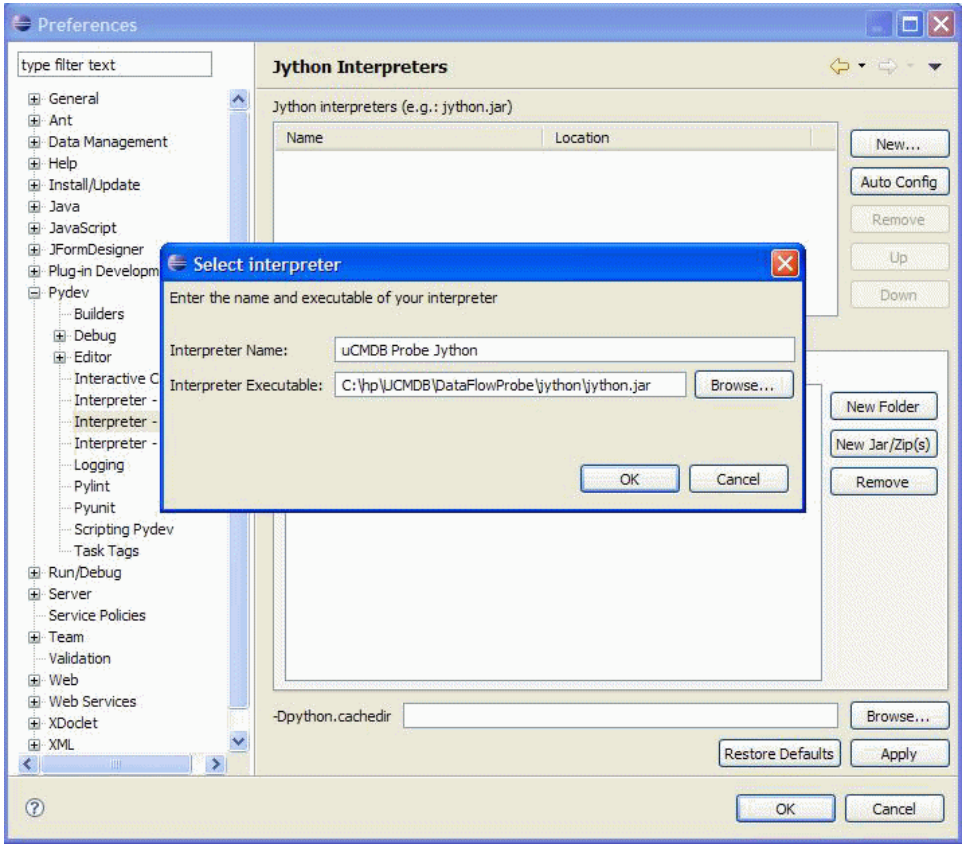
6 Configurare il classpath e l'interprete

a Fare clic con il pulsante destro del mouse su **discoveryAnalyzerWorkspace** e selezionare **Properties** per visualizzare le impostazioni specifiche del progetto.

b Passare a **Pydev > Interpreter/Grammar** e fare clic su **Please configure an interpreter in the related preferences before proceeding**.

Questo passaggio configura lo stesso interprete Jython utilizzato dalla sonda per garantire che gli script non vengano interpretati da una versione Jython diversa.

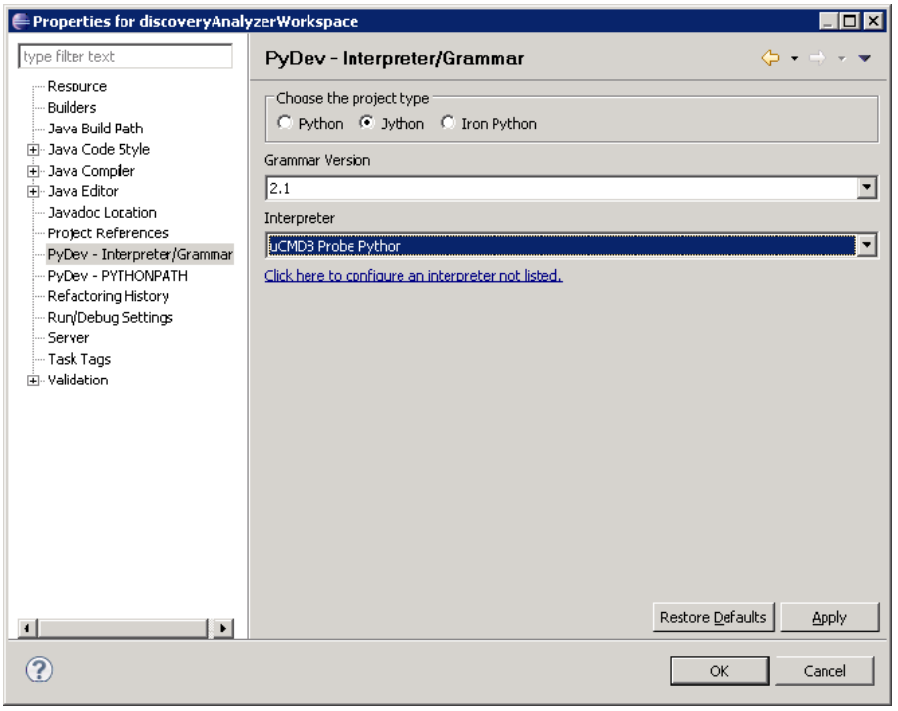
c Fare clic su **New**, digitare un nome per l'interprete e selezionare il file dalla cartella: **C:\hp\UCMDB\DataFlowProbe\jython\jython.jar**.



d Fare clic su **OK**. Se viene visualizzata una finestra che chiede di selezionare le cartelle da importare nel percorso di sistema Python, non cambiare nulla (deve essere **C:\hp\UCMDB\DataFlowProbe\jython** e **C:\hp\UCMDB\DataFlowProbe\jython\lib**) e fare clic su **OK**.

e Fare clic su **Apply** e quindi su **OK**.

f Fare clic su **Interpreter** e selezionare l'interprete appena creato.



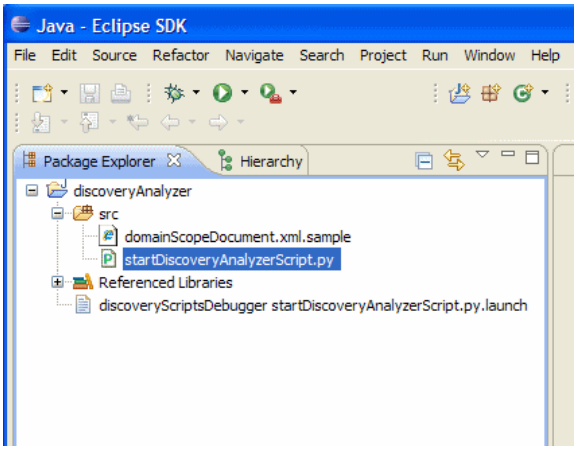
g Fare clic su **Apply** e quindi su **OK**.

L'interprete Jython è ora lo stesso di quello utilizzato dalla sonda.

7 Eseguire Discovery Analyzer

a Aggiungere un punto di interruzione nello script Jython di cui eseguire il debug.

b Per avviare Discovery Analyzer, selezionare **startDiscoveryAnalyzerScript.py** nel progetto **discoveryAnalyzerWorkspace\src**. Fare clic con il pulsante destro del mouse sul file e scegliere **Debug as > Jython run**.

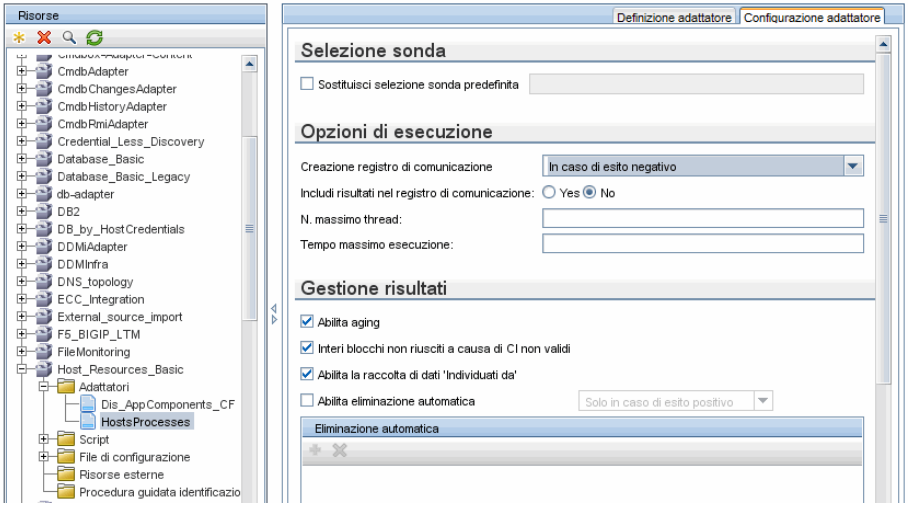


Registrare il codice GFD

Può essere molto utile registrare un'intera esecuzione, inclusi tutti i parametri, ad esempio, quando si effettua il debug o il test del codice. Questo compito descrive come registrare un'intera esecuzione con tutte le relative variabili. È possibile inoltre visualizzare informazioni aggiuntive sul debug spesso non stampate in file di registro anche a livello di debug.

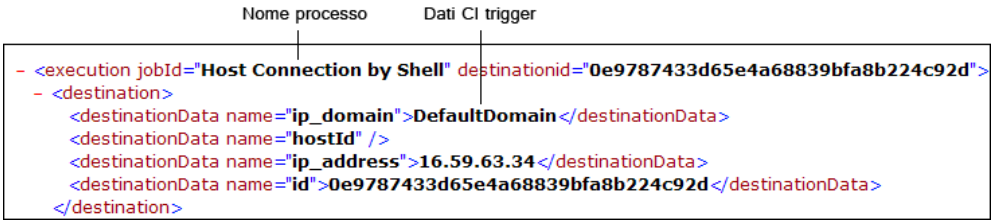
Per registrare il codice GFD:

1. Accedere a **Gestione flusso di dati** > **Pannello di controllo dell'individuazione**. Fare clic con il pulsante destro del mouse sul processo la cui esecuzione deve essere registrata e selezionare **Modifica adattatore** per aprire l'applicazione Gestione adattatore.
2. Individuare il riquadro **Opzioni di esecuzione** nella scheda Gestione sequenza:

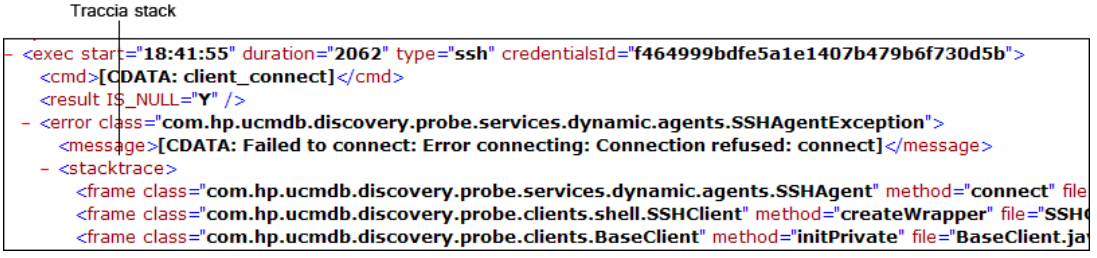


3. Cambiare la casella **Creazione registro di comunicazione** in **Sempre**. Per i dettagli sull'impostazione delle opzioni di registrazione consultare "Riquadro Opzioni di esecuzione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

L'esempio seguente è il file di registro XML creato quando il processo Host Connection by Shell viene eseguito e la casella **Creazione registro di comunicazione** è impostata su **Sempre** o **In caso di esito negativo**:



L'esempio seguente mostra il messaggio e i parametri della traccia dello stack:



Riferimenti

Librerie e utilità Jython

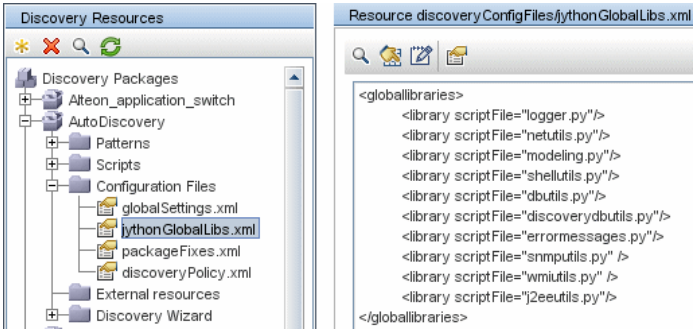
Negli adattatori sono largamente utilizzati diversi script di utilità. Questi script fanno parte del pacchetto AutoDiscovery e si trovano in: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts** con gli altri script scaricati nella sonda.

Nota: la cartella discoveryScript viene creata dinamicamente quando la sonda comincia a funzionare.

Per utilizzare uno degli script di utilità, aggiungere la seguente riga di importazione per importare la sezione dello script.

```
import <script name>
```

La libreria AutoDiscovery Python contiene gli script di utilità Jython. Questi script di libreria sono considerati libreria esterna di GFD. Essi vengono definiti nel file jythonGlobalLibs.xml (ubicato nella cartella **File di configurazione**).



Ciascuno script visualizzato nel file jythonGlobalLibs.xml viene caricato, per impostazione predefinita, all'avvio della sonda, pertanto non è necessario utilizzarli esplicitamente nella definizione dell'adattatore.

In questa sezione vengono trattati i seguenti argomenti:

logger.py

Lo script **logger.py** contiene le utilità di registro e le funzioni di supporto per la segnalazione degli errori. È possibile chiamare il relativo debug, le relative informazioni e le API di errore da scrivere nei file di registro. I messaggi di registro vengono registrati in **C:\hp\UCMDB\DataFlowProbe\runtime\log**.

I messaggi vengono immessi nel file di registro in base al livello di debug definito per l'appender PATTERNS_DEBUG nel file **C:\hp\UCMDB\DataFlowProbe\conf\log\probeMgrLog4j.properties**. (Per impostazione predefinita, il livello è DEBUG.) Per i dettagli consultare "Livelli di gravità degli errori" a pag. 125.

```
#########################################################################
################       PATTERNS_DEBUG log              ##################
#########################################################################
log4j.category.PATTERNS_DEBUG=DEBUG, PATTERNS_DEBUG
log4j.appender.PATTERNS_DEBUG=org.apache.log4j.RollingFileAppender
log4j.appender.PATTERNS_DEBUG.File=C:\hp\UCMDB\DataFlowProbe\runtime\log/pr
obeMgr-patternsDebug.log
log4j.appender.PATTERNS_DEBUG.Append=true
log4j.appender.PATTERNS_DEBUG.MaxFileSize=15MB
log4j.appender.PATTERNS_DEBUG.Threshold=DEBUG
log4j.appender.PATTERNS_DEBUG.MaxBackupIndex=10
log4j.appender.PATTERNS_DEBUG.layout=org.apache.log4j.PatternLayout
log4j.appender.PATTERNS_DEBUG.layout.ConversionPattern=<%d> [%-5p] [%t] -
%m%n
log4j.appender.PATTERNS_DEBUG.encoding=UTF-8
```

I messaggi di informazione e di errore vengono anche visualizzati nella console Prompt dei comandi.

Sono possibili due set di API:

- logger.<debug/info/warn/error>
- logger.<debugException/infoException/warnException/errorException>

Il primo set emette la concatenazione di tutti i relativi argomenti di stringa al livello di registro appropriato, mentre il secondo set emette la concatenazione nonché l'esecuzione della traccia dello stack dell'eccezione generata più recentemente, per fornire ulteriori informazioni, ad esempio:

```
logger.debug('found the result')
logger.errorException('Error in discovery')
```

modeling.py

Lo script **modeling.py** contiene le API per la creazione di host, IP, CI di processo e così via. Queste API consentono la creazione di oggetti comuni e rendono il codice più leggibile. Ad esempio:

```
ipOSH= modeling.createIpOSH(ip)
host = modeling.createHostOSH(ip_address)
member1 = modeling.createLinkOSH('member', ipOSH, networkOSH)
```

netutils.py

La libreria **netutils.py** viene utilizzata per recuperare le informazioni TCP e di rete, come ad esempio il recupero dei nomi del sistema operativo, controllando se un indirizzo MAC è valido, controllando se un indirizzo IP è valido e così via. Ad esempio:

```
dnsName = netutils.getHostName(ip, ip)
isValidIp = netutils.isValidIp(ip_address)
address = netutils.getHostAddress(hostName)
```

shellutils.py

La libreria **shellutils.py** fornisce un'API per l'esecuzione dei comandi shell e per il recupero dello stato finale di un comando eseguito e consente l'esecuzione di più comandi in base a tale stato finale. La libreria viene inizializzata con un client shell e utilizza il client per eseguire i comandi e recuperare i risultati. Ad esempio:

```
ttyClient = clientFactory.createClient(Props)
clientShUtils = shellutils.ShellUtils(ttyClient)
if (clientShUtils.isWinOs()):
        logger.debug ('discovering Windows..')
```

4

Messaggi di errore

Questo capitolo comprende:

Concetti

Riferimenti

Concetti

Messaggi di errore - Panoramica

Durante l'individuazione, molti errori potrebbero non essere individuati, come ad esempio gli errori di connessione, i problemi hardware, le eccezioni, i timeout e così via. GDF visualizza questi errori nel Pannello di controllo dell'individuazione, sia in Modalità di base sia in Modalità avanzata, ogni volta che non si verifica il flusso di individuazione regolare. Per visualizzare lo specifico messaggio di errore, eseguire il drill down a partire dal CI trigger che ha causato il problema.

GDF distingue tra errori che possono talvolta essere ignorati (ad esempio un host non raggiungibile) ed errori che richiedono un intervento (ad esempio problemi di credenziali oppure file di configurazione o DLL mancanti). Inoltre, GDF segnala gli errori una sola volta, anche se lo stesso errore si verifica durante le successive esecuzioni, e segnala un errore anche se questo si verifica solo una volta.

Quando si crea un pacchetto, è possibile aggiungere al pacchetto messaggi appropriati come risorse. Durante la distribuzione del pacchetto, anche i messaggi vengono distribuiti nella posizione corretta. I messaggi devono essere conformi alle convenzioni, come descritto in "Convenzioni di scrittura errori" a pag. 121.

GDF supporta messaggi di errore multilingue. È possibile individuare i messaggi scritti in modo da visualizzarli nella lingua locale.

Per i dettagli sulla ricerca degli errori consultare "Riquadro Stato individuazione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

Per i dettagli sull'impostazione dei registri di comunicazione consultare "Riquadro Opzioni di esecuzione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

Riferimenti

Convenzioni di scrittura errori

- Ciascun errore viene identificato da un codice messaggio di errore e da un array di argomenti (**int, String[]**). La combinazione di un codice messaggio e un array di argomenti definisce un errore specifico. L'array dei parametri può essere null.
- Ciascun codice errore è mappato a un **messaggio breve** rappresentato da una stringa fissa e a un **messaggio dettagliato** ovvero una stringa esemplare contenente zero o più argomenti. Si presuppone una corrispondenza tra il numero di argomenti nell'esemplare e il numero di parametri effettivi.

  Esempio di codice messaggio di errore:

  10234 può rappresentare un errore con il messaggio breve:

  ```
  Errore di connessione
  ```

  e il messaggio dettagliato:

  ```
  Impossibile connettersi tramite protocollo {0} a causa di un timeout di {1} msec
  ```

  dove

  {0} = il primo argomento: un nome protocollo

  {1} = il secondo argomento: la lunghezza del timeout in msec

La sezione è suddivisa negli argomenti seguenti:

Contenuto file proprietà

Un file proprietà deve contenere due chiavi per ciascun codice messaggio di errore. Ad esempio, per l'errore 45:

- **DDM_ERROR_MESSAGE_SHORT_45**. Breve descrizione dell'errore.
- **DDM_ERROR_MESSAGE_LONG_45**. Lunga descrizione dell'errore (può contenere dei parametri, ad esempio {**0**},{**1**}).

File proprietà messaggi di errore

Un file proprietà contiene una mappa tra un codice messaggio di errore e due messaggi (breve e dettagliato).

Dopo che il file proprietà è stato distribuito, i relativi dati vengono uniti con i dati esistenti, ovvero i nuovi codici dei messaggi vengono aggiunti mentre i codici dei messaggi precedenti vengono sostituiti.

I file proprietà dell'infrastruttura fanno parte del pacchetto **AutoDiscoveryInfra**.

Convenzioni di denominazione delle impostazioni internazionali

- Per le impostazioni internazionali predefinite: **<nome file>.properties.errors**
- Per le impostazioni internazionali specifiche: **<nome file>_xx.properties.errors**

  dove **xx** rappresenta le impostazioni internazionali (ad esempio, **infraerr_fr.properties.errors** o **infraerr_en_us.properties.errors**).

Codici dei messaggi di errore

Per impostazione predefinita, con HP Universal CMDB sono inclusi i seguenti codici di errore. È possibile aggiungere i propri messaggi di errore al presente elenco.

Nome errore	Codice errore	Descrizione
Interno	100-199	Nella maggior parte dei casi risolti dalle eccezioni generate durante le esecuzioni dello script Jython
Connessione	200-299	Connessione non riuscita, nessun agente sul computer di destinazione, destinazione non raggiungibile, e così via
Correlato alle credenziali	300-399	Permesso negato, tentativo di connessione bloccato a causa della mancanza di credenziali
Timeout	400-499	Timeout durante la connessione/il comando
Comportamento imprevisto o non valido	500-599	File di configurazione mancanti, interruzioni impreviste e così via
Recupero informazioni	600-699	Informazioni mancanti sui computer di destinazione, errore nella richiesta di informazioni all'agente e così via
Correlato alle risorse	700-799	Errori correlati alla memoria insufficiente o al rilascio non corretto dei client
Analisi	800-899	Errore durante l'analisi del testo
Codifica	900	Errore nella codifica di input non supportata
Correlato a SQL	901-903, 924	Errori ricevuti dalle operazioni SQL
Correlato a HTTP	904-909	Errori generati durante le connessioni HTTP, analizzati dai codici di errore HTTP.
Specifico dell'applicazione	910-923	Errore segnalato in presenza di problemi specifici dell'applicazione, ad esempio versione LSOF errata, Gestioni code non trovate e così via

Errori di contenuto non classificati

Per supportare il contenuto precedente senza causare una regressione, l'applicazione e i metodi pertinenti a SDK gestiscono in modo diverso gli errori associati al codice del messaggio 100 (ovvero un errore di script non classificato).

Tali errori non vengono raggruppati (non sono considerati errori dello stesso tipo) in base al rispettivo codice del messaggio ma in base al contenuto del messaggio. Ciò significa che, se uno script segnala un errore mediante i metodi precedenti e obsoleti (con una stringa di messaggio e senza un codice errore), tutti i messaggi ricevono lo stesso codice errore, ma nell'applicazione o nei metodi pertinenti a SDK vengono visualizzati messaggi differenti come errori diversi.

Cambiamenti nel Framework

(com.hp.ucmdb.discovery.library.execution.BaseFramework)

I metodi seguenti vengono aggiunti all'interfaccia:

- void reportError(int msgCode, String[] params);
- void reportWarning(int msgCode, String[] params);
- void reportFatal(int msgCode, String[] params);

I metodi precedenti di seguito elencati sono ancora supportati per la compatibilità con le versioni precedenti, ma sono contrassegnati come obsoleti:

- void reportError(String message);
- void reportWarning (String message);
- void reportFatal (String message);

Livelli di gravità degli errori

Quando termina l'esecuzione di un adattatore rispetto a un CI trigger, viene restituito uno stato. Se non viene segnalato alcun errore o avviso, lo stato sarà **Operazione riuscita**.

I livelli di gravità sono elencati qui, dall'ambito più specifico a quello più generale:

Errori irreversibili

Questo livello segnala errori gravi, come un problema dell'infrastruttura, file DLL mancanti oppure eccezioni:

- Generazione compiti non riuscita (Impossibile trovare la sonda, impossibile trovare le variabili e così via)
- Impossibile eseguire lo script
- Errore di elaborazione dei risultati sul server e scrittura dati non eseguita su CMDB

Errori

Questo livello segnala i problemi che impediscono a GDF di recuperare i dati. Consultare questi errori poiché in genere richiedono l'esecuzione di un'azione (ad esempio, l'aumento del timeout, il cambiamento di un intervallo, la modifica di un parametro, l'aggiunta di un'altra credenziale dell'utente e così via).

- Nei casi in cui l'intervento dell'utente potrebbe risultare utile, viene segnalato un errore riguardante un problema di credenziali o di rete che potrebbe richiedere un'ulteriore verifica. (Non si tratta di errori nell'individuazione ma nella configurazione).
- Errore interno, in genere causato da un comportamento imprevisto dell'applicazione o del computer individuato, ad esempio file di configurazione mancanti e così via.

Avviso

Quando un'esecuzione viene completata correttamente ma potrebbero verificarsi errori non gravi di cui è opportuno essere a conoscenza, GFD contrassegna il livello di gravità come **Avviso**. È opportuno consultare questi CI per verificare l'eventuale mancanza dei dati prima di avviare una sessione di debug più dettagliata. **Avviso** può includere messaggi che segnalano la mancanza di un agente installato su un host remoto o il calcolo errato di un attributo a causa di dati non validi.

- Agente di connessione mancante (SNMP, WMI).
- L'individuazione è stata completata correttamente, ma non tutte le informazioni disponibili sono state individuate.

5

Sviluppo degli adattatori generici del database

Questo capitolo comprende:

Concetti

Compiti

Riferimenti

Concetti

Adattatore generico del database - Panoramica

Lo scopo della piattaforma dell'adattatore generico del database è la creazione di adattatori che possano integrarsi con i sistemi per la gestione di database relazionali (RDBMS) ed eseguire query TQL e processi di popolamento rispetto database. Gli RDBMS supportati dall'adattatore generico del database sono Oracle, Microsoft SQL Server e MySQL.

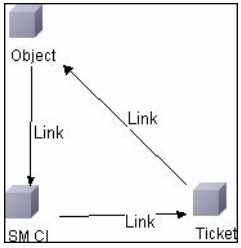
Questa versione dell'implementazione dell'adattatore generico si basa su uno standard JPA (Java Persistence API) standard con libreria Hibernate ORM come provider di persistenza.

Query TQL non supportate

Le limitazioni seguenti riguardano soltanto le query TQL calcolate dall'adattatore generico del database:

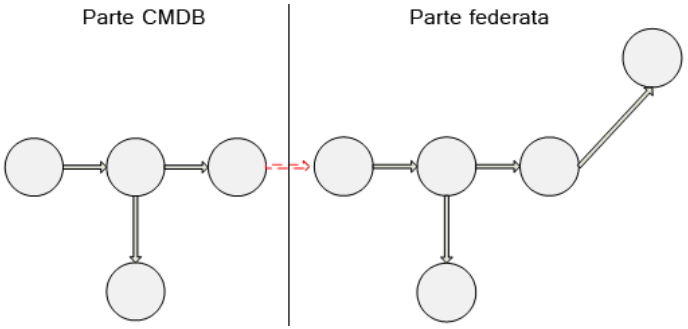
- I sottografici non sono supportati
- Le relazioni compound non sono supportate
- I cicli o parti di essi non sono supportati

  La query TQL seguente è un esempio di ciclo:



- Il layout della funzione non è supportato.
- La cardinalità 0..0 non è supportata.
- La relazione join non è supportata.

- Le condizioni del qualificatore non sono supportate.
- Per collegare due CI è necessario che esista una relazione sotto forma di tabella o chiave esterna nella sorgente esterna del database.



Riconciliazione

La riconciliazione viene eseguita nell'ambito del calcolo TQL lato adattatore. Per realizzare la riconciliazione, il lato CMDB deve essere mappato a un'entità federata denominata CIT riconciliazione.

Mappatura. Ciascun attributo nel CMDB è mappato a una colonna nell'origine di dati.

Anche se la mappatura viene eseguita direttamente, vengono supportate anche le funzioni di trasformazione sulla mappatura dei dati. È possibile aggiungere nuove funzioni mediante il codice Java (ad esempio minuscolo, maiuscolo). Lo scopo di queste funzioni è abilitare le conversioni dei valori (i valori memorizzati nel CMDB in un formato e nel database federato in un altro formato).

Nota:

- Per collegare il CMDB e la sorgente esterna del database, è necessario che esista un'associazione adeguata nel database. Per i dettagli consultare "Prerequisiti" a pag. 135.
- È inoltre supportata la riconciliazione con l'id del CMDB.

Hibernate come provider JPA

Hibernate è uno strumento di mappatura di tipo object-relational (OR) che consente la mappatura di classi Java a tabelle su diversi tipi di database relazionali (ad esempio Oracle e Microsoft SQL Server). Per i dettagli consultare "Limitazioni funzionali" a pag. 219.

In una mappatura elementare, ciascuna classe Java viene mappata a una sola tabella. Una mappatura più avanzata consente la mappatura ereditarietà (come si può verificare nel database CMDB).

Altre funzioni supportate comprendono la mappatura di una classe a diverse tabelle, il supporto di raccolte e le associazioni di tipo one-to-one, one-to-many e many-to-one. Per i dettagli consultare "Associazioni" a pag. 133.

Per i nostri fini non è necessario creare classi Java. La mappatura è definita dai CIT del modello di classe CMDB alle tabelle del database.

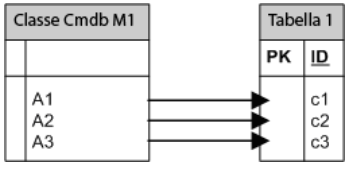
La sezione è suddivisa negli argomenti seguenti:

Esempi di Object-Relational Mapping

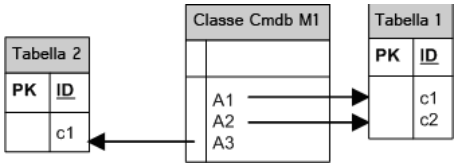
L'esempio seguenti descrive l'object-relational mapping:

Esempio di una classe CMDB mappata a una tabella di database:

La classe M1, con attributi A1, A2 e A3 è mappata alla tabella 1 colonne c1, c2 e c3. Ciò significa che qualsiasi istanza M1 ha una riga corrispondente nella tabella 1.

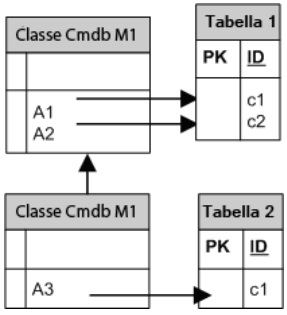


Esempio di una classe CMDB mappata a due tabelle di database:



Esempio di ereditarietà:

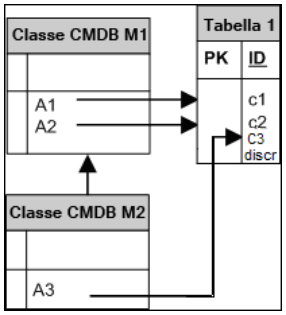
Questo caso viene utilizzato nel CMDB dove ciascuna classe ha una propria tabella di database.



Esempio di ereditarietà di una tabella con discriminatore:

Un'intera gerarchia di classi è mappata a una sola tabella di database le cui colonne comprendono un super-set di tutti gli attributi di classi mappate. La tabella contiene anche un'altra colonna (Discriminatore), i cui valori indicato la classe specifica da mappare a questa voce.

Quando si utilizzano le funzioni del discriminatore, non è possibile saltare una classe nella gerarchia: poiché C3 eredita da C2 e C2 eredita da C1, non è possibile definire soltanto C1 e C3, è necessario definire tutte e tre le classi.



Associazioni

Sono possibili tre tipi di associazioni: one-to-many, many-to-one e many-to-many. Per collegare diversi oggetti del database, è necessario definire una di queste associazioni utilizzando una colonna di chiavi esterne (per il caso one-to-many) oppure una tabella di mappatura (per il caso many-to-many).

Usabilità

Poiché lo schema JPA è molto ampio, viene fornito un file XML semplificato per facilitare le definizioni.

L'applicazione pratica di questo file XML è la seguente: i dati federati sono modellati in una classe federata. Questa classe ha relazioni many-to-one con una classe CMDB non federata. Inoltre, è possibile un solo tipo di relazione tra la classe federata e la classe non federata.

Compiti

Preparare la creazione di un adattatore

Questo compito descrive le preparazioni necessarie per la creazione di un adattatore.

Nota: è possibile visualizzare esempi dell'adattatore generico di DB nell' API UCMDB. In particolare, l'esempio di adattatore DDMi contiene un file **orm.xml** complicato e le implementazioni di alcune interfacce del plug-in.

Questo compito include i passaggi seguenti:

- "Prerequisiti" a pag. 135
- "Creare un tipo di CI" a pag. 137
- "Creare una relazione" a pag. 138

1 Prerequisiti

Per convalidare l'utilizzo dell'adattatore del database con il proprio database, verificare quanto segue:

➤ Nel database esistono le classi di riconciliazione e i relativi attributi (noti anche come multinodo). Ad esempio, se la riconciliazione viene eseguita in base al nome del nodo, verificare che vi sia una tabella che contiene una colonna con i nomi dei nodi. Se la riconciliazione viene eseguita in base al nodo cmdb_id, verificare che vi sia una colonna con gli ID del CMDB che corrisponda agli ID del CMDB dei nodi nel CMDB. Per i dettagli sulla riconciliazione consultare "Riconciliazione" a pag. 130.

ID	NAME	IP_ADDRESS
31	BABA	16.59.33.60
33	ext3.devlab.ad	16.59.59.116
46	LABM1MAM15	16.59.58.188
72	cert-3-j2ee	16.59.57.100
102	labm1sun03.devlab.ad	16.59.58.45
114	LABM2PCOE73	16.59.66.79
116	CUT	16.59.41.214
117	labm1hp4.devlab.ad	16.59.60.182

➤ Per correlare due CIT con una relazione è necessario che vi siano dati di correlazione tra le tabelle dei CIT. La correlazione può essere in base a una colonna di chiavi esterne oppure in base a una tabella di mappatura. Ad esempio, per correlare un nodo e un ticket, è necessario che vi sia una colonna nella tabella ticket che contenga l'ID del nodo, una colonna nella tabella dei nodi con l'ID del ticket che sia collegato a essa oppure una tabella di mappatura il cui end1 sia l'ID del nodo e l'end2 sia l'ID del ticket. Per i dettagli sui dati di correlazione consultare "Hibernate come provider JPA" a pag. 131.

Nella tabella seguente viene mostrata la colonna NODE_ID della chiave esterna:

NODE_ID	CARD_ID	CARD_TYPE	CARD_NAME
2015	1	Serial Bus Controller	Intel ® 82801EB USB Universal Host Controller
3581	2	System	Intel ® 631xESB/6321ESB/3100 Chipset LPC
3581	3	Display	ATI ES1000
3581	4	Base System Peripheral	HP ProLiant iLO 2 Legacy Support Function

➤ Ciascun CIT può essere mappato a una o più tabelle. Per mappare un CIT a più di una tabella, verificare che vi sia una tabella primaria la cui chiave primaria esista nella altre tabelle e che sia una colonna con valori univoci.

Ad esempio un ticket è mappato a due tabelle: ticket1 e ticket2. La prima tabella ha le colonne c1 e c2 e la seconda tabella ha le colonne c3 e c4. Per consentire che siano considerate come una sola tabella entrambe devono avere la stessa chiave primaria. In alternativa, la chiave primaria della prima tabella può essere una colonna nella seconda tabella.

Nell'esempio seguente, le tabelle condividono la stessa chiave primaria denominata CARD_ID:

CARD_ID	CARD_TYPE	CARD_NAME
1	Serial Bus Controller	Intel ® 82801EB USB Universal Host Controller
2	System	Intel ® 631xESB/6321ESB/3100 Chipset LPC
3	Display	ATI ES1000
4	Base System Peripheral	HP ProLiant iLO 2 Legacy Support Function

CARD_ID	CARD_VENDOR
1	Hewlett-Packard Company
2	(Controller host USB standard)
3	Hewlett-Packard Company
4	(Periferiche di sistema standard)
5	Hewlett-Packard Company

2 Creare un tipo di CI

In questo passaggio viene creato un CIT federato da mappare ai dati nel RDBMS (origine di dati esterna).

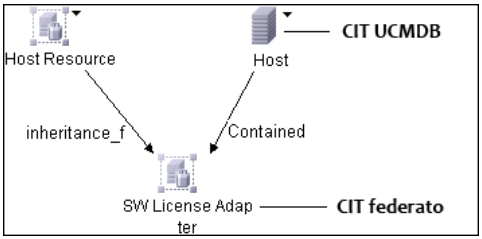
a In UCMDB, accedere a Gestione tipo CI e creare un nuovo tipo CI. Per i dettagli consultare "Creare un tipo di CI" nella *Guida alla modellazione di HP Universal CMDB*.

b Aggiungere gli attributi necessari al CIT, ad esempio l'ultima ora di accesso, il fornitore e così via. Questi sono gli attributi che l'adattatore recupera dall'origine di dati esterna per portarli nelle viste CMDB.

3 Creare una relazione

In questo passaggio viene aggiunta una relazione tra il CIT UCMDB e il nuovo CIT che rappresenta i dati da federare dall'origine di dati esterna.

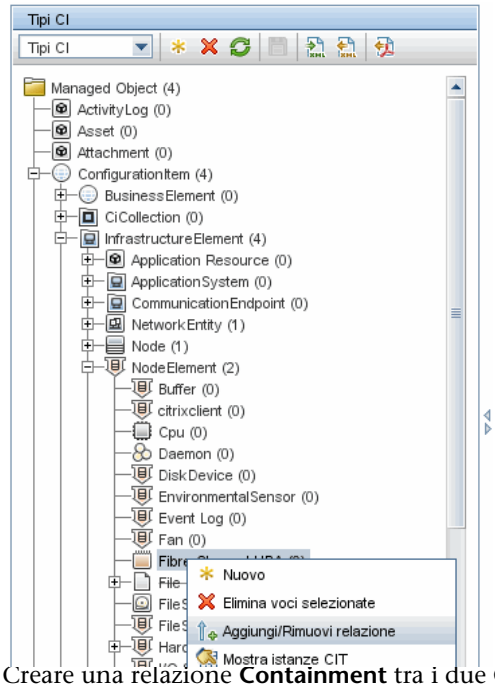
Aggiungere le relazioni adeguate, valide al nuovo CIT. Per i dettagli consultare "Finestra di dialogo Aggiungi/Rimuovi relazione" nella *Guida alla modellazione di HP Universal CMDB*.



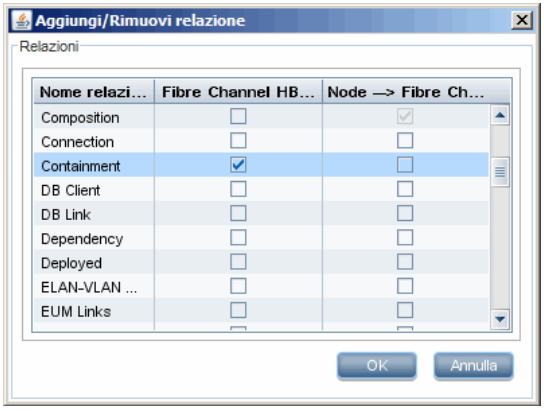
Nota: a questo punto non è ancora possibile visualizzare i dati federati poiché non è stato ancora definito il metodo per portare dentro i dati.

Esempio di creazione di una relazione Containment:

1 In Gestione tipo CI selezionare i due CIT:



2 Creare una relazione **Containment** tra i due CIT:



Preparare il pacchetto dell'adattatore

In questo passaggio individuare e configurare il pacchetto dell'adattatore generico del database.

1. Individuare il pacchetto **db-adapter.zip** nella cartella **C:\hp\UCMDB\UCMDBServer\content\adapters**.
2. Estrarre il pacchetto in una directory temporanea locale.
3. Modificare il file XML dell'adattatore:
   - ➤ Aprire il file **discoveryPatterns\db_adapter.xml** in un editor di testo.
   - ➤ Individuare l'attributo **adapter id** e sostituire il nome:

```
<pattern id="MyAdapter" displayLabel="My Adapter"
xsi:noNamespaceSchemaLocation="../../Patterns.xsd" description="Discovery
Pattern Description"
        schemaVersion="9.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" displayName="UCMDB API Population">
```

Se l'adattatore supporta i dati di replica, la funzione seguente deve essere aggiunta all'elemento <**adapter-capabilities**>:

```
<support-replicatioin-data>
    <source>
        <changes-source/>
    </source>
</support-replicatioin-data>
```

L'etichetta visualizzata o l'ID viene visualizzato nell'elenco degli adattatori nel riquadro Punti integrazione in HP Universal CMDB.

Per i dettagli sul popolamento del CMDB con i dati, consultare "Pagina Studio di integrazione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

➤ Se l'adattatore utilizza il motore di mappatura della versione 8.x (ovvero non utilizza il nuovo motore di mappatura della riconciliazione), sostituire l'elemento seguente:

```
<default-mapping-engine/>
```

con

```
<default-mapping-engine>com.hp.ucmdb.federation.mappingEngine.AdapterMa
ppingEngine</default-mapping-engine>
```

Per invertire il nuovo motore di mappatura, riportare l'elemento al valore seguente:

```
<default-mapping-engine/>
```

➤ Individuare la definizione **category**.

```
<category>Generic</category>
```

Cambiare il nome della categoria **Generic** nella categoria di propria scelta.

Nota: gli adattatori le cui categorie sono specificate come **Generic** non sono elencate in Studio integrazione quando si crea un nuovo punto di integrazione.

---

4 Nella directory temporanea aprire la cartella **adapterCode** e rinominare **GenericDBAdapter** con il valore di **adapter id** utilizzato nel passaggio 3.

Questa cartella contiene i file jar che eseguono la logica della federazione, ad esempio il nome dell'adattatore, le query e le classi nel CMDB, e i campi del RDBMS supportati dall'adattatore.

5 Configurare l'adattatore come necessario. Per i dettagli consultare "Configurare l'adattatore" a pag. 143.

6 Creare un file *.zip con lo stesso nome dato all'attributo **adapter id** come descritto nel passaggio 3 a pag. 140.

Nota: il file **descriptor.xml** è un file predefinito che esiste in ogni pacchetto.

---

7 Salvare il nuovo pacchetto creato nel passaggio precedente. La directory predefinita per gli adattatori è: **C:\hp\UCMDB\UCMDBServer\content\adapters**.

Eseguire l'aggiornamento dell'adattatore generico del database da 9.00 o 9.01 a 9.02 e successivo

1 Estrarre il pacchetto dell'adattatore in una directory temporanea locale.

2 Estrarre i file.

3 Rimuovere le righe seguenti dalla cartella **adapterCode\<nome adattatore>**:

- **asm.jar**
- **asm-attrs.jar**
- **cglib.jar**
- **db-adapter.jar**
- **jboss-archive-browsing.jar**
- **saxon-b.jar**

4 Ricreare il pacchetto dell'adattatore.

Nota: per ogni adattatore generico del database distribuito, il programma di installazione di UCMDB rimuove i file necessari da UCMDB e dal file system della sonda. Sarà comunque necessario preparare il pacchetto per ridistribuirlo quando necessario.

Configurare l'adattatore

È possibile utilizzare uno dei metodi seguenti per configurare l'adattatore:

➤ "Configurazione dell'adattatore - Metodo minimale" a pag. 143

➤ "Configurazione dell'adattatore - Metodo avanzato" a pag. 146

Questi file di configurazione si trovano nel pacchetto **db-adapter.zip** nella cartella **C:\hp\UCMDB\UCMDBServer\content\adapters** estratta nel passaggio 2 di "Preparare il pacchetto dell'adattatore" a pag. 140.

Configurazione dell'adattatore - Metodo minimale

Nota: il file **orm.xml** che viene generato automaticamente in seguito all'esecuzione di questo metodo è un ottimo esempio da utilizzare quando si impiega il metodo avanzato.

La procedura seguente descrive un metodo di mappatura del modello di classe nel CMDB a un RDBMS. È possibile utilizzare il metodo minimale quando è necessario:

➤ Federare un solo nodo come attributo del nodo.

➤ Dimostrare le funzioni dell'adattatore generico del database.

Questo metodo:

➤ supporta soltanto la federazione one-node

➤ supporta soltanto le relazioni virtuali many-to-one

Questo compito include i passaggi seguenti:

➤ "Configurare il file adapter.conf" a pag. 144

➤ "Configurare il file simplifiedConfiguration.xml" a pag. 144

Configurare il file adapter.conf

In questo passaggio, cambiare le impostazioni nel file adapter.conf in modo da federare i dati automaticamente.

1 Aprire il file **adapter.conf** in un editor di testo.

2 Individuare la riga seguente: **use.simplified.xml.config=<true/false>**.

3 Cambiarla in **use.simplified.xml.config=true**.

Configurare il file simplifiedConfiguration.xml

In questo passaggio configurare il file **simplifiedConfiguration.xml** mappando il CIT del CMDB ai campi della tabella del RDBMS.

1 Aprire il file **simplifiedConfiguration.xml** in un editor di testo.

Questo file include un esemplare che si può utilizzare per ciascuna entità da mappare.

Nota: non modificare il file **simplifiedConfiguration.xml** file in nessuna versione di Notepad di Microsoft Corporation. Utilizzare Notepad++, UltraEdit o altri editor di testo di terze parti.

---

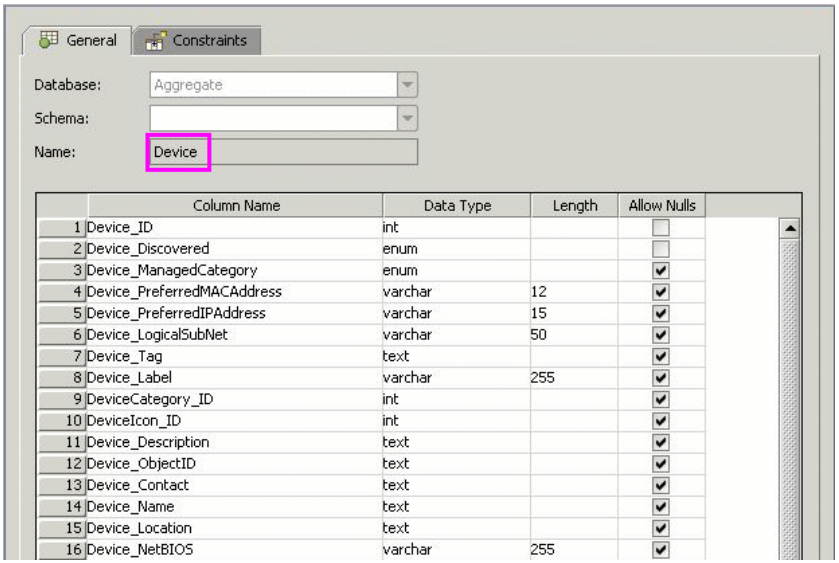
2 Apportare modifiche agli attributi seguenti:

➤ Il nome del CIT in UCMDB (cmdb-class-name) e il nome della tabella corrispondente nel RDBMS (default-table-name):

```
<cmdb-class cmdb-class-name="node" default-table-name="Device">
```

L'attributo cmdb-class-name viene preso dal CIT del nodo:

L'attributo default-table-name viene preso dalla tabella Device:



	Column Name	Data Type	Length	Allow Nulls
1	Device_ID	int		☐
2	Device_Discovered	enum		☐
3	Device_ManagedCategory	enum		☑
4	Device_PreferredMACAddress	varchar	12	☑
5	Device_PreferredIPAddress	varchar	15	☑
6	Device_LogicalSubNet	varchar	50	☑
7	Device_Tag	text		☑
8	Device_Label	varchar	255	☑
9	DeviceCategory_ID	int		☑
10	DeviceIcon_ID	int		☑
11	Device_Description	text		☑
12	Device_ObjectID	text		☑
13	Device_Contact	text		☑
14	Device_Name	text		☑
15	Device_Location	text		☑
16	Device_NetBIOS	varchar	255	☑

- L'identificatore univoco nel RDBMS:

```
<primary-key column-name="Device_ID"/>
```

- La regola di riconciliazione (reconciliation-by-two-nodes):

```
<reconciliation-by-two-nodes connected-node-cmdb-class-name="ip_address"
cmdb-link-type="containment">
```

- L'attributo di riconciliazione in UCMDB (cmdb-attribute-name) e nel RDBMS (column-name):

```
<connected-node-attribute cmdb-attribute-name="name"
column-name="[column_name]"/>
```

- Il nome del CIT in (cmdb-class-name) e il nome della tabella corrispondente nel RDBMS (default-table-name): anche la relazione CMDB (connected-cmdb-class-name) e la relazione CIT (link-class-name):

```
<class cmdb-class-name="sw_sub_component"
default-table-name="SWSubComponent" connected-cmdb-class-name="node"
link-class-name="composition">
```

- La chiave primaria e la chiave esterna:

```
<foreign-primary-key column-name="Device_ID"
cmdb-class-primary-key-column="Device_ID"/>
```

- L'identificatore univoco nel RDBMS:

```
<primary-key column-name="Device_ID"/>
```

- La mappatura tra l'attributo di CMDB (cmdb-attribute-name) e il nome della colonna nel RDBMS (column-name):

```
<attribute cmdb-attribute-name="last_access_time"
column-name="SWSubComponent_LastAccess TimeStamp"/>
```

3 Salvare il file.

Configurazione dell'adattatore - Metodo avanzato

Questo compito include i passaggi seguenti:

- "Configurare il file orm.xml" a pag. 147
- "Configurare il file reconciliation_types.txt" a pag. 151
- "Configurare il file reconciliation_rules.txt" a pag. 151

Configurare il file orm.xml

In questo passaggio, mappare i CIT e le relazioni del CMDB alle tabelle del RDBMS.

1 Aprire il file **orm.xml** in un editor di testo.

Questo file, per impostazione predefinita, contiene un esemplare da utilizzare per mappare tutti i CIT e le relazioni necessari per la federazione.

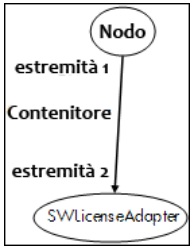
Nota: non modificare il file **orm.xml** file in nessuna versione di Notepad di Microsoft Corporation. Utilizzare Notepad++, UltraEdit o altri editor di testo di terze parti.

---

2 Apportare le modifiche al file in base alle entità dei dati da mappare. Per i dettagli consultare gli esempi seguenti.

I tipi seguenti di relazioni possono essere mappati nel file **orm.xml**:

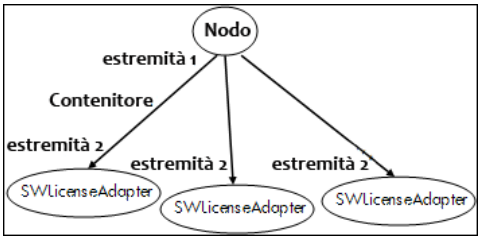
➤ One to one:



Il codice per questo tipo di relazione è:

```
<one-to-one name="end1" target-entity="node">
        <join-column name=“Device_ID" />
</one-to-one>
<one-to-one name="end2" target-entity=“sw_sub_component">
        <join-column name=“Device_ID" />
        <join-column name=“Version_ID" />
</one-to-one>
```

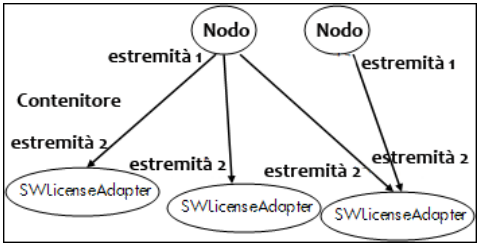
➤ Many to one:



Il codice per questo tipo di relazione è:

```
<many-to-one name="end1" target-entity="node">
        <join-column name="Device_ID" />
</many-to-one>
<one-to-one name="end2" target-entity="sw_sub_component">
        <join-column name="Device_ID" />
        <join-column name="Version_ID" />
</one-to-one>
```

➤ Many to many:



Il codice per questo tipo di relazione è:

```
<many-to-one name="end1" target-entity="node">
        <join-column name="Device_ID" />
</many-to-one>
<many-to-one name="end2" target-entity="sw_sub_component">
        <join-column name="Device_ID" />
        <join-column name="Version_ID" />
</many-to-one>
```

Per i dettagli sulla denominazione delle convenzioni, consultare "Convenzioni di denominazione" a pag. 186.

Esempio di mappatura di entità tra il modello di dati e il RDBMS:

Nota: gli attributi che non devono essere configurati sono omessi dagli esempi seguenti.

➤ La classe del CIT di CMDB:

```
<entity class="generic_db_adapter.node">
```

➤ Il nome della tabella del RDBMS:

```
<table name="Device"/>
```

➤ Il nome della colonna dell'identificatore univoco tabella del RDBMS:

```
<column name="Device ID"/>
```

➤ Il nome dell'attributo del CIT di CMDB:

```
<basic name="name">
```

➤ Il nome del campo della tabella dell'origine di dati esterna:

```
<column name="Device_Name"/>
```

➤ Il nome del nuovo CIT creato in "Creare un tipo di CI" a pag. 137:

```
<entity class="generic_db_adapter.MyAdapter">
```

➤ Il nome della tabella corrispondente del RDBMS:

```
<table name="SW_License"/>
```

➤ L'identità univoca nel RDBMS:

```
<id name="id1">
        <column updatable="false" insertable="false" name="Device_ID"/>
        <generated-value strategy="TABLE"/>
</id>
<id name="id2">
        <column updatable="false" insertable="false" name="Version_ID"/>
        <generated-value strategy="TABLE"/>
</id>
```

➤ Il nome dell'attributo nel CIT di CMDB e il nome dell'attributo corrispondente nel RDBMS:

```
<basic name="license_required">
        <column updatable="false" insertable="false"
name="MyAdapter_LicenseRequired"/>
```

Esempio di mappatura di relazione tra il modello di dati e il RDBMS:

➤ La classe della relazione di CMDB:

```
<entity class="generic_db_adapter.node_containment_MyAdapter">
```

➤ Il nome della tabella del RDBMS dove viene eseguita la relazione:

```
<table name="MyAdapter"/>
```

➤ L'ID univoco nel RDBMS:

```
<id name="id1">
        <column updatable="false" insertable="false" name="Device_ID"/>
        <generated-value strategy="TABLE"/>
</id>
<id name="id2">
        <column updatable="false" insertable="false" name="Version_ID"/>
        <generated-value strategy="TABLE"/>
</id>
```

➤ Il tipo di relazione e il CIT di CMDB:

```
<many-to-one target-entity="node" name="end1">
```

➤ I campi della chiave primaria e della chiave esterna nel RDBMS:

```
<join-column updatable="false" insertable="false"
referenced-column-name="[column_name]" name="Device_ID"/>
```

Configurare il file reconciliation_types.txt

Aprire il file **reconciliation_types.txt** in un editor di testo.

Per i dettagli consultare "File reconciliation_types.txt" a pag. 193.

Configurare il file reconciliation_rules.txt

In questo passaggio definire le regole in base alle quali l'adattatore riconcilia il CMDB e il RDBMS (solo se viene utilizzato il motore di mappatura per la compatibilità inversa con la versione 8.x):

1. Aprire il file **META-INF\reconciliation_rules.txt** in un editor di testo.
2. Apportare le modifiche al file in base al CIT che si sta mappando. Ad esempio per mappare un CIT di un nodo utilizzare l'espressione seguente:

```
multinode[node] ordered expression[^name]
```

Nota:

- Se i dati del database sono sensibili alle maiuscole/minuscole, non eliminare il carattere di controllo (^).
- Verificare che ciascuna parentesi quadra di apertura abbia una parentesi di chiusura corrispondente.

Per i dettagli consultare "File reconciliation_rules.txt (per la contabilità inversa)" a pag. 194.

Implementare un plug-in

Questo compito descrive come implementare e distribuire un adattatore generico di database con i plug-in.

Nota: prima di scrivere un plug-in per un adattatore, accertarsi di aver completato tutti i passaggi necessari in "Preparare il pacchetto dell'adattatore" a pag. 140.

---

1 Copiare i file jar seguenti dalla directory di installazione del server UCMDB nel percorso della classe di sviluppo:

- Copiare i file **db-interfaces.jar** e **db-interfaces-javadoc.jar** dalla cartella **tools\adapter-dev-kit**.
- Copiare i file **federation-api.jar** e **federation-api-javadoc.jar** dalla cartella **\tools\adapter-dev-kit\SampleAdapters\production-lib**.

Nota: per maggiori informazioni sullo sviluppo di un plug-in consultare i file **db-interfaces-javadoc.jar** e **federation-api-javadoc.jar** e la documentazione online all'indirizzo:

- **C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\doc_lib\DevRef_guide\DBAdapterFramework_JavaAPI\index.html**
- **C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\doc_lib\DevRef_guide\Federation_JavaAPI\index.html**

---

2 Scrivere una classe Java per l'implementazione dell'interfaccia Java del plug-in. Le interfacce sono definite nel file **db-interfaces.jar**. La tabella sottostante specifica l'interfaccia da implementare per ciascun plug-in:

Tipo di plug-in	Nome interfaccia	Metodo
Sincronizzazione topologia completa	FcmdbPluginForSyncGetFull Topology	getFullTopology
Sincronizzazione cambiamenti	FcmdbPluginForSyncGet ChangesTopology	getChangesTopology
Sincronizzazione layout	FcmdbPluginForSyncGet Layout	getLayout
Recupero query supportate	FcmdbPluginForSyncGet SupportedQueries	getSupportedQueries
Modifica definizione query TQL e risultati	FcmdbPluginGetTopology CmdbFormat	getTopologyCmdbFormat
Modifica richiesta layout per CI	FcmdbPluginGetCIsLayout	getCisLayout
Modifica richiesta layout per collegamenti	FcmdbPluginGetRelations Layout	getRelationsLayout

La classe del plug-in deve avere un costruttore pubblico predefinito. Inoltre, tutte le interfacce presentano un metodo denominato initPlugin. È assodato che questo metodo viene chiamato prima di qualsiasi altro metodo e viene utilizzato per inizializzare l'adattatore con l'oggetto dell'ambiente dell'adattatore che lo contiene.

3 Accertarsi che i file Federation SDK JAR e Generic DB Adapter JAR si trovino nel percorso della classe prima di compilare il codice Java. Il Federation SDK è il file **federation_api.jar** che si trova nella directory **C:\hp\UCMDB\UCMDBServer\lib**.

4 Includere la classe in un file jar e posizionarlo nella cartella adapterCode\<proprionome> nel pacchetto dell'adattatore prima di distribuirlo.

I plug-in sono configurati mediante il file **plugins.txt** che si trova nella cartella **META-INF** dell'adattatore.

Di seguito viene riportato un esempio di un file dell'adattatore DDMi:

```
# mandatory plugin to sync full topology
[getFullTopology]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin

# mandatory plugin to sync changes in topology
[getChangesTopology]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin

# mandatory plugin to sync layout
[getLayout]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin

# plugin to get supported queries in sync. If not defined return all tqls names
[getSupportedQueries]

# internal not mandatory plugin to change tql definition and tql result
[getTopologyCmdbFormat]

# internal not mandatory plugin to change layout request and CIs result
[getCisLayout]

# internal not mandatory plugin to change layout request and relations result
[getRelationsLayout]
```

Legenda:

- Riga di commento.

[<tipo di adattatore>] - avvio della sezione della definizione di un tipo di adattatore specifico.

Ci può essere una riga vuota sotto ogni [<tipo di adattatore>], ovvero non sono presenti classi plug-in associate oppure è possibile elencare il nome completo della classe plugin.

5 Includere l'adattatore con il nuovo file jar e il file **plugins.xml** aggiornato. ll resto dei file del pacchetto deve essere lo stesso in ogni adattatore in base all'adattatore generico del database.

Distribuire l'adattatore

1 In UCMDB accedere a Gestione pacchetti. Per i dettagli consultare "Pagina Gestione pacchetti" nella *Guida all'amministrazione di HP Universal CMDB*.

2 Fare clic sull'icona **Distribuisci pacchetti sul server (dal disco locale)** e passare al pacchetto dell'adattatore. Selezionare il pacchetto e fare clic su **Apri**, quindi fare clic su **Distribuisci** per visualizzare il pacchetto in Gestione pacchetti.

3 Selezionare il pacchetto nell'elenco e fare clic sull'icona **Visualizza risorse pacchetto** per verificare che i contenuti del pacchetto sono riconosciuti da Gestione pacchetti.

Modificare l'adattatore

Dopo aver creato e distribuito l'adattatore è possibile modificarlo in UCMDB. Per i dettagli consultare "Gestione adattatore" a pag. 113.

Creare un punto di integrazione

In questo passaggio verificare che la federazione sia attiva ovvero che la connessione sia valida e che il file XML sia valido. Questo controllo comunque non verifica che il file XML sia mappato ai campi corretti del RDBMS.

1 In UCMDB, accedere a Studio di integrazione (**Gestione flusso dati > Studio di integrazione**).

2 Creare un punto di integrazione. Per i dettagli consultare "Finestra di dialogo Crea nuovo punto di integrazione/Modifica punto di integrazione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

La scheda Federazione visualizza tutti i CIT che si possono federare utilizzando questo punto di integrazione. Per i dettagli consultare "Scheda Federazione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

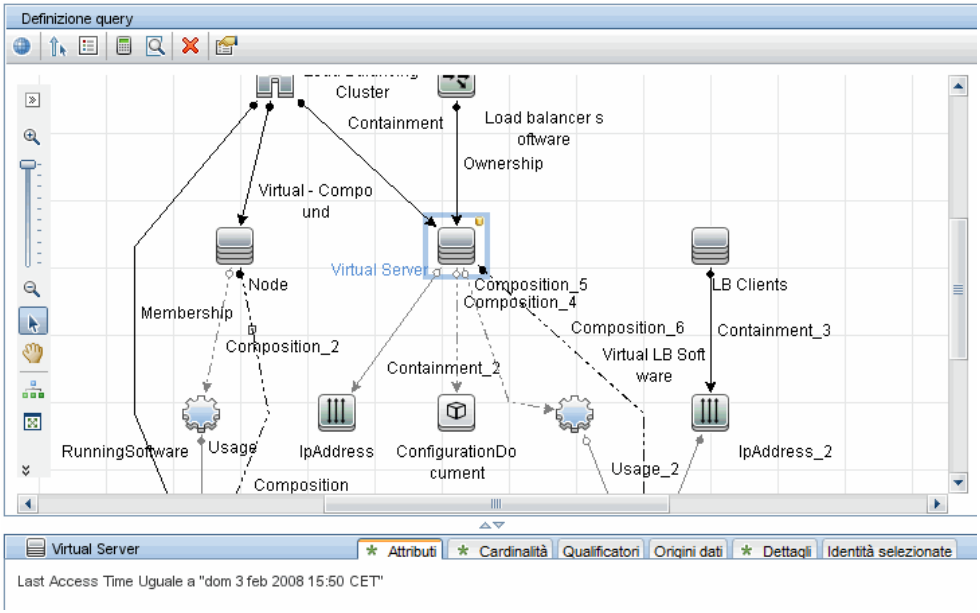
Creare una vista

In questo passaggio creare una vista che consenta di visualizzare le istanze del CIT.

1 In UCMDB, accedere a Studio di modellazione (**Modellazione** > **Studio di modellazione**).

2 Creare una vista. Per i dettagli consultare "Creare un vista basata sull'esemplare" nella *Guida alla modellazione di HP Universal CMDB*.

3 È possibile aggiungere condizioni alla TQL, ad esempio l'ultima ora di accesso risale a oltre sei mesi fa:



Calcolare i risultati

In questo passaggio si verificano i risultati.

1. In UCMDB, accedere a Studio di modellazione (**Modellazione** > **Studio di modellazione**).
2. Aprire una vista.
3. Per calcolare i risultati fare clic sul pulsante **Calcola conteggio risultati query**.
4.  Fare clic su **Anteprima** per visualizzare i CI nella vista.

Visualizzare i risultati

In questo passaggio si possono visualizzare i risultati e i problemi di debug nella procedura. Ad esempio, se non viene visualizzato nulla nella vista, verificare le definizioni nel file **orm.xml**; rimuovere gli attributi della relazione e ricaricare l'adattatore.

1. In UCMDB, accedere a Gestione universo IT (**Modellazione** > **Gestione Universo IT**).
2. Selezionare un CI.

   Nella scheda Proprietà vengono visualizzati i risultati della federazione.

Visualizzare report

In questo passaggio vengono visualizzati i report Topologia. Per i dettagli consultare "Report Topologia - Panoramica" nella *Guida alla modellazione di HP Universal CMDB*.

Abilitare i file di registro

Per comprendere i flussi di calcolo, il ciclo di vita dell'adattatore e visualizzare le informazioni di debug è possibile consultare i file di registro. Per i dettagli consultare "File di registro dell'adattatore" a pag. 216.

Utilizzare Eclipse per eseguire la mappatura tra gli attributi dei CIT e le tabelle del database

---

Attenzione: questa procedura si rivolge a utenti con una conoscenza avanzata della distribuzione di contenuto. Per eventuali domande rivolgersi a HP Software Support.

Questo compito descrive come installare e utilizzare il plug-in JPA fornito con l'edizione J2EE di Eclipse per le operazioni seguenti:

- Consentire la mappatura grafica tra classi di attributi di CMDB e colonne di tabella del database.
- Abilitare la modifica manuale del file di mappatura (orm.xml) garantendo la correttezza. La verifica di correttezza include la verifica della sintassi e il controllo che gli attributi delle classi e le colonne di tabella del database siano indicati correttamente.
- Abilitare la distribuzione del file di mappatura sul server CMDB e la visualizzazione degli errori come ulteriore controllo di correttezza.
- Definire una query di esempio sul server CMDB ed eseguirla direttamente da Eclipse per verificare il file di mappatura.

Questo compito comprende i passaggi seguenti:

1 Prerequisiti

Installare **Java Runtime Environment (JRE) *6* Update *7*** sul computer sul quale viene eseguito Eclipse dal sito seguente: **http://java.sun.com/javase/downloads/index.jsp**.

La procedura funziona con ambiente di runtime Java 5 (o versione successiva).

2 Installazione

a Scaricare ed estrarre **Eclipse IDE for Java EE Developers** da **<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/ganymede/SR1/eclipse-jee-ganymede-SR1-win32.zip>** in una cartella locale, ad esempio **C:\Program Files\eclipse**.

b Copiare **com.hp.plugin.import_cmdb_model_1.0.jar** da **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\bin** a **C:\Program Files\Eclipse\plugins**.

c Avviare **C:\Program Files\Eclipse\eclipse.exe** (richiede almeno un ambiente di runtime Java 5). Se viene visualizzato un messaggio che comunica che la macchina virtuale non è stata trovata, avviare **eclipse.exe** con la riga di comando seguente:

```
"C:\Program Files\eclipse\eclipse.exe" -vm "<JRE installation folder>\bin"
```

3 Preparare l'ambiente di lavoro

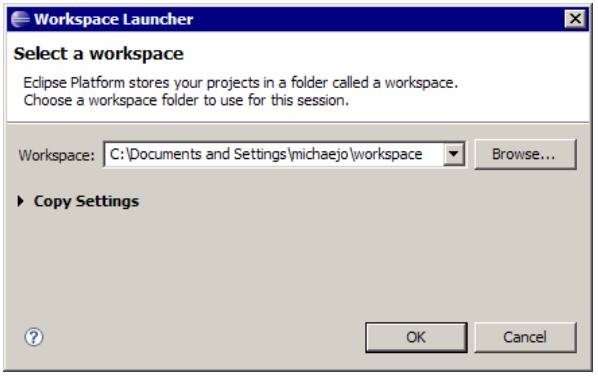
In questo passaggio viene impostato lo spazio di lavoro, il database, le connessioni e le proprietà dei driver.

a Estrarre il file **workspaces_gdb.rar** da **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** in **C:\Documents and Settings\All Users\workspaces**.

Nota: è necessario utilizzare il percorso esatto della cartella. Se viene dezippato il file nel percorso errato o il file non viene dezippato, la procedura non potrà funzionare.

---

b In Eclipse, selezionare **File** > **Switch Workspace** > **Other**:



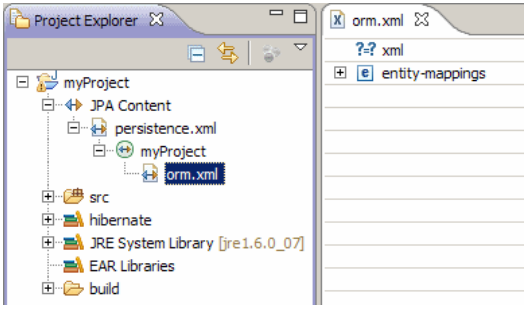
Se si utilizza:

➤ SQL Server, selezionare la cartella seguente: **C:\Documents and Settings\All Users\workspace_gdb_sqlserver**.

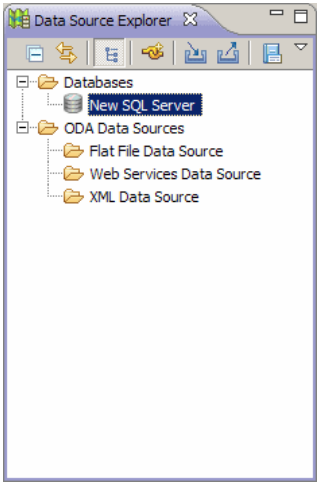
- MySQL, selezionare la cartella seguente: **C:\Documents and Settings\All Users\workspace_gdb_mysql**.
- Oracle, selezionare la cartella seguente: **C:\Documents and Settings\All Users\workspace_gdb_oracle**.

c Fare clic su **OK**.

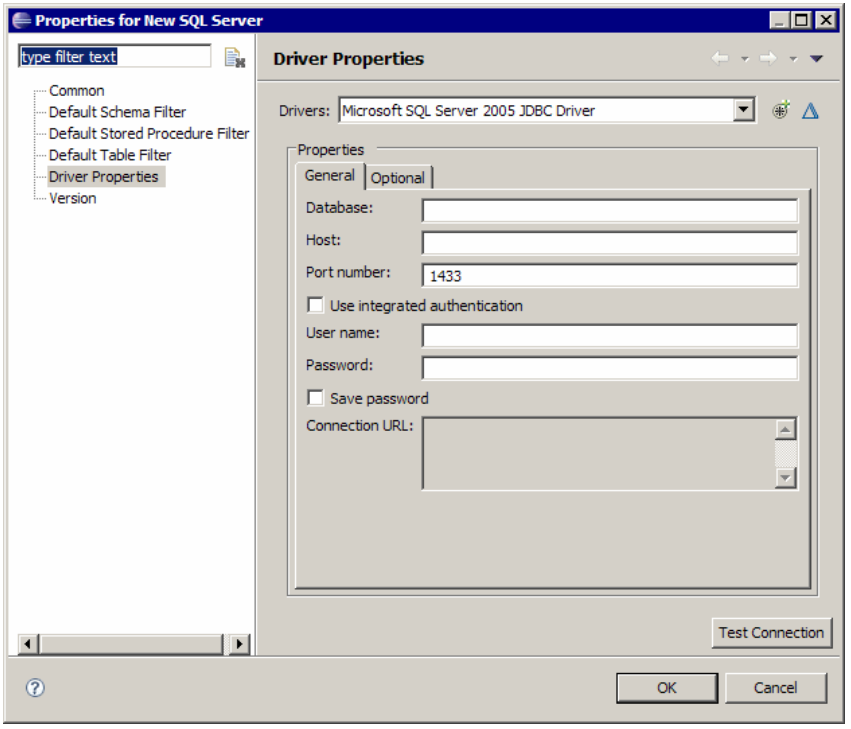
d In Eclipse, visualizzare la vista Project Explorer e selezionare **<Active project> > JPA Content > persistence.xml > <active project name> > orm.xml**.



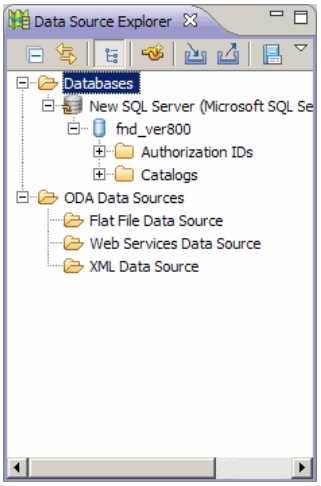
e Nella vista Data Source Explorer (riquadro in basso di sinistra), fare clic con il pulsante destro del mouse sulla connessione del database e selezionare il menu **Properties**.



f Nella finestra di dialogo **Properties for <Connection name>** selezionare **Common** e selezionare la casella di controllo **Connect every time the workbench is started**. Selezionare **Driver Properties** e immettere le proprietà di connessione. Fare clic su **Test Connection** e verificare che la connessione sia in essere. Fare clic su **OK**.



g Nella vista Data Source Explorer, fare clic con il pulsante destro del mouse sulla connessione del database e selezionare **Connect**. Sotto l'icona di connessione del database viene visualizzata una struttura contenente gli schemi e le tabelle del database.

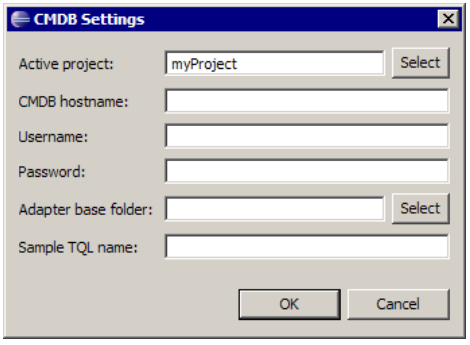


4 Creare un adattatore

Creare un adattatore utilizzando le linee guida indicate in "Passaggio 1: Creare un adattatore" a pag. 42.

5 Configurare il plug-in CMDB

a In Eclipse, fare clic su **UCMDB** > **Settings** per aprire la finestra di dialogo **CMDB Settings**:



b Se non è già selezionato, selezionare il progetto JPA appena creato come progetto attivo.

c Immettere il nome host CMDB, ad esempio **localhost** oppure **labm1.itdep1**. Non è necessario includere il numero di porta o il prefisso http:// all'indirizzo.

d Immettere nome utente e la password per accedere all'API di CMDB, di norma sono **admin**/**admin**.

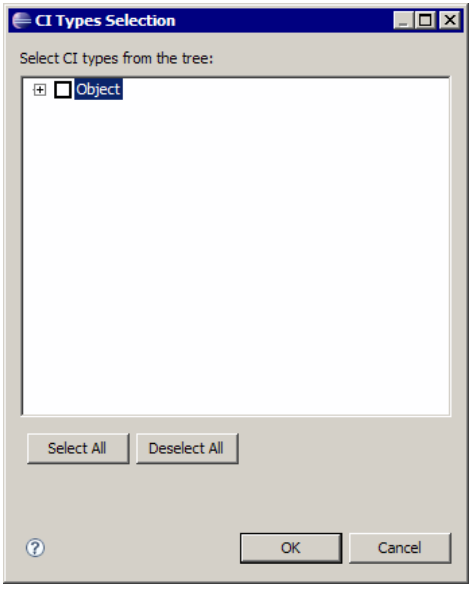
e Accertarsi che la cartella **C:\hp** sul server CMDB sia mappata come unità di rete.

f Selezionare la cartella di base dell'adattatore rilevante in **C:\hp**. La cartella di base è quella che contiene il file **dbAdapter.jar** e la sottocartella **META-INF**. Il percorso dovrebbe essere **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\<nome adattatore>**. Accertarsi che non vi siano barre rovesciate alla fine (\).

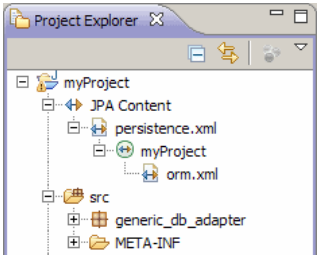
6 Importare il modello di classe di UCMDB

In questo passaggio, selezionare il CIT da mappare come entità JPA.

a Fare clic su **UCMDB** > **Import CMDB Class Model** per aprire la finestra di dialogo **CI Type Selection**:



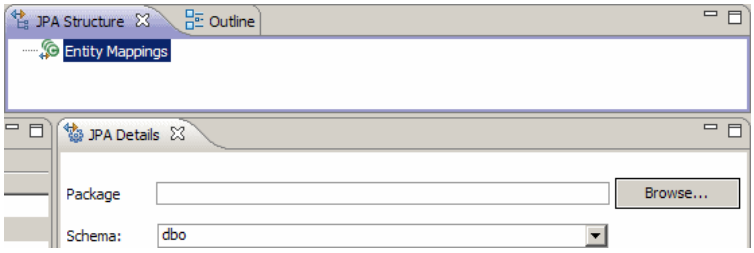
b Selezionare i tipi CI che si desidera mappare come entità JPA. Fare clic su **OK**. I tipi CI vengono importati come classi Java. Verificare che siano visualizzati nella cartella **src** del progetto attivo:



7 Costruire il file ORM File - Mappare le classi di UCMDB alle tabelle del database

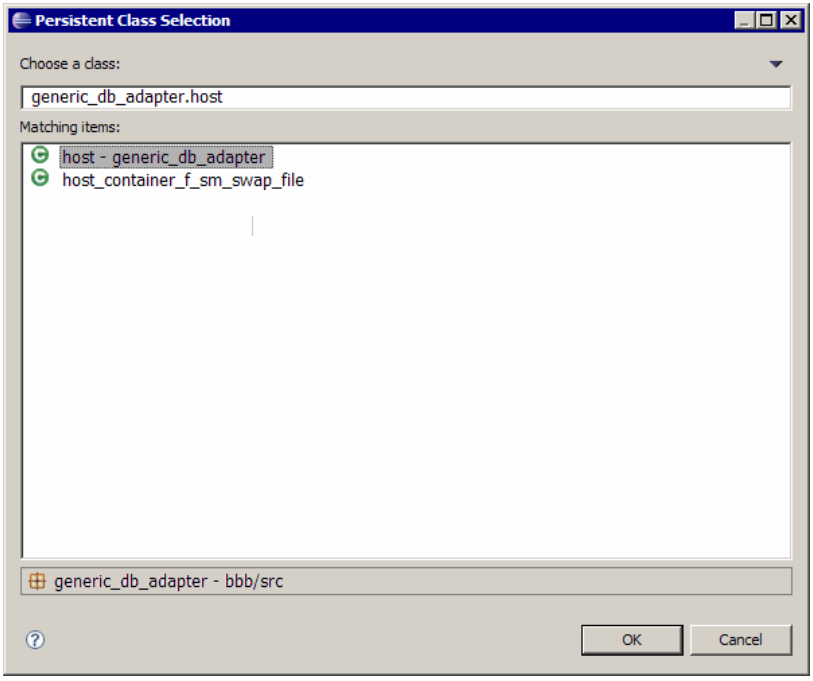
In questo passaggio vengono mappate le classi Java (importate nel passaggio precedente) alle tabelle del database.

a Accertarsi che la connessione del dabatase sia in corso. Fare clic con il pulsante destro del mouse sul progetto attivo (denominato myProject per impostazione predefinita) in Project Explorer. Selezionare la vista JPA, selezionare la casella di controllo **Override default schema from connection** e selezionare lo schema rilevante del database. Fare clic su **OK**.



b Mappare a un CIT: nella vista JPA Structure, fare clic con il pulsante destro del mouse sul ramo **Entity Mappings** e selezionare **Add Class**. Si apre la finestra di dialogo **Add Persistent Class**. Non cambiare il campo **Map as** (**Entity**).

c Fare clic su **Browse** e selezionare la classe di UCMDB da mappare (tutte le classi di UCMDB appartengono al pacchetto **generic_db_adapter**).

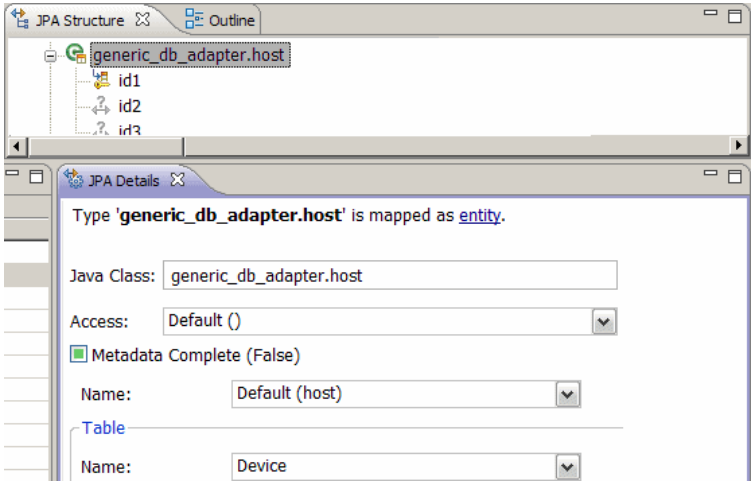


d Fare clic su **OK** nelle finestre di dialogo. La classe selezionata viene visualizzata sotto il ramo **Entity Mappings** nella vista JPA Structure.

Nota: se l'entità viene visualizzata senza una struttura dell'attributo, fare clic con il pulsante destro del mouse sul progetto attivo nella vista Project Explorer. Selezionare **Close** e quindi **Open**.

---

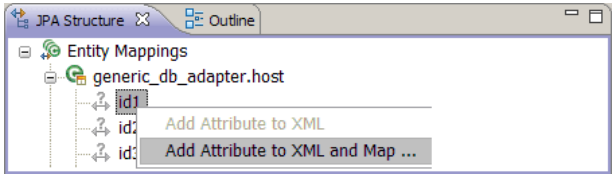
e Nella vista JPA Details, selezionare la tabella primaria del database alla quale deve essere mappata la classe di UCMDB. Lasciare invariati tutti gli altri campi.



8 Mappare gli ID

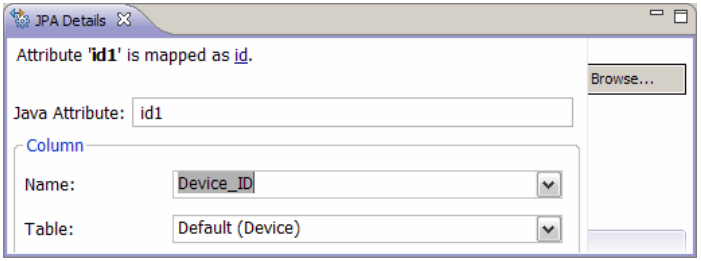
In base agli standard JPA, ciascuna classe persistente deve avere almeno un attributo dell'ID. Per le classi di UCMDB è possibile mappare fino a tre attributi come ID. I possibili attributi dell'ID sono denominati **id1**, **id2** e **id3**. Per mappare un attributo dell'ID:

a Espandere la classe corrispondente sotto il ramo **Entity Mappings** nella vista JPA Structure, fare clic con il pulsante destro del mouse sull'attributo rilevante (ad esempio **id1**) e selezionare **Add Attribute to XML and Map...**:



b Si apre la finestra di dialogo **Add Persistent Attribute**. Selezionare **Id** nel campo **Map as** e fare clic su **OK**.

c Nella vista JPA Details, selezionare la colonna di tabella del database alla quale deve essere mappato il campo dell'ID.



9 Mappare gli attributi

In questo passaggio si mappano gli attributi alle colonne del database.

a Espandere la classe corrispondente sotto il ramo **Entity Mappings** nella vista JPA Structure, fare clic con il pulsante destro del mouse sull'attributo rilevante (ad esempio **host_hostname**) e selezionare **Add Attribute to XML and Map...**:

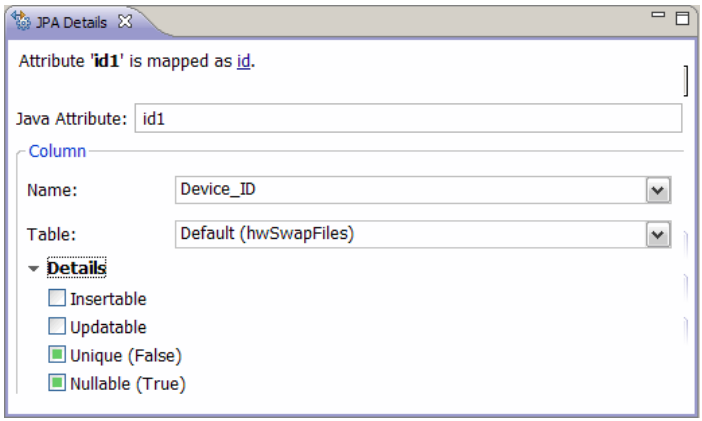
b Si apre la finestra di dialogo **Add Persistent Attribute**. Selezionare **Basic** nel campo **Map as** e fare clic su **OK**.

c Nella vista JPA Details, selezionare la colonna di tabella del database alla quale deve essere mappato il campo dell'attributo.

10 Mappare un collegamento valido

Eseguire i passaggi descritti al punto b a pag. 166 per la mappatura di una classe di UCMDB che denota un collegamento valido. Il nome di ognuna di tali classi assume la struttura seguente: **<end1 entity name>_<link name>_<end 2 entity name>**. Ad esempio un collegamento **Contains** tra un host e una posizione è denotato da una classe Java denominata **generic_db_adapter.host_contains_location**. Per i dettagli consultare "File reconciliation_rules.txt (per la contabilità inversa)" a pag. 194.

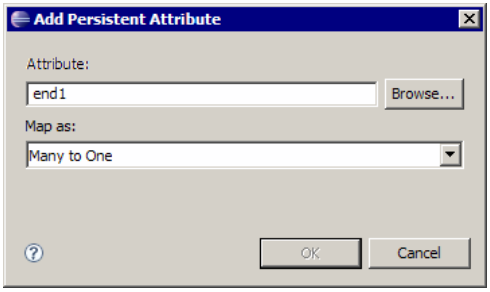
a Mappare gli attributi dell'ID della classe del collegamento come descritto in "Mappare gli ID" a pag. 168. Per ciascun attributo dell'ID espandere il gruppo della casella di controllo **Details** nella vista JPA Details e deselezionare le caselle di controllo **Insertable** e **Updateable**.



b Mappare gli attributi **end1** e **end2** della classe del collegamento come segue: per ciascuno degli attributi **end1** e **end2** della classe del collegamento:

➤ Espandere la classe corrispondente sotto il ramo **Entity Mappings** nella vista JPA Structure, fare clic con il pulsante destro del mouse sull'attributo rilevante (ad esempio **end1**) e selezionare **Add Attribute to XML and Map...**:

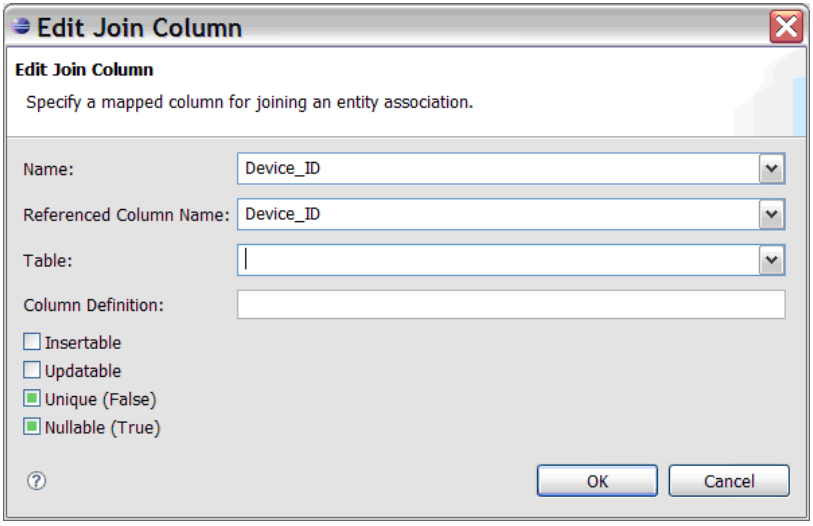
➤ Nella finestra di dialogo **Add Persistent Attribute** selezionare **Many to One** oppure **One to One** nel campo **Map as**.



➤ Selezionare **Many to One** se il CI **end1** oppure **end2** può avere più collegamenti di questo tipo. In caso contrario selezionare **One to One**. Ad esempio per un collegamento **host_contains_ip** l'estremità **host** dovrebbe essere mappata come **Many to One** poiché un host può avere più IP e l'estremità **ip** dovrebbe essere mappata come **One to One** poiché un IP può avere un solo host.

➤ Nella vista JPA Details selezionare **Target entity**, ad esempio **generic_db_adapter.host**.

➤ Nella sezione **Join Columns** della vista JPA Details fare clic su **Override Default**. Fare clic su **Modifica**. Nella finestra di dialogo **Edit Join Column** selezionare la colonna della chiave esterna della tabella del database di collegamento che punta a una voce della tabella dell'entità di destinazione **end1**/**end2**. Se il nome della colonna di riferimento nella tabella dell'entità di destinazione **end1**/**end2** è mappato al proprio attributo dell'ID, lasciare **Referenced Column Name** invariato. In caso contrario selezionare il nome della colonna al quale punta la colonna della chiave esterna. Deselezionare le caselle di controllo **Insertable** e **Updatable** e fare clic su **OK**.



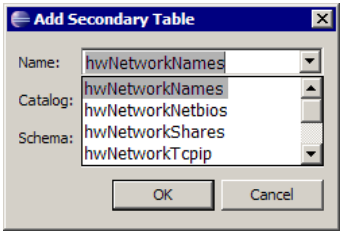
➤ Se l'entità di destinazione **end1**/**end2** ha più di un ID, fare clic sul pulsante **Add** per aggiungere altre colonne join e mapparle nello stesso modo descritto nel passaggio precedente.

11 Creare il file ORM File - Utilizzare le tabelle secondarie

JPA consente a una classe Java di essere mappata a più di una tabella del database. Ad esempio **Host** può essere mappato alla tabella **Device** per consentire la persistenza della maggior parte degli attributi e alla tabella **NetworkNames** per consentire la persistenza di **host_hostName**. In questo caso **Device** è la tabella primaria e **NetworkNames** è la tabella secondaria. È possibile definire un numero qualsiasi di tabelle secondarie. L'unica condizione è che vi sia una relazione one-to-one tra le voci delle tabelle primaria e secondaria.

12 Definire una tabella secondaria

Selezionare la classe adeguata nella vista JPA Structure. Nella vista **JPA Details** accedere alla sezione **Secondary Tables** e fare clic su **Add**. Nella finestra di dialogo **Add Secondary Table** selezionare la tabella secondaria adeguata. Lasciare invariati gli altri campi.



Se la tabella primaria e secondaria non ha le stesse chiavi primarie, configurare le colonne join nella sezione **Primary Key Join Columns** della vista **JPA Details**.

13 Mappare un attributo a una tabella secondaria

Mappare un attributo di classe a un campo di una tabella secondaria come segue:

a Mappare l'attributo come descritto in "Mappare gli attributi" a pag. 169.

b Nella sezione **Column** della vista JPA Details selezionare il nome della tabella secondaria nel campo **Table** per sostituire il valore predefinito.

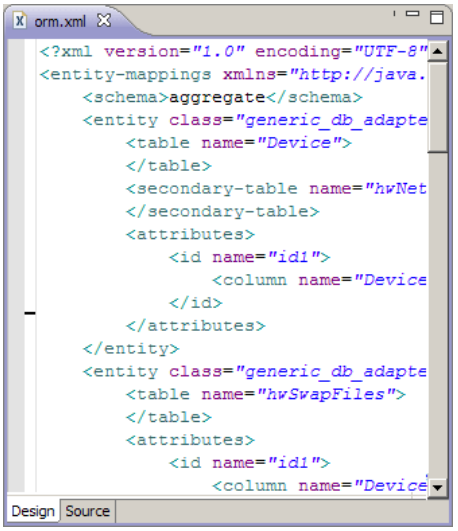
14 Utilizzare un file ORM esistente come base

Per utilizzare un file orm.xml esistente come base per quello che si sta sviluppando, eseguire i passaggi seguenti:

a Verificare che tutti i CIT mappati nel file **orm.xml** esistente siano importati nel progetto Eclipse attivo.

b Selezionare e copiare tutte o parte delle mappature dell'entità dal file esistente.

c Selezionare la scheda **Source** del file **orm.xml** nella prospettiva eclipse JPA.



d Incollare tutte le mappature di le entità copiate nel tag **<entity-mappings>** del file **orm.xml** modificato sotto il tag **<schema>**. Accertarsi che il tag dello schema sia configurato come descritto nel passaggio b a pag. 166. Tutte le entità incollate ora sono visualizzate nella vista JPA Structure. D'ora in poi le mappature possono essere modificate graficamente e manualmente con il codice xml del file **orm.xml**.

e Fare clic su **Salva**.

15 Importazione di un file ORM esistente da un adattatore

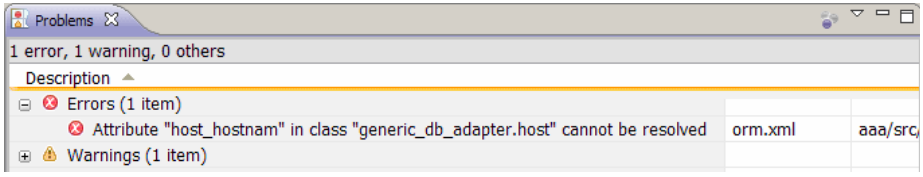
Se esiste già un adattatore, il plug-in Eclipse può essere utilizzato per modificare graficamente il file ORM. Importare il file ORM in Eclipse, modificarlo utilizzando il plug-in e ridistribuirlo sul computer di UCMDB. Per importare il file ORM premere il pulsante sulla barra degli strumenti di Eclipse. Viene visualizzata una finestra di dialogo di conferma. Fare clic su **OK**. Il file ORM viene copiato dal computer di UCMDB nel progetto attivo Eclipse e tutte le classi rilevanti vengono importate dal modello di classe UCMDB.

Se le classi rilevanti non vengono visualizzate nella vista JPA Structure, fare clic con il pulsante destro del mouse sul progetto attivo nella vista Project Explorer, selezionare **Close** quindi **Open**.

D'ora in poi è possibile modificare graficamente il file ORM utilizzando Eclipse e ridistribuirlo sul computer di UCMDB come descritto in "Distribuire il file ORM sul CMDB" a pag. 176.

16 Verificare la correttezza del file ORM File - Verifica correttezza incorporata

Il plug-in Eclipse JPA verifica la presenza di eventuali errori e li contrassegna nel file **orm.xml**. Vengono verificati errori di sintassi (ad esempio il nome errato del tag, un tag non chiuso, un ID mancante) e di mappatura (ad esempio il nome errato dell'attributo o del campo della tabella del database). In caso di errori ne viene visualizzata la descrizione nella vista **Problems**:



17 Creare un nuovo punto di integrazione

Se non esiste alcun punto di integrazione nel CMDB per questo adattatore, è possibile crearlo in Studio di integrazione. Per i dettagli consultare "Studio di integrazione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

Immettere il nome del punto di integrazione nella finestra di dialogo che si apre. Il file **orm.xml** viene copiato nella cartella dell'adattatore. Viene creato un punto di integrazione con tutti i tipi CI importati come classi supportate, salvo i CIT multinodo se sono configurati nel file reconciliation_rules.txt. Per i dettagli consultare "File reconciliation_rules.txt (per la contabilità inversa)" a pag. 194.

18 Distribuire il file ORM sul CMDB

Salvare il file **orm.xml** e distribuirlo sul server UCMDB: fare clic su **UCMDB** > **Deploy ORM**. Il file **orm.xml** viene copiato nella cartella dell'adattatore e l'adattatore viene ricaricato. Il risultato dell'operazione viene visualizzato in una finestra di dialogo **Operation Result**. Se si verificano errori durante il processo di ricaricamento, l'analisi dello stack delle eccezioni Java viene visualizzato nella finestra di dialogo. Se non è stato ancora definito alcun punto di integrazione utilizzando l'adattatore, non viene rilevato alcun errore di mappatura al momento della distribuzione.

19 Eseguire una query TQL di esempio

a Definire una query utilizzando Gestione query e non Gestione viste.

b Creare un punto di integrazione utilizzando l'adattatore **GenericDBAdapter**. Per i dettagli consultare "Finestra di dialogo Crea nuovo punto di integrazione/Modifica punto di integrazione" nella *Guida alla gestione del flusso di dati di HP Universal CMDB*.

c Durante la creazione dell'adattatore, verificare che i tipi CI che dovrebbero partecipare alla query siano supportati da questo punto di integrazione.

d Quando si configura il plug-in di CMDB utilizzare questo nome di query di esempio nella finestra di dialogo Impostazioni. Per i dettagli consultare "Configurare il plug-in CMDB" a pag. 163.

e Fare clic sul pulsante **Esegui TQL** per eseguire una TQL di esempio e verificare se restituisce i risultati richiesti utilizzando il file **orm.xml** appena creato.

Riferimenti

File di configurazione dell'adattatore.

I file esaminati in questa sezione si trovano nel pacchetto **db-adapter.zip** nella cartella **C:\hp\UCMDB\UCMDBServer\content\adapters**.

In questa sezione vengono trattati i seguenti argomenti:

Configurazione generale

- **adapter.conf.** File di configurazione dell'adattatore. Per i dettagli consultare "File adapter.conf" a pag. 178.

Configurazione avanzata

- **orm.xml**. File di mappatura di tipo object-relational per mappare i CIT di CMDB e le tabelle del database. Per i dettagli consultare "File orm.xml" a pag. 182.
- **reconciliation_types.txt**. Contiene le regole utilizzate per configurare i tipi di riconciliazione. Per i dettagli consultare "File reconciliation_types.txt" a pag. 193.
- **reconciliation_rules.txt**. Contiene le regole di riconciliazione. Per i dettagli consultare "File reconciliation_rules.txt (per la contabilità inversa)" a pag. 194.
- **transformations.txt.** File delle trasformazioni per indicare i convertitori da applicare per convertire il valore del CMDB nel valore del database e viceversa. Per i dettagli consultare "File transformations.txt" a pag. 196.
- **Discriminator.properties.** Questo file mappa ciascun tipo di CI supportato in un elenco separato da virgola di possibili valori corrispondenti. Per i dettagli consultare "File discriminator.properties" a pag. 198.

- **Replication_config.txt.** Questo file contiene un elenco separato da virgola di tipi di CI e relazioni le cui condizioni delle proprietà sono supportate dal plug-in di replica. Per i dettagli consultare "File replication_config.txt" a pag. 200.
- **Fixed_values.txt.** Questo file consente di configurare valori fissi per determinati attributi di certi CIT. Per i dettagli consultare "File fixed_values.txt" a pag. 200.

Configurazione di Hibernate

- **persistence.xml.** Utilizzato per sostituire le configurazioni di Hibernate preimpostate. Per i dettagli consultare "File persistence.xml" a pag. 197.

Configurazione semplice

- **simplifiedConfiguration.xml**. File di configurazione che sostituisce **orm.xml**, **transformations.txt** e **reconciliation_rules.txt** con funzioni minori. Per i dettagli consultare "File simplifiedConfiguration.xml" a pag. 179.

File adapter.conf

Questo file contiene le impostazioni seguenti:

- **use.simplified.xml.config=false. true**: uses simplifiedConfiguration.xml.

Nota: l'utilizzo di questo file di configurazione comporta la sostituzione di orm.xml, transformations.txt e reconciliation_rules.txt con funzioni minori.

- **dal.ids.chunk.size=300.** Non cambiare questo valore.
- **dal.use.persistence.xml=false**. **true**: l'adattatore legge la configurazione di Hibernate da persistence.xml.

Nota: si sconsiglia di sostituire la configurazione di Hibernate.

File simplifiedConfiguration.xml

Questo file viene utilizzato per la mappatura semplice delle classi di UCMDB alle tabelle del database. Per accedere all'esemplare per la modifica del file passare a **Gestione adattatore** > **db-adapter** > **File di configurazione**.

In questa sezione vengono trattati i seguenti argomenti:

➤ "Esemplare del file simplifiedConfiguration.xml" a pag. 179

➤ "Limitazioni" a pag. 181

Esemplare del file simplifiedConfiguration.xml

La proprietà **CMDB-class-name** e il tipo multinodo (il nodo al quale si collegano i CIT federati nella TQL):

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="[table_name]">
        <primary-key column-name="[column_name]"/>
```

reconciliation-by-two-nodes. La riconciliazione viene eseguita utilizzando un nodo oppure due nodi. In questo esempio la riconciliazione utilizza due nodi.

connected-node-CMDB-class-name. Il secondo tipo di classe necessario nella TQL di riconciliazione.

CMDB-link-type. Il tipo di relazione necessario nella TQL di riconciliazione.

link-direction. Direzione della relazione nella TQL di riconciliazione (da node a ip_address oppure da ip_address a node):

```
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address"
CMDB-link-type="containment" link-direction="main-to-connected">
```

L'espressione di riconciliazione è sotto forma di OR e ciascun OR include AND.

is-ordered. Stabilisce se la riconciliazione è stata eseguita sotto forma ordinata oppure per normale confronto OR.

```
<or is-ordered="true">
```

Se la proprietà di riconciliazione viene recuperata dalla classe principale (multinodo) utilizzare il tag **attribute**, in caso contrario utilizzare il tag **connected-node-attribute**.

ignore-case. true: quando i dati del modello di classe di UCMDB viene confrontato con i dati del RDBMS, senza distinzione di maiuscole e minuscole:

```
<attribute CMDB-attribute-name="name"
column-name="[column_name]" ignore-case="true"/>
```

Il nome della colonna è il nome della colonna della chiave esterna (colonna con valori che puntano alla colonna della chiave primaria del multinodo).

Se la colonna della chiave primaria del multinodo è composta da diverse colonne, sono necessarie diverse colonne di chiave esterna, una per ogni colonna di chiave primaria.

```
<foreign-primary-key column-name="[column_name]"
CMDB-class-primary-key-column="[column_name]"/>
```

Se le colonne della chiave primaria sono poche, duplicare questa colonna.

```
<primary-key column-name="[column_name]"/>
```

Le proprietà **from-CMDB-converter** e **to-CMDB-converter** sono classi Java che implementano le interfacce seguenti:

- com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB
- com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB

Utilizzare questi convertitori se i valori nel CMDB e nel database non sono gli stessi. Ad esempio il nome del nodo nel CMDB ha il suffisso mer.com.

In questo esempio GenericEnumTransformer viene utilizzato per convertire l'enumeratore in base al file XML scritto tra parentesi (**generic-enum-transformer-example.xml**):

```
        <attribute CMDB-attribute-name="[CMDB_attribute_name]"
column-name="[column_name]"
from-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.i
mpl.GenericEnumTransformer(generic-enum-transformer-example.xml)"
to-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)"/>
        <attribute CMDB-attribute-name="[CMDB_attribute_name]"
column-name="[column_name]"/>
        <attribute CMDB-attribute-name="[CMDB_attribute_name]"
column-name="[column_name]"/>
    </class>
</generic-DB-adapter-config>
```

Limitazioni

- Si possono applicare per mappare query TQL di un solo nodo (nella sorgente del database). Ad esempio è possibile eseguire una query TQL node > ticket e ticket. Per portare la gerarchia dei nodi dal database, è necessario utilizzare il file **orm.xml** avanzato.
- Sono supportate soltanto le relazioni one-to-many. Ad esempio è possibile portare uno o più ticket su ciascun nodo. Non è possibile portare ticket che appartengono a più di un nodo.
- Non è possibile collegare la stessa classe a diversi tipi di CIT di CMDB. Ad esempio se si stabilisce che ticket è collegato a node, non è possibile collegarlo anche a application.

File orm.xml

Questo file viene utilizzato per la mappatura dei CIT di UCMDB alle tabelle del database.

Un esemplare da utilizzare per la creazione di un nuovo file si trova nella directory **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INF META-INF**.

Per modificare il file XML per un adattatore distribuito passare a **Gestione adattatore** > **db-adapter** > **File di configurazione**.

In questa sezione vengono trattati i seguenti argomenti:

Esemplare di file orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" version="1.0" xsi:schemaLocation="http://
java.sun.com/xml/ns/persistence/orm http://java.sun.com/xml/ns/persistence/
orm_1_0.xsd">
    <description>Generic DB adapter orm</description>
```

Non cambiare il nome del pacchetto.

```
    <package>generic_db_adapter</package>
```

entity. Nome del CIT di CMDB. Questa è l'entità del multinodo.

Accertarsi che **class** includa un prefisso **generic_db_adapter.**

```
<entity class="generic_db_adapter.node">
    <table name="[table_name]"/>
```

Utilizzare una tabella secondaria se l'entità è mappata a più di una tabella.

```
    <secondary-table name=""/>
    <attributes>
```

Per l'ereditarietà di una sola tabella con discriminatore utilizzare il codice seguente:

```
    <inheritance strategy="SINGLE_TABLE"/>
    <discriminator-value>node</discriminator-value>
    <discriminator-column name="[column_name]"/>
```

Gli attributi con tag **id** sono colonne della chiave primaria. Accertarsi che le convenzioni di denominazione per queste colonne della chiave primaria siano **idX** (id1, id2 e così via) dove **X** è l'indice della colonna nella chiave primaria.

```
        <id name="id1">
```

Cambiare soltanto il nome della colonna della chiave primaria.

```
            <column updatable="false" insertable="false" name="[column_name]"/>
            <generated-value strategy="TABLE"/>
        </id>
```

basic. Utilizzato solo per dichiarare gli attributi di CMDB. Accertarsi di modificare soltanto le proprietà **name** e **column_name**.

```
        <basic name="name">
            <column updatable="false" insertable="false" name="[column_name]"/>
        </basic>
```

Per l'ereditarietà di una sola tabella con discriminatore mappare le classi esistenti come segue:

```
    <entity name="[cmdb_class_name]" class="generic_db_adapter.nt" name="nt">

        <discriminator-value>nt</discriminator-value>
        <attributes/>
    </entity>
    <entity class="generic_db_adapter.unix" name="unix">
        <discriminator-value>unix</discriminator-value>
        <attributes/>
    </entity>
    <entity name="[CMDB_class_name]"
class="generic_db_adapter.[CMDB[cmdb_class_name]">
        <table name="[default_table_name]"/>
        <secondary-table name=""/>
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[column_name]"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column updatable="false" insertable="false" name="[column_name]"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <column updatable="false" insertable="false" name="[column_name]"/>
                <generated-value strategy="TABLE"/>
            </id>
```

L'esempio seguente mostra un nome di attributo di CMDB senza prefisso:

```
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_name]"/>
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_name]"/>
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_name]"/>
            </basic>
        </attributes>
    </entity>
```

Questa è un'entità della relazione. La convenzione di denominazione è **end1Type_linkType_end2Type**. In questo esempio **end1Type** è **node** e **linkType** è **composition**.

```
    <entity name="node_composition_[CMDB_class_name]"
class="generic_db_adapter.node_composition_[CMDB_class_name]">
        <table name="[default_table_name]"/>
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[column_name]"/>
                <generated-value strategy="TABLE"/>
            </id>
```

L'entità di destinazione è l'entità alla quale punta questa proprietà. In questo esempio **end1** è mappato all'entità **node**.

many-to-one. Molte relazioni possono essere collegate a un solo node.

join-column. La colonna che contiene gli ID di **end1** (ID dell'entità di destinazione).

referenced-column-name. Nome della colonna nell'entità di destinazione (**node**) che contiene gli ID utilizzati nella colonna join.

```
            <many-to-one target-entity="node" name="end1">
                <join-column updatable="false" insertable="false"
referenced-column-name="[column_name]" name="[column_name]"/>
            </many-to-one>
```

one-to-one. Una relazione può essere collegata a un **[CMDB_class_name]**.

```
            <one-to-one target-entity="[CMDB_class_name]" name="end2">
                <join-column updatable="false" insertable="false"
referenced-column-name="" name="[column_name]"/>
            </one-to-one>
        </attributes>
    </entity>
</entity-mappings>
```

File ORM multipli

Sono supportati file di mappatura multipli. Ciascun nome di file di mappatura deve terminare con **orm.xml**. Tutti i file di mappatura devono essere collocati sotto la cartella META-INF dell'adattatore.

Convenzioni di denominazione

- In ciascuna entità, la proprietà della classe deve corrispondere alla proprietà del nome con il prefisso di generic_db_adapter.
- Le colonne della chiave primaria devono prendere il nome nella forma **idX** dove **X** = **1**, **2**, ... deve seguire il numero delle chiavi primaria nella tabella.
- I nomi degli attributi devono corrispondere ai nomi degli attributi delle classi a seconda dei casi.
- Il nome della relazione prende la forma end1Type_linkType_end2Type.
- I CIT del CMDB che sono anche parole riservate in Java devono essere prefissati da **gdba_**. Ad esempio per il CIT di CMDB **goto**, l'entità ORM deve essere denominata **gdba_goto**.

Utilizzo delle dichiarazione SQL inline anziché dei nomi delle tabelle

È possibile mappare le entità alle clausole inline select anziché alle tabelle del database. Ciò equivale a definire una vista nel database e a mappare un'entità a questa vista. Ad esempio:

```
    <entity class="generic_db_adapter.node">
        <table name="(select d.id as id1, d.name as name , d.os as host_os from
Device d)"/>
```

In questo esempio gli attributi del nodo devono essere mappati alle colonne id1, name, and host_os, rather than id, name e os.

Si applicano le limitazioni seguenti:

- La dichiarazione SQL inline è disponibile solo quando si utilizza Hibernate come provider JPA.
- Le parentesi tonde della clausola select SQL inline sono obbligatorie.
- L'elemento **<schema>** non deve essere presente nel file **orm.xml**. Nel caso di Microsoft SQL Server 2005, ciò significa che tutti i nomi delle tabelle devono avere il prefisso dbo. invece di definirle globalmente con <schema>dbo</schema>.

Schema orm.xml

Nella tabella seguente vengono spiegati gli elementi comuni del file **orm.xml**. Lo schema completo si trova all'indirizzo http://java.sun.com/xml/ns/persistence/orm_1_0.xsd. L'elenco non è completo e spiega nel complesso il comportamento specifico dello standard Java Persistence API per l'adattatore generico del database.

Elemento		Attributi
Nome e percorso	**Descrizione**	
entity-mappings	L'elemento principale per il documento di mappatura dell'entità. Questo elemento deve essere esattamente lo stesso di quello indicato nei file di esempio dell'adattatore generico del database.	
description (entity-mappings)	Descrizione a testo libero del documento di mappatura dell'entità. Facoltativo.	

Elemento		Attributi
Nome e percorso	**Descrizione**	
package (entity-mappings)	Nome del pacchetto Java che conterrà le classi di mappatura. Dovrebbe contenere sempre il testo generic_db_adapter.	**Nome**. name **Descrizione.** Nome del tipo di CI di UCMDB al quale è mappata questa entità. Se questa entità è mappata a un collegamento nel CMDB, il nome dell'entità deve essere in formato <end_1>_<link_name>_<end_2>. Ad esempio node_composition_cpu definisce un'entità che sarò mappata al collegamento di composizione tra un nodo e una CPU. Se il nome del tipo di CI è lo stesso del nome della classe Java senza il prefisso del pacchetto, il campo può essere omesso. **Obbligatorietà.** Facoltativo **Tipo.** String
		Nome. Classe **Descrizione.** Nome completo della classe Java che sarà creata per questa entità del database. Il nome del pacchetto della classe Java deve essere lo stesso del nome indicato nell'elemento package. Non è possibile utilizzare le parole riservate Java come interfaccia o switch come nomi di classe. Aggiungere invece il prefisso gdba_ al nome (quindi interfaccia sarà generic_db_adapter.gdba_interface. **Obbligatorietà.** Obbligatorio **Tipo.** String

Elemento		Attributi
Nome e percorso	**Descrizione**	
table (entity-mappings > entity)	Questo elemento definisce la tabella primaria dell'entità del database. Può essere visualizzato solo una volta. Obbligatorio.	**Nome**. name **Descrizione.** Nome della tabella primaria. Se il nome della tabella non contiene lo schema al quale appartiene, sarà eseguita la ricerca nella tabella solo nello schema dell'utente utilizzato per creare il punto di integrazione. Questa può essere una qualsiasi dichiarazione SELECT valida. Se è una dichiarazione SELECT, deve essere incapsulata tra parentesi. **Obbligatorietà.** Obbligatorio **Tipo.** String
secondary-table (entity-mappings > entity)	Questo elemento può essere utilizzato per definire una tabella secondaria per l'entità del database. Questa tabella deve essere collegata alla tabella primaria con una relazione 1-to-1. È possibile definire più di una tabella secondaria. Facoltativo.	**Nome**. name **Descrizione.** Nome della tabella secondaria. Se il nome della tabella non contiene lo schema al quale appartiene, sarà eseguita la ricerca nella tabella solo nello schema dell'utente utilizzato per creare il punto di integrazione. Questa può essere una qualsiasi dichiarazione SELECT valida. Se è una dichiarazione SELECT, deve essere incapsulata tra parentesi. **Obbligatorietà.** Obbligatorio **Tipo.** String

Elemento		Attributi
Nome e percorso	**Descrizione**	
primary-key-join-column (entity-mappings > entity > secondary-table)	Se la tabella secondaria e la tabella primaria non vengono collegate utilizzando i campi con lo stesso nome, questo elemento definisce il nome del campo della chiave primaria nella tabella secondaria che deve essere collegata al campo della chiave primaria della tabella primaria.	**Nome**. name **Descrizione.** Nome del campo della chiave primaria nella tabella secondaria. Se questo elemento non esiste, si presume che il campo della chiave primaria abbia lo stesso nome del campo della chiave primaria della tabella primaria. **Obbligatorietà.** Facoltativo **Tipo.** String
inheritance (entity-mappings > entity)	Se l'entità corrente è l'entità principale di una famiglia di entità del database, utilizzare questo elemento per contrassegnarlo come tale. Facoltativo.	**Nome**. strategy **Descrizione.** Definisce il modo di implementazione dell'ereditarietà nel database. **Obbligatorietà.** Obbligatorio **Tipo.** Uno dei valori seguenti: ➤ SINGLE_TABLE - Questa entità e tutte le entità figlie esistono nella stessa tabella. ➤ JOINED - Le entità figlie sono tabelle unite. ➤ TABLE_PER_CLASS - Ciascuna entità è completamente definita da una tabella separata.
discriminator-column (entity-mappings > entity)	Se l'ereditarietà è di tipo SINGLE_TABLE, questo elemento viene utilizzato per definire il nome del campo utilizzato per stabilire il tipo di entità per ciascuna riga.	**Nome**. name **Descrizione.** Nome della colonna del discriminatore. **Obbligatorietà.** Obbligatorio **Tipo.** String

Elemento		Attributi
Nome e percorso	**Descrizione**	
discriminator-value (entity-mappings > entity)	Questo elemento definisce il tipo di entità specifica nella struttura dell'ereditarietà. Il nome deve essere lo stesso di quello definito nel file **discriminator.properties** per il gruppo di valori di questo specifico tipo di entità.	
attributes (entity-mappings > entity)	L'elemento principale per tutte le mappature dell'attributo di un'entità.	
id (entity-mappings > entity attributes)	Questo elemento definisce il campo della chiave per l'entità. Deve essere almeno un campo id definito. Se esiste più di un elemento id, i campi creano una chiave composta per l'entità. Si dovrebbe cercare di evitare le chiavi composte per le entità CI (non per i collegamenti).	**Nome**. name **Descrizione.** Una stringa di tipo idX, dove X è un numero tra 1 e 9. Il primo id deve essere contrassegnato come id1, il secondo come id2 e così via. Questo NON è il nome dell'attributo chiave in CMDB. **Obbligatorietà.** Obbligatorio **Tipo.** String
basic (entity-mappings > entity attributes)	Questo elemento definisce una mappatura tra un campo nella tabella, che non fa parte della chiave primaria della tabella, e un attributo di UCMDB.	**Nome**. name **Descrizione.** Nome dell'attributo di UCMDB al quale è mappato il campo. Questo attributo deve esistere nel tipo di CI di UCMDB al quale è mappata questa entità. **Obbligatorietà.** Obbligatorio **Tipo.** String

Elemento		Attributi
Nome e percorso	**Descrizione**	
column (entity-mappings > entity > attributes > id -OPPURE- (entity-mappings > entity > attributes > basic)	Definisce il nome della colonna nella tabella per la mappatura base o un campo id.	**Nome**. name **Descrizione.** Nome del campo. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. table **Descrizione.** Nome della tabella alla quale appartiene il campo. Deve essere la tabella primaria oppure una delle tabelle secondarie definite per l'entità. Se questo è omesso, si presume che il campo appartenga alla tabella primaria. **Obbligatorietà.** Facoltativo **Tipo.** String
one-to-one (entity-mappings > entity > attributes)	Definisce una colonna il cui valore è in un'altra tabella e le due tabelle vengono collegate utilizzando una relazione. Questo elemento è supportato soltanto per le mappature dell'entità del collegamento e non per altri tipi CI. Questo è l'unico modo per definire una mappatura tra una tabella e un collegamento di UCMDB.	**Nome**. name **Descrizione.** Quale delle due estremità rappresenta il campo. **Obbligatorietà.** Obbligatorio **Tipo.** O end1 oppure end2
		Nome. target-entity **Descrizione.** Nome dell'entità alla quale si riferisce il campo. **Obbligatorietà.** Obbligatorio **Tipo.** Uno dei nomi dell'entità definito nel documento di mappatura dell'entità

Elemento		Attributi
Nome e percorso	**Descrizione**	
join-column (entity-mappings > entity attributes > one-to-one)	Definisce il modo di giunzione dell'entità di destinazione definita nell'elemento principale one-to-one e l'entità corrente.	**Nome**. name **Descrizione.** Nome del campo nella tabella corrente che sarà utilizzato per eseguire giunzioni di tipo one-to-one. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. name **Descrizione.** Nome di un campo nell'entità congiunta in base al quale eseguire la giunzione. Se l'attributo è omesso, si presume che la tabella congiunta abbia una colonna con lo stesso nome come campo definito nell'attributo del nome. **Obbligatorietà.** Facoltativo **Tipo.** String

File reconciliation_types.txt

Questo file viene utilizzato per configurare i tipi di riconciliazione.

Ciascuna riga del file rappresenta un CIT di CMDB collegato a un CIT del database federato nella query TQL.

File reconciliation_rules.txt (per la contabilità inversa)

Questo file viene utilizzato per configurare le regole di riconciliazione se si desidera eseguire la riconciliazione quando DBMappingEngine è configurato nell'adattatore. Se non si utilizza DBMappingEngine, il meccanismo di riconciliazione generico di UCMDB viene utilizzato e non è necessario configurare questo file.

Ciascuna riga del file rappresenta una regola. Ad esempio:

```
multinode[node] expression[^node.name OR ip_address.name] end1_type[node]
end2_type[ip_address] link_type[containment]
```

Il multinodo viene compilato con il nome del multinodo (il CIT CMDB collegato a un CIT del database federato nella TQL).

Questa espressione comprende la logica che decide se due multinodi sono uguali (un multinodo nel CMDB e l'altro nella sorgente del database).

L'espressione è composta di OR o AND.

La convenzione relativa ai nomi dell'attributo nella parte di espressione è [className].[attributeName]. Ad esempio attributeName nella classe ip_address è scritto ip_address.name.

Per una corrispondenza ordinata (se la prima sottoespressione OR restituisce una risposta che dichiara che i multinodi non sono uguali, la seconda sottoespressione OR non viene confrontata), quindi utilizzare ordered expression invece di expression.

Per ignorare le maiuscole/minuscole durante un confronto, utilizzare il segno di controllo (^).

I parametri end1_type, end2_type e link_type vengono utilizzati solo se la TQL di riconciliazione contiene due nodi e non soltanto un multinodo. In questo caso, la TQL di riconciliazione è end1_type > (link_type) > end2_type.

Non è necessario aggiungere il layout rilevante poiché viene preso dall'espressione.

Tipi di regole di riconciliazione

Le regole di riconciliazione prendono la forma della condizioni OR e AND. È possibile definire queste regole su nodi diversi (ad esempio il nodo viene identificato da name from node AND/OR name from ip_address).

Le opzioni seguenti trovano una corrispondenza:

- **Ordered match.** L'espressione di riconciliazione viene letta da sinistra a destra. Due sottoespressioni OR sono considerate uguali se contengono valori e sono uguali. Due sottoespressioni OR sono considerate non uguali se entrambe contengono valori e non sono uguali. Per qualsiasi altro caso non si prendono decisioni e la successiva sottoespressione OR viene testata in base all'uguaglianza.

  name from node OR from ip_address. Se sia il CMDB sia l'origine dati includono name e sono uguali i nodi vengono considerati uguali. Se entrambi hanno name ma non sono uguali, i nodi vengono considerati non uguali senza testare il name di ip_address. Se o il CMDB o l'origine di dati non hanno name of node, viene verificato name of ip_address.

- **Regular match.** Se c'è uguaglianza in una delle sottoespressioni OR il CMDB e l'origine dati vengono considerati uguali.

  name from node OR from ip_address. Se non c'è corrispondenza sul name of node, viene verificato per l'uguaglianza name of ip_address.

Per le riconciliazioni complesse, dove l'entità di riconciliazione viene modellata nel modello di classe come più CIT con relazioni (ad esempio node), la mappatura di un nodo superset include tutti gli attributi rilevanti da tutti i CIT modellati.

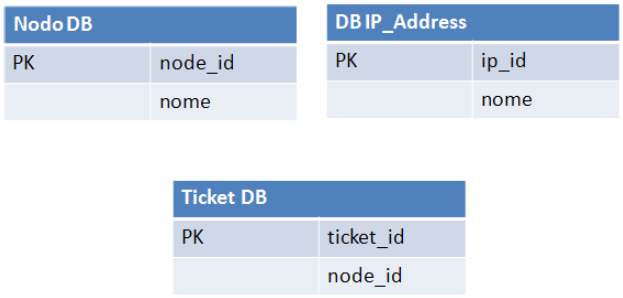
Nota: di conseguenza, c'è la limitazione che tutti gli attributi di riconciliazione nell'origine di dati devono risiedere nelle tabelle che condividono la stessa chiave primaria.

Un'altra limitazione dichiara che la TQL di riconciliazione non deve avere più di due nodi. Ad esempio la TQL node > ticket ha un nodo nel CMDB e un ticket nell'origine di dati.

Per riconciliare i risultati, è necessario recuperare name dal nodo e/o ip_address.

Se name nel CMDB è nel formato *.m.com, è possibile utilizzare un convertitore dal CMDB al database federato e viceversa per convertire questi valori.

La colonna node_id nella tabella dei ticket del database viene utilizzata per collegare le entità (l'associazione definita può inoltre essere eseguita in una tabella del nodo):



Nota: le tre tabelle devono essere parte dell'origine federata del RDBMS e non del database CMDB.

File transformations.txt

Questo file contiene tutte le definizioni del convertitore.

Il formato è che ogni riga contiene una nuova definizione.

Esemplare del file transformations.txt

```
entity[[CMDB_class_name]] attribute[[CMDB_attribute_name]]
to_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)]
```

entity. Nome dell'entità come viene visualizzata nel file orm.xml.

attribute. Nome dell'attributo come viene visualizzato nel file orm.xml.

to_DB_class. Nome completo di una classe che implementa l'interfaccia **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB**. Gli elementi in parentesi sono assegnati a questo costruttore di classe. Utilizzare questo convertitore per trasformare i valori di CMDB in valori di database, ad esempio per collegare il suffisso di **.com** a ciascun nome di nodo.

from_DB_class. Nome completo di una classe che implementa l'interfaccia **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB** interface. Gli elementi in parentesi sono assegnati a questo costruttore di classe. Utilizzare questo convertitore per trasformare i valori di database in valori di CMDB, ad esempio per collegare il suffisso di **.com** a ciascun nome di nodo.

Per i dettagli consultare "Convertitori preimpostati" a pag. 200.

File persistence.xml

Questo file viene utilizzato per sostituire le impostazioni predefinite di Hibernate e aggiungere supporto per i tipi di database che non sono preimpostati (tipi di database preimpostati sono Oracle Server, Microsoft MSSQL Server e MySQL).

Se si necessita di supporto per un nuovo tipo di database, accertarsi di disporre del provider di pool di connessioni (valore predefinito c3p0) e di un driver JDBC per il proprio database (collocare i file *.jar nella cartella dell'adattatore).

Per visualizzare tutti i valori Hibernate disponibili che si possono cambiare, verificare la classe **org.hibernate.cfg.Environment**.

Esempio di file persistence.xml:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/
xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
    <!-- Don't change this value -->
    <persistence-unit name="GenericDBAdapter">
        <proprietà>
            <!-- Don't change this value -->
            <property name="hibernate.archive.autodetection" value="class, hbm"/>
            <!--The driver class name"/-->
            <property name="hibernate.connection.driver_class"
value="com.mercury.jdbc.MercOracleDriver"/>
            <!--The connection url"/-->
            <property name="hibernate.connection.url" value="jdbc:mercury:oracle://
artist:1521;sid=cmdb2"/>
            <!--DB login credentials"/-->
            <property name="hibernate.connection.username" value="CMDB"/>
            <property name="hibernate.connection.password" value="CMDB"/>
            <!--connection pool properties"/-->
            <property name="hibernate.c3p0.min_size" value="5"/>
            <property name="hibernate.c3p0.max_size" value="20"/>
            <property name="hibernate.c3p0.timeout" value="300"/>
            <property name="hibernate.c3p0.max_statements" value="50"/>
            <property name="hibernate.c3p0.idle_test_period" value="3000"/>
            <!--The dialect to use-->
            <property name="hibernate.dialect"
value="org.hibernate.dialect.OracleDialect"/>
        </properties>
    </persistence-unit>
</persistence>
```

File discriminator.properties

Questo file mappa ciascun tipo di CI supportato (utilizzato anche come valore del discriminatore nel file orm.xml) in un elenco separato da virgola di possibili valori corrispondenti della colonna del discriminatore.

Se l'adattatore che si sta creando utilizza le funzioni del discriminatore, è necessario definire tutti i valori del discriminatore nel file **discriminator.properties**.

Esempio di mappatura del discriminatore:

Il file **discriminator.properties** include il codice seguente:

```
node=10001, 10005,10010,10011,10012
nt=10002,10003
unix=10004,10006,10008
```

Il file orm.xml file include il codice seguente:

```
<entity class="generic_db_adapter.node" >
    <table name="[table_name]"/>
    …
    <inheritance strategy="SINGLE_TABLE"/>
    <discriminator-value>node</discriminator-value>
    <discriminator-column name="[discriminator_column]"/>
    …
</entity>
<entity class="generic_db_adapter.nt" name="nt">
    <discriminator-value>nt</discriminator-value>
    <attributes/>
</entity>
<entity class="generic_db_adapter.unix" name="unix">
    <discriminator-value>unix</discriminator-value>
    <attributes/>
</entity>
```

L'attributo [discriminator_column] viene calcolato come segue:

- La colonna del discriminatore della tabella corrispondente contiene 10002 per una determinata entità. La voce viene mappata al CIT **nt**.
- La colonna del discriminatore della tabella corrispondente contiene 10006 per una determinata entità. La voce viene mappata al CIT **unix**.
- La colonna del discriminatore della tabella corrispondente contiene 10010 per una determinata entità. La voce viene mappata al CIT **node**.

Tenere presente che il CIT **node** è anche padre di **nt** e **unix**.

File replication_config.txt

Questo file contiene un elenco separato da virgola di tipi di CI e relazioni le cui condizioni delle proprietà sono supportate dal plug-in di replica. Per i dettagli consultare "Plug-in" a pag. 204.

File fixed_values.txt

Questo file consente di configurare valori fissi per determinati attributi di certi CIT. In questo modo a ciascuno di questi attributi può essere assegnato un valore fisso che non è memorizzato nel database.

Il file deve contenere zero o più voci nel formato seguente:

```
entity[<entityName>] attribute[<attributeName>] value[<value>]
```

Ad esempio:

```
entity[ip_address] attribute[ip_domain] value[DefaultDomain]
```

Convertitori preimpostati

È possibile utilizzare i convertitori seguenti (trasformatori) per convertire le query federate e i processi di replica da e verso i dati del database.

In questa sezione vengono trattati i seguenti argomenti:

Convertitore enum-transformer

Questo convertitore utilizza un file XML che viene assegnato come parametro di input.

Il XML file mappa i valori hard-coded di CMDB e i valori del database (enums). Se uno dei valori non esiste è possibile scegliere di restituire lo stesso valore, restituire null oppure generare un'eccezione.

Utilizzare un file di mappatura XML per ciascun attributo dell'entità.

Nota: questo convertitore può essere utilizzato per i campi to_DB_class e from_DB_class nel file **transformations.txt**.

Esempio di file di input XSD:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="enum-transformer">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="value" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="DB-type" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="integer"/>
                        <xs:enumeration value="long"/>
                        <xs:enumeration value="float"/>
                        <xs:enumeration value="double"/>
                        <xs:enumeration value="boolean"/>
                        <xs:enumeration value="string"/>
                        <xs:enumeration value="date"/>
                        <xs:enumeration value="xml"/>
                        <xs:enumeration value="bytes"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="CMDB-type" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="integer"/>
```

```
                        <xs:enumeration value="long"/>
                        <xs:enumeration value="float"/>
                        <xs:enumeration value="double"/>
                        <xs:enumeration value="boolean"/>
                        <xs:enumeration value="string"/>
                        <xs:enumeration value="date"/>
                        <xs:enumeration value="xml"/>
                        <xs:enumeration value="bytes"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="non-existing-value-action" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="return-null"/>
                        <xs:enumeration value="return-original"/>
                        <xs:enumeration value="throw-exception"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:complexType>
    </xs:element>
    <xs:element name="value">
        <xs:complexType>
            <xs:attribute name="CMDB-value" type="xs:string" use="required"/>
            <xs:attribute name="external-DB-value" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Esempio di conversione del valore 'sys' nel valore 'System':

In questo esempio il valore sys nel CMDB viene trasformato nel valore System nel database federato e il valore System nel database federato viene trasformato nel valore sys nel CMDB.

Se il valore non esiste nel file XML (ad esempio la stringa demo), il convertitore restituisce lo stesso valore di input ricevuto.

```
<enum-transformer CMDB-type="string" DB-type="string" non-existing-value-action="return-original"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../
META-CONF/generic-enum-transformer.xsd">
    <value CMDB-value="sys" external-DB-value="System"/>
</enum-transformer>
```

Convertitore SuffixTransformer

Questo convertitore viene utilizzato per aggiungere o rimuovere suffissi dal CMDB o valori dell'origine del database federato.

Sono possibili due implementazioni:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddSuffixTransformer**. Aggiunge il suffisso (assegnato come input) quando si esegue la conversione dal valore del database federato al valore di CMDB e rimuove il suffisso quando si esegue la conversione dal valore del CMDB al valore del database federato.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemoveSuffixTransformer**. Rimuove il suffisso (assegnato come input) quando si esegue la conversione dal valore del database federato al valore di CMDB e aggiunge il suffisso quando si esegue la conversione dal valore del CMDB al valore del database federato.

Convertitore PrefixTransformer

Questo convertitore viene utilizzato per aggiungere o rimuovere un prefisso dal CMDB o valori del database federato.

Sono possibili due implementazioni:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddPrefixTransformer**. Aggiunge il prefisso (assegnato come input) quando si esegue la conversione dal valore del database federato al valore di CMDB e rimuove il prefisso quando si esegue la conversione dal valore del CMDB al valore del database federato.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemovePrefixTransformer**. Rimuove il prefisso (assegnato come input) quando si esegue la conversione dal valore del database federato al valore di CMDB e aggiunge il prefisso quando si esegue la conversione dal valore del CMDB al valore del database federato.

Convertitore BytesToStringTransformer

Questo convertitore viene utilizzato per convertire matrici di byte nel CMDB nella rispettiva rappresentazione della stringa nell'origine del database federato.

Il convertitore è:
com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer.

Plug-in

L'adattatore generico del database supporta i plug-in seguenti:

- Un plug-in facoltativo per la sincronizzazione completa della topologia.
- Un plug-in facoltativo per la sincronizzazione dei cambiamenti nella topologia. Se non viene implementato alcun plug-in per la sincronizzazione, è possibile eseguire una sincronizzazione differenziale ma tale sincronizzazione sarà effettivamente completa.
- Un plug-in facoltativo per il layout della sincronizzazione.
- Un plug-in facoltativo per il recupero delle query supportate per la sincronizzazione. Se il plug-in non è definito vengono restituiti tutti i nomi TQL.
- Plug-in interno facoltativo per cambiare la definizione TQL e il risultato TQL.
- Plug-in interno facoltativo per cambiare una richiesta di layout e il risultato dei CI.
- Plug-in interno facoltativo per cambiare una richiesta di layout e il risultato delle relazioni.

Per i dettagli sull'implementazione e la distribuzione di plug-in, consultare "Implementare un plug-in" a pag. 152.

Esempi di configurazione

In questa sezione vengono presentati esempi di configurazione.

In questa sezione vengono trattati i seguenti argomenti:

Caso di utilizzo

Caso di utilizzo. Una TQL è:

node > **(composition)** > **card**

dove:

node è l'entità del CMDB

card è l'entità dell'origine del database federato

composition è la relazione tra di essi

L'esempio viene eseguito rispetto al database ED. ED nodes viene memorizzato nella tabella Device e card viene memorizzato nella tabella hwCards. Negli esempi seguenti card è sempre mappato nello stesso modo.

Riconciliazione del singolo nodo

In questo esempio viene eseguita la riconciliazione rispetto alla proprietà name.

Definizione semplificata

La riconciliazione viene eseguita da node e viene enfatizzata dalla classe del tag speciale **CMDB-class**.

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-single-node>
            <or>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"/>
            </or>
        </reconciliation-by-single-node>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="composition">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-column="Device_ID"/>
        <primary-key column-name="hwCards_Seq"/>
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
    </class>
</generic-DB-adapter-config>
```

Definizione avanzata

File orm.xml

Prestare attenzione all'aggiunta della mappatura della relazione. Per i dettagli consultare la sezione sulla definizione in "File orm.xml" a pag. 182.

Esempio di file orm.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/
persistence/orm http://java.sun.com/xml/ns/persistence/orm_1_0.xsd" version="1.0">
    <description>Generic DB adapter orm</description>
```

```
<package>generic_db_adapter</package>
<entity class="generic_db_adapter.node" >
    <table name="Device"/>
    <attributes>
        <id name="id1">
            <column name="Device_ID" insertable="false" updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <basic name="name">
            <column name="Device_Name"/>
        </basic>
    </attributes>
</entity>
<entity class="generic_db_adapter.card" >
    <table name="hwCards"/>
    <attributes>
        <id name="id1">
            <column name="hwCards_Seq" insertable="false" updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <basic name="card_class">
            <column name="hwCardClass" insertable="false" updatable="false"/>
        </basic>
        <basic name="card_vendor">
            <column name="hwCardVendor" insertable="false" updatable="false"/>
        </basic>
        <basic name="card_name">
            <column name="hwCardName" insertable="false" updatable="false"/>
        </basic>
    </attributes>
</entity>
<entity class="generic_db_adapter.node_composition_card" >
    <table name="hwCards"/>
    <attributes>
        <id name="id1">
            <column name="hwCards_Seq" insertable="false" updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <many-to-one name="end1" target-entity="node">
            <join-column name="Device_ID" insertable="false" updatable="false"/>
        </many-to-one>
```

```
            <one-to-one name="end2" target-entity="card">
                <join-column name="hwCards_Seq" referenced-column-name="hwCards_Seq"
insertable="false" updatable="false"/>
            </one-to-one>
        </attributes>
    </entity>
</entity-mappings>
```

File reconciliation_types.txt

Per i dettagli consultare "File reconciliation_types.txt" a pag. 193.

```
node
```

File reconciliation_rules.txt

Per i dettagli consultare "File reconciliation_rules.txt (per la contabilità inversa)" a pag. 194.

```
multinode[node] expression[node.name]
```

File transformation.txt

Questo file resta vuoto poiché non sono necessari valori da convertire in questo esempio.

Riconciliazione di due nodi

In questo esempio, la riconciliazione viene calcolata in base alla proprietà name di node e di ip_address con varianti diverse.

La TQL di riconciliazione è **node** > **(containment)** > **ip_address**.

Definizione semplificata

La riconciliazione avviene per name di node OR di ip_address:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address"
CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"/>
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress"/>
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-column="Device_ID"/>
        <primary-key column-name="hwCards_Seq"/>
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
    </class>
</generic-DB-adapter-config>
```

La riconciliazione avviene per name di node AND di ip_address:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address"
CMDB-link-type="containment">
            <and>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"/>
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress"/>
            </and>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-column="Device_ID"/>
        <primary-key column-name="hwCards_Seq"/>
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
    </class>
</generic-DB-adapter-config>
```

La riconciliazione avviene per name di ip_address:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address"
CMDB-link-type="containment">
            <or>
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress"/>
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-column="Device_ID"/>
        <primary-key column-name="hwCards_Seq"/>
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
    </class>
</generic-DB-adapter-config>
```

Definizione avanzata

File orm.xml

Poiché l'espressione di riconciliazione non viene definita in questo file, la stessa versione deve essere utilizzata per qualsiasi espressione di riconciliazione.

File reconciliation_types.txt

Per i dettagli consultare "File reconciliation_types.txt" a pag. 193.

```
node
```

File reconciliation_rules.txt

Per i dettagli consultare "File reconciliation_rules.txt (per la contabilità inversa)" a pag. 194.

```
multinode[node] expression[ip_address.name OR node.name] end1_type[node]
end2_type[ip_address] link_type[containment]
```

```
multinode[node] expression[ip_address.name AND node.name] end1_type[node]
end2_type[ip_address] link_type[containment]
```

```
multinode[node] expression[ip_address.name] end1_type[node] end2_type[ip_address]
link_type[containment]
```

File transformation.txt

Questo file resta vuoto poiché non sono necessari valori da convertire in questo esempio.

Utilizzo di una chiave primaria che contiene più di una colonna

Se la chiave primaria è composta da più di una colonna, il codice seguente viene aggiunto alle definizioni XML:

Definizione semplificata

Esiste più di un tag di chiave primaria e per ciascuna colonna esiste un tag.

```
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID"
CMDB-class-primary-key-column="Device_ID"/>
        <primary-key column-name="Device_ID"/>
        <primary-key column-name="hwBusesSupported_Seq"/>
        <primary-key column-name="hwCards_Seq"/>
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
        <attribute CMDB-attribute-name="card_vendor"
column-name="hwCardVendor"/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
    </class>
```

Definizione avanzata

File orm.xml

Viene aggiunta una nuova entità id mappata alle colonne della chiave primaria. Le entità che utilizzano questa entità id devono aggiungere un tag speciale.

Se si utilizza una chiave esterna (tag join-column) per questa chiave primaria è necessario eseguire la mappatura tra ciascuna colonna della chiave esterna e una colonna della chiave primaria.

Per i dettagli consultare "File orm.xml" a pag. 182.

Esempio di file orm.xml:

```
<   <entity class="generic_db_adapter.card" >
    <table name="hwCards"/>
    <attributes>
        <id name="id1">
            <column name="Device_ID" insertable="false" updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <id name="id2">
            <column name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <id name="id3">
            <column name="hwCards_Seq" insertable="false" updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
.
.
.
<entity class="generic_db_adapter.node_containment_card" >
    <table name="hwCards"/>
    <attributes>
        <id name="id1">
            <column name="Device_ID" insertable="false" updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <id name="id2">
            <column name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <id name="id3">
```

```
            <column name="hwCards_Seq" insertable="false" updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <many-to-one name="end1" target-entity="node">
            <join-column name="Device_ID" insertable="false" updatable="false"/>
        </many-to-one>
        <one-to-one name="end2" target-entity="card">
            <join-column name="Device_ID" referenced-column-name="Device_ID" insertable="false"
updatable="false"/>
            <join-column name="hwBusesSupported_Seq"
referenced-column-name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
            <join-column name="hwCards_Seq" referenced-column-name="hwCards_Seq"
insertable="false" updatable="false"/>
        </one-to-one>
    </attributes>
  </entity>
</entity-mappings>
```

Utilizzo delle trasformazioni

Nell'esempio seguente il trasformatore generico **enum** viene convertito rispettivamente dai valori 1, 2, 3 nei valori a, b, c nella colonna name.

Il file di mappatura è generic-enum-transformer-example.xml.

```
<enum-transformer CMDB-type="string" DB-type="string"
non-existing-value-action="return-original" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:noNamespaceSchemaLocation="../META-CONF/
generic-enum-transformer.xsd">
    <value CMDB-value="1" external-DB-value="a"/>
    <value CMDB-value="2" external-DB-value="b"/>
    <value CMDB-value="3" external-DB-value="c"/>
</enum-transformer>
```

Definizione semplificata

```
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address"
CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"
from-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.i
mpl.GenericEnumTransformer(generic-enum-transformer-example.xml)"
to-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)"/>
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress"/>
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
.
.
.
```

Definizione avanzata

C'è un cambiamento soltanto nel file **transformation.txt**.

File transformation.txt

Accertarsi che i nomi degli attributi e i nomi delle entità siano gli stessi del file orm.xml.

```
entity[node] attribute[name]
to_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.Generic
EnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.Gene
ricEnumTransformer(generic-enum-transformer-example.xml)]
```

File di registro dell'adattatore

Per comprendere i flussi di calcolo e il ciclo di vita dell'adattatore e visualizzare le informazioni di debug è possibile consultare i file di registro seguenti.

In questa sezione vengono trattati i seguenti argomenti:

➤ "Livelli di registro" a pag. 216

➤ "Percorsi del registro" a pag. 217

Livelli di registro

Si può configurare il livello di registro per ciascuno dei registri.

In un editor di testo aprire il file **C:\hp\UCMDB\UCMDBServer\conf\log\fcmdb.gdba.properties**.

Il livello di registro predefinito è **ERROR**:

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL
loglevel=ERROR
```

➤ Per aumentare il livello di registro per tutti i file di registro, cambiare **loglevel=ERROR** in **loglevel=DEBUG** oppure **loglevel=INFO**.

➤ Per cambiare il livello di registro per un file specifico, cambiare di conseguenza la riga specifica della categoria **log4j**. Ad esempio per cambiare il livello di registro di fcmdb.gdba.dal.sql.log in **INFO**, cambiare

```
log4j.category.fcmdb.gdba.dal.SQL=${loglevel},fcmdb.gdba.dal.SQL.appender
```

in:

```
log4j.category.fcmdb.gdba.dal.SQL=INFO,fcmdb.gdba.dal.SQL.appender
```

Percorsi del registro

I file di registro si trovano nella directory **C:\hp\UCMDB\UCMDBServer\runtime\log** .

- **Fcmdb.gdba.log**

  Registro del ciclo di vita dell'adattatore. Fornisce i dettagli sui tempi di avvio e di arresto dell'adattatore e su quali CIT sono supportati da questo adattatore.

  Consultarlo per errori di inizializzazione (caricamento/scaricamento adattatore).

- **fcmdb.log**

  Consultarlo per le eccezioni.

- **cmdb.log**

  Consultarlo per le eccezioni.

- **Fcmdb.gdba.mapping.engine.log**

  Registro del motore di mappatura. Fornisce i dettagli sulla TQL di riconciliazione che utilizza il motore di mappatura e le topologie di riconciliazione confrontate durante la fase di connessione.

  Consultare questo registro quando una query TQL non fornisce risultati anche se si sa che ci sono CI rilevanti nel database oppure i risultati non sono previsti (verificare la riconciliazione).

- **Fcmdb.gdba.TQL.log**

  Registro della TQL. Fornisce dettagli sulle query TQL e i rispettivi risultati.

  Consultare questo registro quando una query TQL non restituisce risultati e il registro del motore di mappatura mostra che non ci sono risultati nell'origine di dati federati.

- **Fcmdb.gdba.dal.log**

  Registro del ciclo di vita DAL. Fornisce dettagli sulla generazione di CIT e sulla connessione del database.

  Consultare questo registro quando non è possibile connettersi al database oppure quando ci sono CIT o attributi che non sono supportati dalla query.

➤ **Fcmdb.gdba.dal.command.log**

Registro delle operazioni DAL. Fornisce dettagli sulle operazioni DAL interne chiamate. (Questo registro è simile a cmdb.dal.command.log).

➤ **Fcmdb.gdba.dal.SQL.log**

Registro delle query SQL DAL. Fornisce dettagli sulle query JPAQL (query SQL object oriented) e i rispettivi risultati.

Consultare questo registro quando non è possibile connettersi al database oppure quando ci sono CIT o attributi che non sono supportati dalla query.

➤ **Fcmdb.gdba.hibrnate.log**

Registro Hibernate. Fornisce dettagli sulle query SQL che vengono eseguite, il parsing di ciascun JPAQL su SQL, i risultati delle query, i dati relativi alla chache Hibernate e così via. Per i dettagli su Hibernate consultare "Hibernate come provider JPA" a pag. 131.

Riferimenti esterni

Per i dettagli sulla specifica JavaBeans 3.0, consultare http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

Risoluzione dei problemi e limitazioni

Questa sezione descrive la risoluzione dei problemi e le limitazioni per l'adattatore generico del database.

Limitazioni generali

➤ Autenticazione SQL Server NTLM non supportata.

➤ Quando si aggiorna un pacchetto dell'adattatore, utilizzare Notepad++, UltraEdit o un altro editor di testo di terze parti anziché Notepad (qualsiasi versione) di Microsoft Corporation per modificare i file esemplari. Ciò evita l'utilizzo di simboli speciali che impedisce la distribuzione del pacchetto preparato.

Limitazioni JPA

- Tutte le tabelle devono avere una colonna della chiave primaria.
- I nomi degli attributi di classe di CMDB devono seguire la convezione di denominazione JavaBeans (ad esempio i nomi devono iniziare con lettere minuscole).
- Due CI connessi a una relazione nel modello di classe devono avere associazione diretta nel database (ad esempio se node è connesso a ticket deve esserci una chiave esterna o tabella di collegamento che li connette).
- Tabelle diverse mappate allo stesso CIT devono condividere la stessa tabella di chiave primaria.

Limitazioni funzionali

- Non è possibile creare una relazione manuale tra il CMDB e i CIT federati. Per definire le relazioni virtuali è necessario definire una logica di relazione speciale (si può basare sulle proprietà della classe federata).
- I CIT federati non possono attivare i CIT di una regola d'impatto ma possono essere inclusi in una query TQL di impatto analisi.
- Un CIT federato può essere parte di una TQL di accrescimento ma non può essere utilizzato come nodo sul quale viene eseguito l'accrescimento (non è possibile aggiungere, aggiornare o eliminare il CIT federato).
- L'utilizzo di un qualificatore di classe in una condizione non è supportato.
- I sottografici non sono supportati.
- Le relazioni compound non sono supportate
- Il CMDB id del CI esterno è composto dalla rispettiva chiave primaria e non dagli attributi della chiave.
- Una colonna di tipo bytes non può essere utilizzata come colonna di chiave primaria in Microsoft SQL Server.
- Il calcolo della query TQL non riesce se le condizioni dell'attributo definite su un nodo federato non hanno i nomi mappati nel file **orm.xml** file.
- L'adattatore generico del database non supporta l'autenticazione Windows per SQL Server.

6

Sviluppo degli adattatori Java

Questo capitolo comprende:

Concetti

Compiti

Riferimenti

Concetti

Framework di federazione - Panoramica

Nota:

- Il termine **relazione** equivale al termine **collegamento**.
- Il termine **CI** equivale al termine **oggetto**.
- Un grafico è una raccolta di nodi e collegamenti.
- Per un glossario di termini e definizioni consultare "Glossario" nella *Guida all'amministrazione di HP Universal CMDB*.

La funzionalità del framework di federazione utilizza un'API per recuperare le informazioni dalle origini federate. Il framework di federazione garantisce tre funzioni principali:

- **Federazione** in tempo reale. Tutte le query vengono eseguite su repository di dati originali e i risultati sono generati in tempo reale nel CMDB.
- **Popolamento.** Popola i dati (dati topologici e proprietà CI) nel CMDB da un'origine dati esterna.
- **Invio dati.** Invia i dati (dati topologici e proprietà CI) dal CMDB locale a un'origine dati remota.

Tutti i tipi di azione richiedono un adattatore per ciascun repository di dati, che può garantire le specifiche funzioni del repository di dati nonché recuperare e/o aggiornare i dati richiesti. Ogni richiesta al repository di dati viene eseguita attraverso il relativo adattatore.

La sezione è suddivisa negli argomenti seguenti:

- "Federazione in tempo reale" a pag. 223
- "Invio dati" a pag. 224
- "Popolamento" a pag. 225

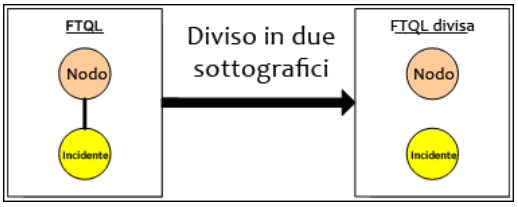
Federazione in tempo reale

Le query TQL federate permettono il recupero dei dati da un repository di dati esterno senza replicarne i relativi dati.

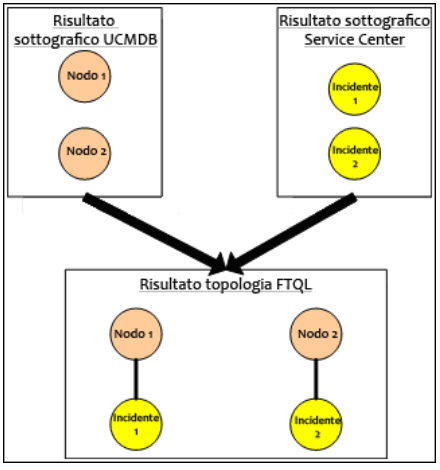
Una query TQL federata utilizza adattatori che rappresentano repository di dati esterni, al fine di creare relazioni esterne adeguate tra CI di repository di dati esterni diversi e CI di UCMDB.

Esempio di flusso di federazione in tempo reale:

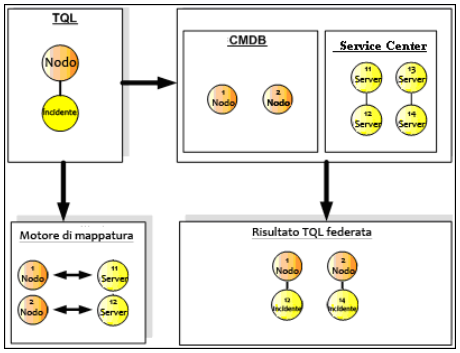
1 Il framework di federazione suddivide una query TQL federata in vari sottografici; tutti i nodi di un sottografico fanno riferimento allo stesso repository di dati. Ciascun sottografico è connesso ad altri sottografici da una relazione virtuale (ma non contiene di per sé relazioni virtuali).



2 Dopo che la query TQL federata viene suddivisa in sottografici, il framework di federazione calcola la topologia di ciascun sottografico e connette due sottografici appropriati creando relazioni virtuali tra i nodi adeguati.



3 Dopo aver completato il calcolo della topologia della TQL federata, il framework di federazione recupera un layout per il risultato della topologia.



Invio dati

Il flusso di invio dati viene utilizzato per sincronizzare i dati del CMDB locale corrente con un servizio remoto o repository di dati di destinazione.

Nell'invio dati, i repository di dati vengono suddividi in due categorie: origine (CMDB locale) e destinazione. I dati vengono recuperati dal repository di dati di origine e aggiornati nel repository di dati di destinazione. Il processo di invio dati si basa sui nomi query, ovvero i dati vengono sincronizzati tra il repository di dati di origine (CMDB locale) e il repository di dati di destinazione, nonché recuperati tramite un nome query TQL dal CMDB locale.

Il flusso del processo di invio dati include i seguenti passaggi:

1 Recupero del risultato della topologia con firme dal repository di dati di origine.

2 Confronto tra i risultati nuovi e quelli precedenti.

3 Recupero del layout completo (ovvero tutte le proprietà CI) dei CI e delle relazioni, esclusivamente per i risultati cambiati.

4 Aggiornamento del repository di dati di destinazione con il layout completo dei CI e delle relazioni ricevuto. Se viene eliminato un CI o una relazione nel repository di dati di origine e la query è esclusiva, il processo di replica rimuove il CI o la relazione anche nel repository di dati di destinazione.

Il CMDB ha 2 origini dati nascoste (**hiddenRMIDataSource** e **hiddenChangesDataSource**), che corrispondono sempre all'origine dati 'di origine' nei flussi di invio dati. Per implementare un nuovo adattatore per i flussi di invio dati, è necessario implementare solo l'adattatore 'di destinazione'.

Popolamento

Viene utilizzato il flusso di popolamento per popolare il CMDB con dati provenienti dalle origini esterne.

Il flusso utilizza sempre un'origine dati 'di origine' per recuperare i dati e invia i dati recuperati alla sonda attraverso un processo simile al flusso di un processo di individuazione.

Per implementare un nuovo adattatore per i flussi di popolamento, è necessario implementare solo l'adattatore di origine, poiché la sonda del flusso di dati agisce da destinazione.

L'adattatore nel flusso di popolamento viene eseguito sulla sonda. Il debug e le registrazioni devono essere eseguiti sulla sonda e non sul CMDB.

Il flusso di popolamento si basa sui nomi query, ovvero i dati vengono sincronizzati tra il repository di dati di origine e la sonda del flusso di dati, nonché recuperati tramite un nome query nel repository di dati di origine. Ad esempio, in UCMDB, il nome query è il nome della query TQL. Tuttavia, in un altro repository di dati, il nome query può essere un nome codice che restituisce dati. L'adattatore è progettato per gestire correttamente il nome query.

Ciascun processo può essere definito come esclusivo. Ciò significa che i CI e le relazioni nei risultati del processo sono univoci nel CMDB locale e che nessuna altra query può portarli nella destinazione. L'adattatore del repository di dati di origine supporta query specifiche e può recuperare i dati da questo repository di dati. L'adattatore del repository di dati di destinazione consente l'aggiornamento dei dati recuperati su questo repository di dati.

Flusso SourceDataAdapter

- Recupera il risultato della topologia con firme dal repository di dati di origine.
- Confronta i risultati nuovi rispetto ai precedenti.
- Recupera un layout completo (ovvero tutte le proprietà CI) dei CI e delle relazioni, esclusivamente per i risultati cambiati.
- Aggiorna il repository di dati di destinazione con il layout completo dei CI e delle relazioni ricevuto. Se viene eliminato un CI o una relazione nel repository di dati di origine e la query è esclusiva, il processo di replica rimuove il CI o la relazione anche nel repository di dati di destinazione.

Flusso SourceChangesDataAdapter

- Recupera il risultato della topologia a partire dall'ultima data fornita.
- Recupera un layout completo (ovvero tutte le proprietà CI) dei CI e delle relazioni, esclusivamente per i risultati cambiati.
- Aggiorna il repository di dati di destinazione con il layout completo dei CI e delle relazioni ricevuto. Se viene eliminato un CI o una relazione nel repository di dati di origine e la query è esclusiva, il processo di replica rimuove il CI o la relazione anche nel repository di dati di destinazione.

Flusso PopulateDataAdapter

- Recupera la topologia completa con il risultato di layout richiesto.
- Utilizza il meccanismo a blocchi della topologia per recuperare i dati in blocchi.
- La sonda filtra qualsiasi dato già portato nelle precedenti esecuzioni
- Aggiorna il repository di dati di destinazione con il layout dei CI e delle relazioni ricevuto. Se viene eliminato un CI o una relazione nel repository di dati di origine e la query è esclusiva, il processo di replica rimuove il CI o la relazione anche nel repository di dati di destinazione.

Flusso PopulateChangesDataAdapter

- Recupera la topologia con il risultato di layout richiesto che presenta modifiche dall'ultima esecuzione.
- Utilizza il meccanismo a blocchi della topologia per recuperare i dati in blocchi.
- La sonda filtra qualsiasi dato già portato nelle precedenti esecuzioni (incluso questo flusso).
- Aggiorna il repository di dati di destinazione con il layout dei CI e delle relazioni ricevuto. Se viene eliminato un CI o una relazione nel repository di dati di origine e la query è esclusiva, il processo di replica rimuove il CI o la relazione anche nel repository di dati di destinazione.

 Interazione dell'adattatore e della mappatura con il framework di federazione

Un adattatore è un'entità in UCMDB che rappresenta i dati esterni (i dati non salvati in UCMDB). Nei flussi federati, tutte le interazioni con le origini dati esterne sono eseguite attraverso gli adattatori. Il flusso di interazione del framework di federazione e le interfacce dell'adattatore sono diversi per la replica e per le query TQL federate.

La sezione è suddivisa negli argomenti seguenti:

- "Ciclo di vita dell'adattatore" a pag. 227
- "Metodi assist dell'adattatore" a pag. 228

Ciclo di vita dell'adattatore

Viene creata un'istanza dell'adattatore per ciascun repository di dati esterno. L'adattatore inizia il suo ciclo di vita con la prima azione ad esso applicata (ad esempio, calculate TQL o retrieve/update data). Quando viene chiamato il metodo **start**, l'adattatore riceve informazioni ambientali, come la configurazione del repository di dati, il registratore, e così via. Il ciclo di vita dell'adattatore termina quando il repository di dati viene rimosso dalla configurazione e viene chiamato il metodo **shutdown**. Ciò significa che l'adattatore ha uno stato e che può contenere la connessione al repository di dati esterno se richiesto.

Metodi assist dell'adattatore

L'adattatore dispone di vari metodi assist che consentono di aggiungere configurazioni dei repository di dati esterni. Tali metodi non fanno parte del ciclo di vita dell'adattatore e creano un nuovo adattatore ogni volta che vengono chiamati.

- Il primo metodo verifica la connessione al repository di dati esterno per una data configurazione. testConnection può essere eseguito sul server UCMDB o sulla sonda del flusso di dati, a seconda del tipo di adattatore.
- Il secondo metodo riguarda solo l'adattatore d'origine e restituisce le query supportate per la replica. (Questo metodo viene eseguito esclusivamente sulla sonda).
- Il terzo metodo riguarda solo i flussi di federazione e di popolamento e restituisce le classe esterna supportate dal repository di dati esterno. (Questo metodo viene eseguito sul server UCMDB).

Tutti questi metodi vengono utilizzati quando si creano o si visualizzano le configurazioni di integrazione.

Flusso del framework di federazione per le query TQL federate

In questa sezione vengono trattati i seguenti argomenti:

Definizioni e termini

Dati di riconciliazione. La regola per creare la corrispondenza dei CI del tipo specificato che vengono ricevuti dal CMDB e repository di dati esterno. La regola di riconciliazione può essere di tre tipi:

- **Riconciliazione ID.** Può essere utilizzata solo se il repository di dati esterno contiene l'ID CMDB degli oggetti di riconciliazione.
- **Riconciliazione proprietà.** Viene utilizzata quando la corrispondenza può essere eseguita tramite le proprietà solo del tipo CI di riconciliazione.
- **Riconciliazione topologia.** Viene utilizzata quando sono richieste le proprietà di CIT aggiuntivi (non solo del CIT di riconciliazione) per eseguire una corrispondenza sui CI di riconciliazione. Ad esempio, è possibile eseguire la riconciliazione del tipo nodo tramite la proprietà name che appartiene al CIT ip_address.

Oggetto di riconciliazione. L'oggetto è creato dall'adattatore in base ai dati di riconciliazione ricevuti. Questo oggetto deve fare riferimento a un CI esterno ed è utilizzato dal motore di mappatura per eseguire la connessione tra CI esterni e CI del CMDB.

Tipo CI di riconciliazione. Il tipo di CI che rappresentano oggetti di riconciliazione. Questi CI devono essere memorizzati sia nel CMDB sia nei repository di dati esterni.

Motore di mappatura. Un componente che identifica le relazioni tra CI di diversi repository di dati tra i quali esiste una relazione virtuale. L'identificazione viene eseguita riconciliando gli oggetti di riconciliazione di CMDB e gli oggetti di riconciliazione dei CI esterni.

Motore di mappatura

Il framework di federazione utilizza il motore di mappatura per calcolare la query TQL federata. Il motore di mappatura esegue la connessione tra CI ricevuti da diversi repository di dati e connessi tramite relazioni virtuali. Il motore di mappatura fornisce inoltre dati di riconciliazione per la relazione virtuale. Un'estremità della relazione virtuale deve riferirsi a CMDB. Tale estremità è di tipo reconciliation. Per il calcolo dei due sottografici, è possibile avviare una relazione virtuale da qualsiasi nodo estremità.

Adattatore federato

L'adattatore federato porta due tipi di dati dai repository di dati esterni: dati CI esterni e oggetti di riconciliazione che appartengono ai CI esterni.

- **Dati CI esterni.** I dati esterni che non esistono nel CMDB. Si tratta dei dati di destinazione del repository di dati esterno.
- **Dati oggetto di riconciliazione.** I dati ausiliari che vengono utilizzati dal framework di federazione per connettere i CI del CMDB e i dati esterni. Ciascun oggetto di riconciliazione deve riferirsi a un CI esterno. Il tipo di oggetto di riconciliazione è il tipo (o sottotipo) di una delle estremità della relazione virtuale dalle quali i dati vengono recuperati. Gli oggetti di riconciliazione devono essere compatibili con i dati di riconciliazione ricevuti dall'adattatore. L'oggetto di riconciliazione può essere di tre tipi: IdReconciliationObject, PropertyReconciliationObject o TopologyReconciliationObject.

Nelle interfacce basate su DataAdapter (DataAdapter, PopulateDataAdapter e PopulateChangesDataAdapter), la riconciliazione è richiesta come parte della definizione della query.

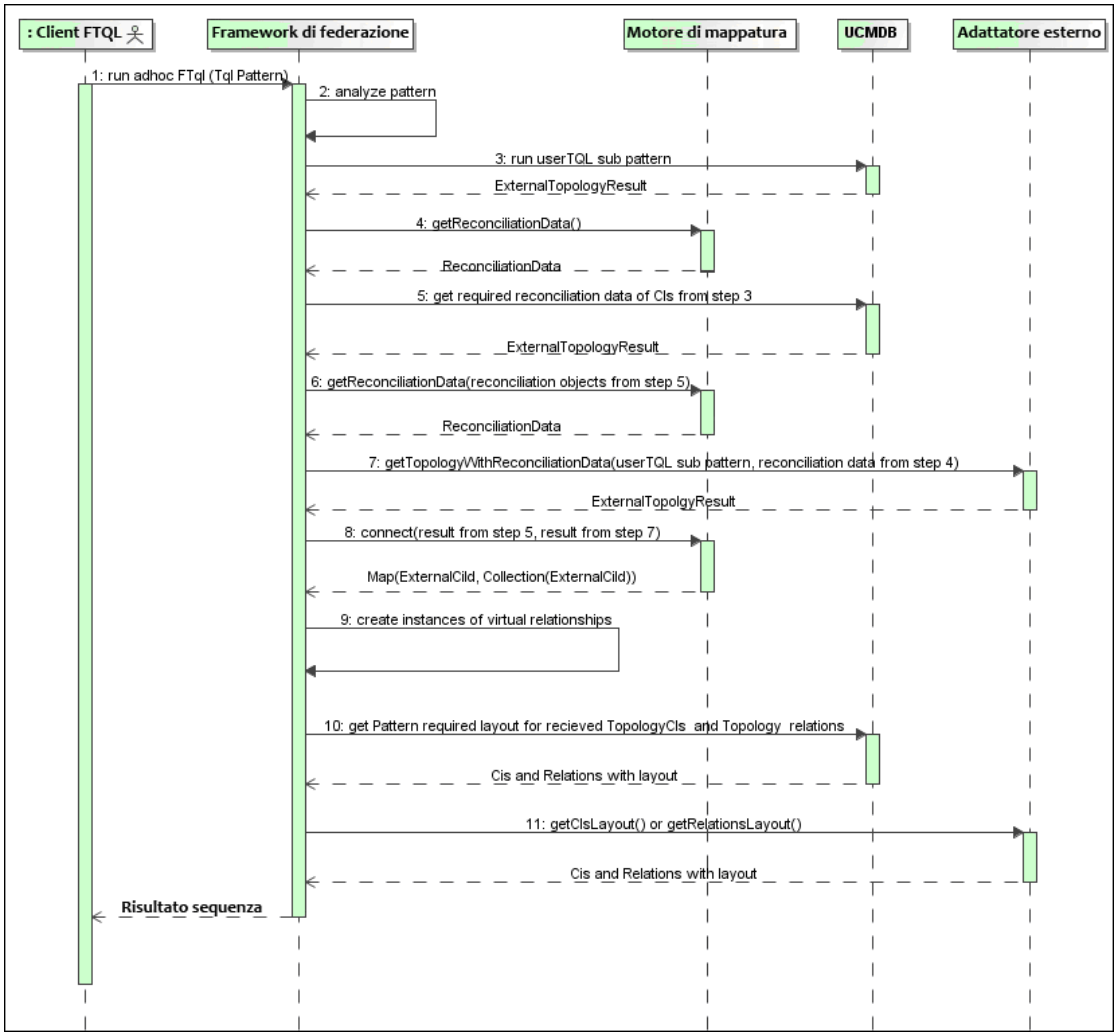
Diagrammi di flusso

I seguenti diagrammi illustrano le interazioni tra il framework di federazione, UCMDB, l'adattatore e il motore di mappatura. La query TQL federata dei diagrammi di esempio ha solo una relazione virtuale e pertanto solo UCMDB e un repository di dati esterno sono coinvolti nella query TQL federata.

Nel primo diagramma il calcolo comincia in UCMDB e il secondo diagramma nell'adattatore esterno. Ciascun passaggio nel diagramma include i riferimenti alla chiamata del metodo appropriato dell'interfaccia del motore di mappatura o adattatore.

Il calcolo inizia all'estremità di HP Universal CMDB

Il seguente diagramma di sequenza illustra l'interazione tra il framework di federazione, UCMDB, l'adattatore e il motore di mappatura. La query TQL federata nel diagramma di esempio ha solo una relazione virtuale e pertanto solo UCMDB e un repository di dati esterno sono coinvolti nella query TQL federata.

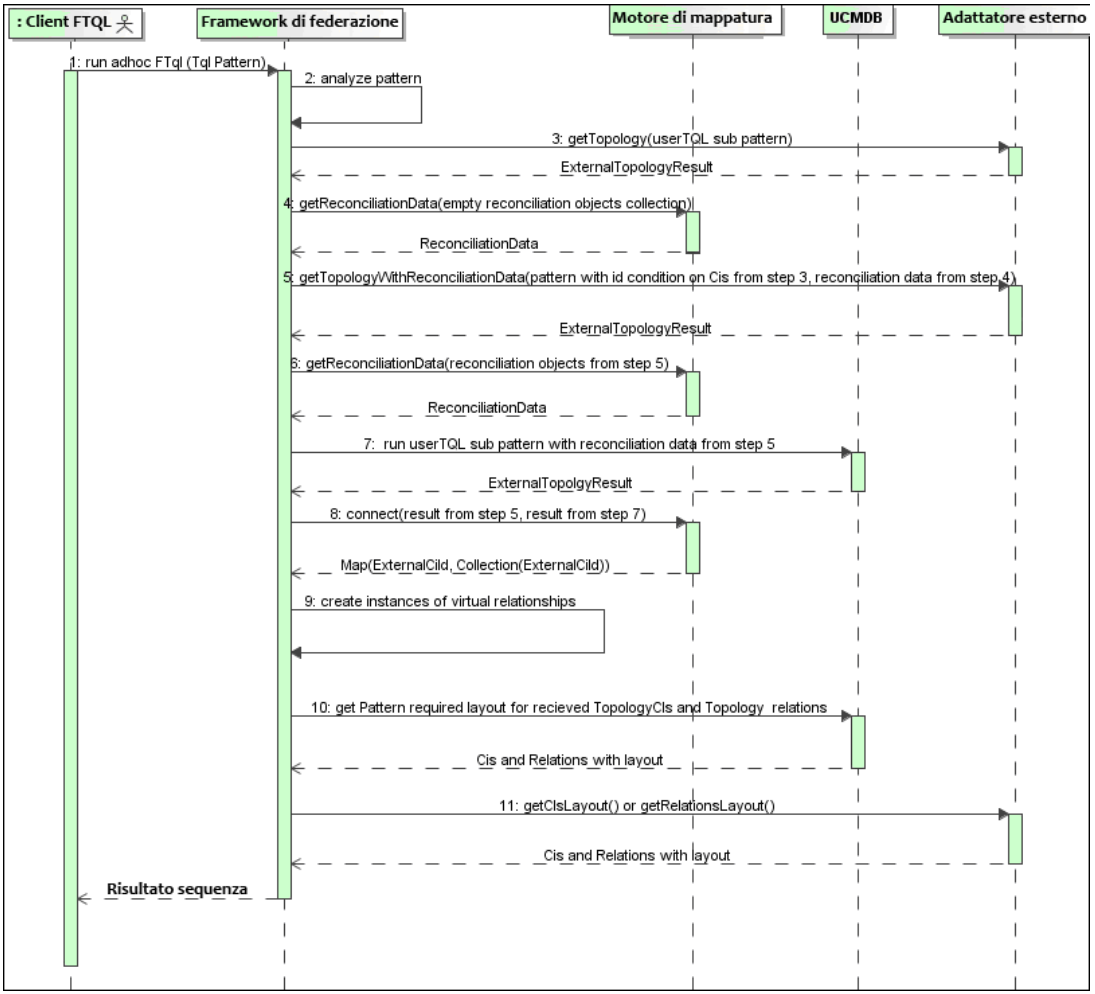


I numeri in questa immagine sono spiegati di seguito:

Numero	Spiegazione
1	Il framework di federazione riceve una chiamata per un calcolo della TQL federata.
2	Il framework di federazione analizza l'adattatore, trova la relazione virtuale e suddivide la TQL originale in due sottoadattatori, uno per UCMDB uno per il repository di dati esterno.
3	Il framework di federazione richiede la topologia della TQL secondaria da UCMDB.
4	Dopo aver ricevuto i risultati della topologia, il framework di federazione chiama il motore di mappatura adeguato per la relazione virtuale corrente e richiede i dati di riconciliazione. Il parametro reconciliationObject è vuoto in questa fase, nessuna condizione viene aggiunta ai dati di riconciliazione in questa chiamata. I dati di riconciliazione restituiti definiscono quali dati sono necessari per creare una corrispondenza tra i CI di riconciliazione in UCMDB e il repository di dati esterno. I dati di riconciliazione possono essere di uno dei tipi seguenti: ➤ **IdReconciliationData.** I CI vengono riconciliati in base ai relativi ID. ➤ **PropertyReconciliationData.** I CI vengono riconciliati in base alle proprietà di uno dei CI. ➤ **TopologyReconciliationData.** I CI vengono riconciliati in base alla topologia (ad esempio, per riconciliare CI nodo, è richiesto anche l'indirizzo IP di **IP**).
5	Il framework di federazione richiede i dati di riconciliazione per i CI delle estremità della relazione virtuale ricevuti al passaggio 3 da UCMDB.
6	Il framework di federazione chiama il motore di mappatura per il recupero dei dati di riconciliazione. In questo stato (in contrasto con il passaggio 3), il motore di mappatura riceve gli oggetti di riconciliazione dal passaggio 5 come parametri. Il motore di mappatura traduce l'oggetto di riconciliazione ricevuto nella condizione dei dati di riconciliazione.

Numero	Spiegazione
7	Il framework di federazione richiede la topologia della TQL secondaria dal repository di dati esterno. L'adattatore esterno riceve i dati di riconciliazione dal passaggio 6 come un parametro.
8	Il framework di federazione chiama il motore di mappatura per eseguire la connessione tra i risultati ricevuti. Il parametro firstResult è il risultato della topologia esterna ricevuto da UCMDB nel passaggio 5 e il parametro secondResult è il risultato della topologia esterna ricevuto dall'adattatore esterno nel passaggio 7. Il motore di mappatura restituisce una mappa in cui l'ID CI esterno del primo repository di dati (UCMDB in questo caso) viene mappato agli ID CI esterni del secondo repository di dati (esterno).
9	Per ciascuna mappatura, il framework di federazione crea una relazione virtuale.
10	Dopo il calcolo dei risultati della query TQL federata (solo nella fase della topologia), il framework di federazione recupera il layout TQL originale per le relazioni e i CI risultanti dai repository di dati appropriati.

Il calcolo inizia all'estremità dell'adattatore esterno



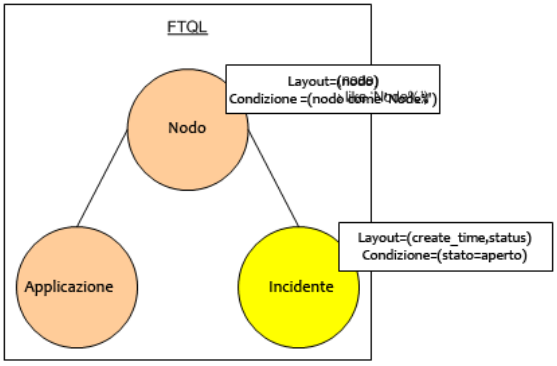
I numeri in questa immagine sono spiegati di seguito:

Numero	Spiegazione
1	Il framework di federazione riceve una chiamata per un calcolo della TQL federata.
2	Il framework di federazione analizza l'adattatore, trova la relazione virtuale e suddivide la TQL originale in due sottoadattatori, uno per UCMDB uno per il repository di dati esterno.
3	Il framework di federazione richiede la topologia della TQL secondaria dall'adattatore esterno. Il parametro ExternalTopologyResult restituito non dovrebbe contenere alcun oggetto di riconciliazione, poiché i dati di riconciliazione non fanno parte della richiesta.
4	Dopo aver ricevuto i risultati della topologia, il framework di federazione chiama il motore di mappatura appropriato per la relazione virtuale corrente e richiede i dati di riconciliazione. Il parametro reconciliationObjects è vuoto in questo stato, nessuna condizione viene aggiunta ai dati di riconciliazione in questa chiamata. I dati di riconciliazione restituiti definiscono quali dati sono necessari per creare una corrispondenza tra i Ci di riconciliazione in UCMDB e il repository di dati esterno. I dati di riconciliazione possono essere di uno dei tre tipi seguenti: ➤ **IdReconciliationData.** I CI vengono riconciliati in base ai relativi ID. ➤ **PropertyReconciliationData.** I CI vengono riconciliati in base alle proprietà di uno dei CI. ➤ **TopologyReconciliationData.** I CI vengono riconciliati in base alla topologia (ad esempio, per riconciliare CI nodo, è richiesto anche l'indirizzo IP di **IP**).
5	Il framework di federazione richiede gli oggetti di riconciliazione per i CI ricevuti nel passaggio 3 dal repository di dati esterno. Il framework di federazione chiama il metodo **getTopologyWithReconciliationData()** nell'adattatore esterno, dove la topologia richiesta è una topologia a un nodo con i CI ricevuti nel passaggio 3 come la condizione ID e i dati di riconciliazione del passaggio 4.

Numero	Spiegazione
6	Il framework di federazione chiama il motore di mappatura per il recupero dei dati di riconciliazione. In questo stato (in contrasto con il passaggio 3), il motore di mappatura riceve gli oggetti di riconciliazione dal passaggio 5 come parametri. Il motore di mappatura traduce l'oggetto di riconciliazione ricevuto nella condizione dei dati di riconciliazione.
7	Il framework di federazione richiede la topologia della TQL secondaria con i dati di riconciliazione del passaggio 6 da UCMDB.
8	Il framework di federazione chiama il motore di mappatura per eseguire la connessione tra i risultati ricevuti. Il parametro firstResult è il risultato della topologia esterna ricevuto dall'adattatore esterno al passaggio 5 e il parametro secondResult è il risultato della topologia esterna ricevuto da UCMDB al passaggio 7. Il motore di mappatura restituisce una mappa in cui l'ID CI esterno del primo repository di dati (il repository di dati esterno in questo caso) viene mappato agli ID CI esterni del secondo repository di dati (UCMDB).
9	Per ciascuna mappatura, il framework di federazione crea una relazione virtuale.
10	Dopo il calcolo dei risultati della query TQL federata (solo nella fase della topologia), il framework di federazione recupera il layout TQL originale per le relazioni e i CI risultanti dai repository di dati appropriati.

Esempio di flusso del framework di federazione per le query TQL federate

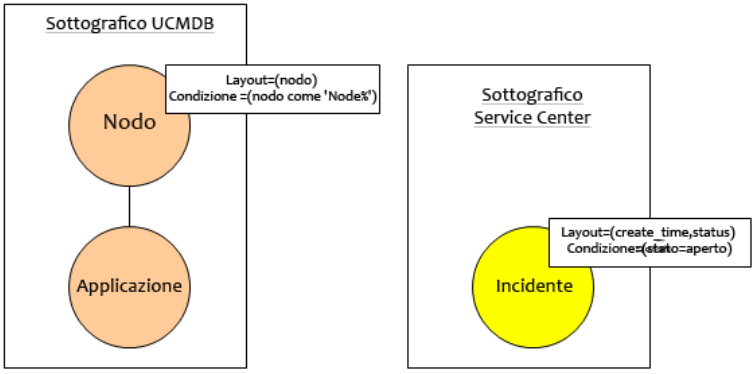
Questo esempio spiega come visualizzare tutti gli incidenti aperti su nodi specifici. ServiceCenter è il repository di dati esterno. Le istanze del nodo sono memorizzate in UCMDB e le istanze degli incidenti sono memorizzate in ServiceCenter. Si presuppone che per connettere le istanze degli incidenti al nodo appropriato siano richieste le proprietà node e ip_address dell'host e dell'IP. Si tratta di proprietà di riconciliazione che identificano i nodi da ServiceCenter in UCMDB.



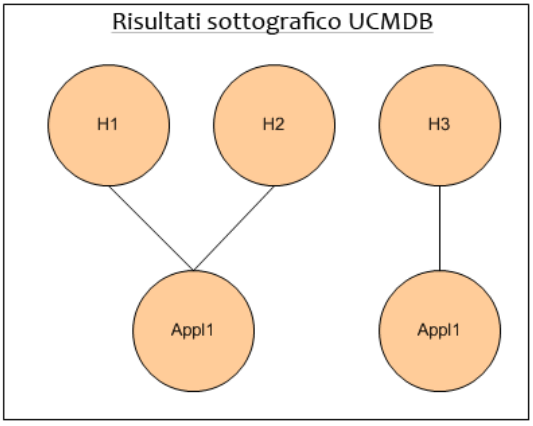
Nota: per la federazione dell'attributo, viene chiamato il metodo **getTopology** dell'adattatore. I dati di riconciliazione sono adattati nella TQL utente (in questo caso, l'elemento CI).

---

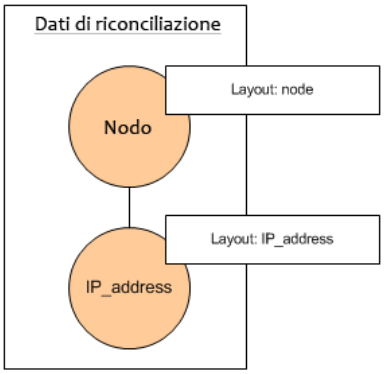
1 Dopo aver analizzato l'adattatore, il framework di federazione riconosce la relazione virtuale tra Node e Incident e suddivide la query TQL federata in due sottografici:



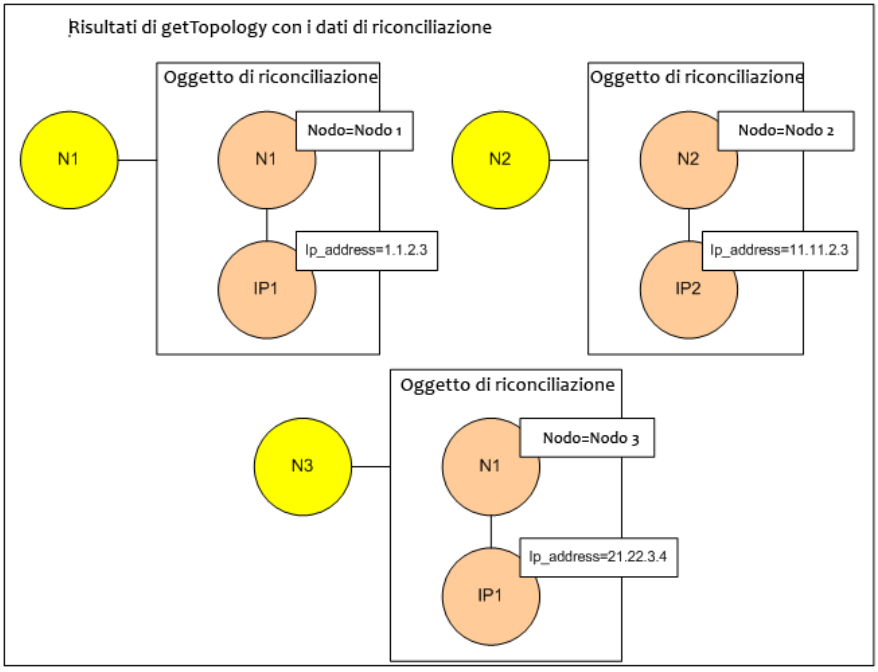
2 Il framework di federazione esegue il sottografico UCMDB per richiedere la topologia e riceve i risultati seguenti:



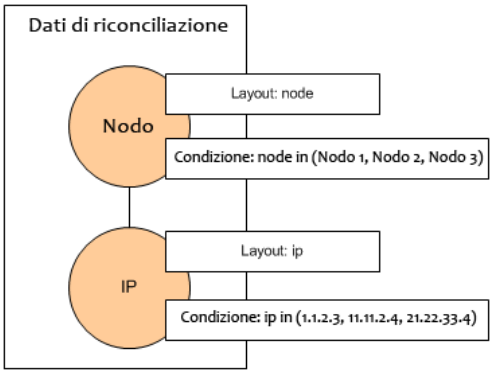
3 Il framework di federazione richiede, dal motore di mappatura appropriato, i dati di riconciliazione per il primo repository di dati (UCMDB) che contiene le informazioni necessarie per la connessione tra i dati ricevuti da due repository di dati. I dati di riconciliazione in questo caso sono:



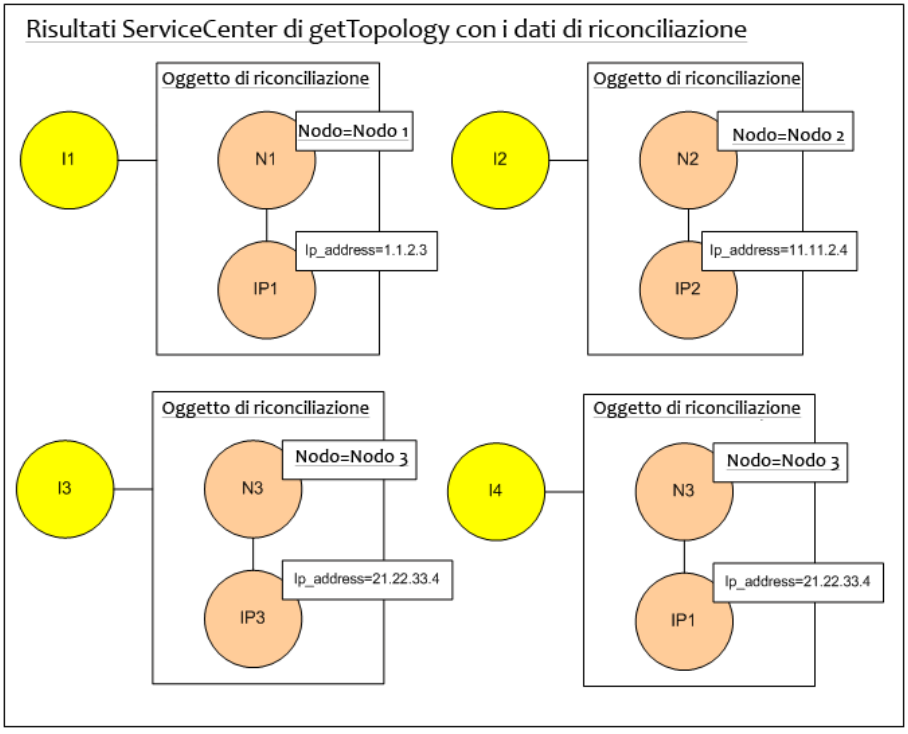
4 Il framework di federazione crea una query topologia a un nodo con le condizioni Node e ID del precedente risultato (nodo in H1, H2, H3) ed esegue questa query con i dati di riconciliazione richiesti su UCMDB. Il risultato include i CI nodo che riguardano la condizione ID e l'oggetto di riconciliazione appropriato per ciascun CI:



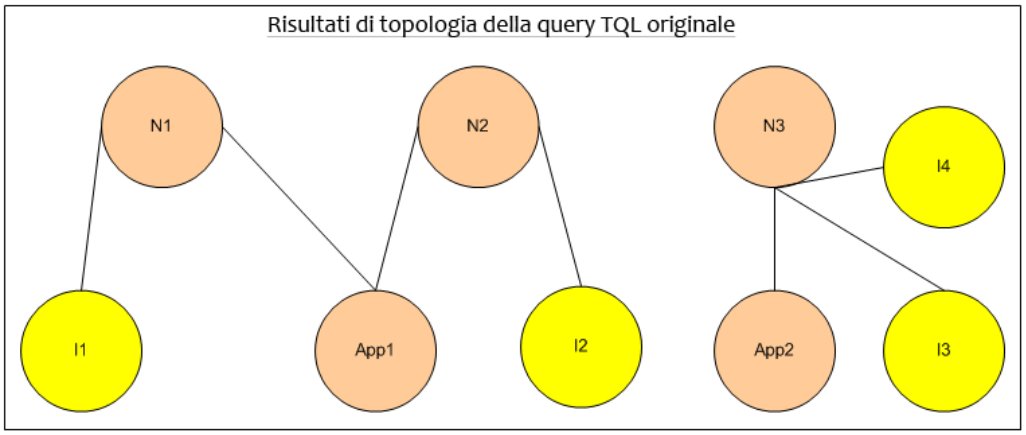
5 I dati di riconciliazione per ServiceCenter devono contenere una condizione per node e ip derivata dagli oggetti di riconciliazione ricevuti da UCMDB:



6 Il framework di federazione esegue il sottografico di ServiceCenter con i dati di riconciliazione per richiedere la topologia e gli oggetti di riconciliazione appropriati, ricevendo i risultati seguenti:



7 Il risultato dopo la connessione nel motore di mappatura e la creazione della relazione virtuale è:



8 Il framework di federazione richiede il layout TQL originale per le istanze ricevute da UCMDB e ServiceCenter.

Flusso del framework di federazione per il popolamento

In questa sezione vengono trattati i seguenti argomenti:

➤ "Definizioni e termini" a pag. 243

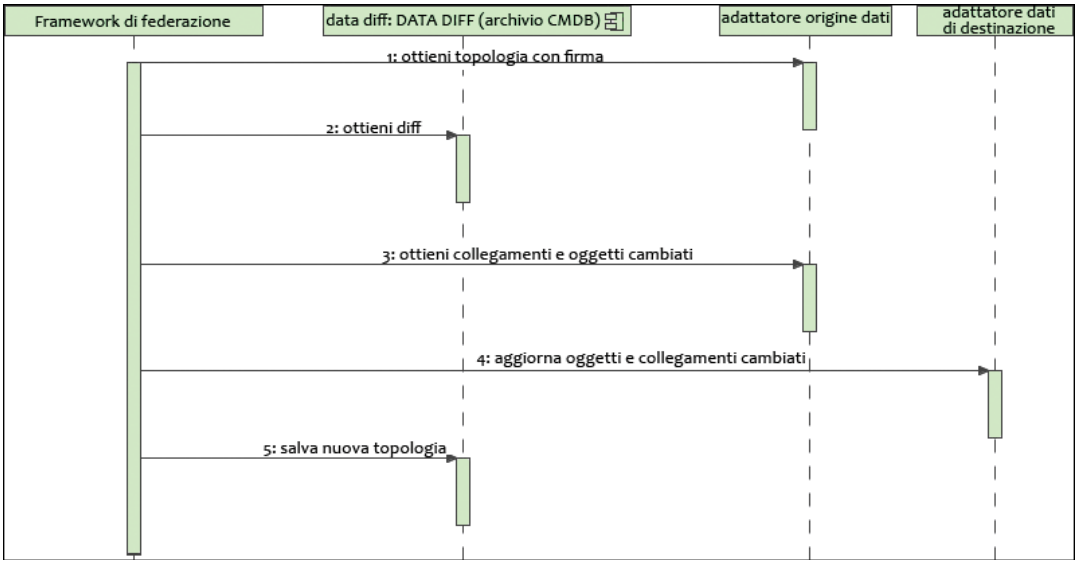
➤ "Diagramma di flusso" a pag. 244

Definizioni e termini

Firma. Denota lo stato delle proprietà nel CI. Se vengono apportati dei cambiamenti ai valori delle proprietà in un CI, sarà necessario cambiare anche la firma CI. La firma CI aiuta a rilevare se un CI è stato cambiato senza aver recuperato e confrontato tutte le proprietà CI. Il CI e la relativa firma sono forniti dall'adattatore appropriato. L'adattatore è responsabile del cambiamento della firma CI in caso di modifica delle proprietà CI.

Diagramma di flusso

Il seguente diagramma sequenza illustra l'interazione tra il framework di federazione e gli adattatori di origine e di destinazione in un flusso di popolamento:



1. Il framework di federazione riceve la topologia per il risultato della query dall'adattatore di origine. L'adattatore riconosce la query dal nome e la esegue sul repository di dati esterno. Il risultato della topologia contiene l'ID e la firma per ciascun CI e relazione nel risultato. L'ID è l'ID logico che definisce il CI come univoco nel repository di dati esterno. La firma deve essere modificata se viene modificato il CI o la relazione.
2. Il framework di federazione utilizza le firme per confrontare i risultati della query topologia recentemente ricevuti con quelli salvati e per determinare quali CI sono stati cambiati.
3. Dopo che il framework di federazione trova le relazioni e i CI cambiati, chiama l'adattatore di origine con gli ID delle relazioni e dei CI cambiati come un parametro per recuperare il loro layout completo.
4. Il framework di federazione invia l'aggiornamento all'adattatore di destinazione. Quest'ultimo aggiorna l'origine dati esterna con i dati ricevuti.
5. Dopo l'aggiornamento, il framework di federazione salva l'ultimo risultato della query.

Interfacce dell'adattatore

In questa sezione vengono trattati i seguenti argomenti:

Definizioni e termini

La relazione esterna. La relazione tra due tipi CI esterni supportati dallo stesso adattatore.

Interfacce adattatore per query TQL federate

Utilizzare l'interfaccia adattatore appropriata per ciascun adattatore, come di seguito indicato.

Viene utilizzata un'interfaccia topologia **One Node** quando l'adattatore non supporta alcuna relazione esterna; pertanto, l'adattatore non dovrebbe mai ricevere una richiesta con più di un CI esterno. Tutte le interfacce OneNode sono create per semplificare il flusso di lavoro; per i casi in cui si richiede l'utilizzo di una query più completa, usare l'interfaccia DataAdapter.

Obsoleta a partire da UCMDB 9.00: Interfaccia topologia sequenza

Un'**interfaccia DataAdapter** è utilizzata per definire gli adattatori che supportano query federate complesse. La richiesta di riconciliazione in questi adattatori è parte di un singolo parametro QueryDefinition. Tali adattatori potrebbero inoltre essere utilizzati per il popolamento.

Interfacce OneNode

Le seguenti interfacce hanno diversi tipi di dati di riconciliazione:

- **OneNodeTopologyIdReconciliationDataAdapter.** Utilizzare se l'adattatore supporta una **TQL a nodo singolo** e la riconciliazione tra repository di dati viene calcolata dall' ID.
- **OneNodeTopologyPropertyReconciliationDataAdapter.** Utilizzare se l'adattatore supporta una **TQL a nodo singolo** e la riconciliazione tra repository di dati viene eseguita dalle proprietà di un CI.
- **OneNodeTopologyDataAdapter.** Utilizzare se l'adattatore supporta una **TQL a nodo singolo** e la riconciliazione tra repository di dati viene eseguita dalla topologia.

Interfacce adattatore dati

- **DataAdapter.** Utilizzare questo adattatore per supportare query TQL federate complesse. Consente la massima diversità.
- **PopulateDataAdapter.** Utilizzare questo adattatore per supportare query TQL federate complesse e flussi di popolamento. In un flusso di popolamento, questo adattatore recupera l'intero set di dati e permette alla sonda di filtrare la differenza dall'ultima esecuzione del processo.
- **PopulateChangesDataAdapter.** Utilizzare questo adattatore per supportare query TQL federate complesse e flussi di popolamento. In un flusso di popolamento, questo adattatore supporta il recupero dei soli cambiamenti verificatisi dall'ultima esecuzione del processo.

Interfacce topologia sequenza (obsolete a partire da UCMDB 9.00)

Le seguenti interfacce hanno diversi tipi di dati di riconciliazione:

- **PatternTopologyIdReconciliationDataAdapter.** Utilizzare se l'adattatore supporta una **TQL complessa** e la riconciliazione tra repository di dati viene eseguita dall' ID.
- **PatternTopologyPropertyReconciliationDataAdapter.** Utilizzare se l'adattatore supporta una **TQL complessa** e la riconciliazione tra repository di dati viene eseguita da proprietà a nodo singolo.
- **PatternTopologyDataAdapter.** Utilizzare se l'adattatore supporta una **TQL complessa** e la riconciliazione tra repository di dati viene eseguita dalla topologia.

Interfacce aggiuntive

- **SortResultDataAdapter.** Utilizzare se è possibile ordinare i CI risultanti nel repository di dati esterno.
- **FunctionalLayoutDataAdapter.** Utilizzare se è possibile calcolare il layout funzionale nel repository di dati esterno.

Interfacce adattatore per la sincronizzazione

- **SourceDataAdapter.** Utilizzare per gli adattatori di origine nei flussi di popolamento.
- **TargetDataAdapter.** Utilizzare per gli adattatori di destinazione nei flussi di invio dati.

Compiti

Aggiungere un adattatore per una nuova origine dati esterna

Questo compito spiega come definire un adattatore per supportare una nuova origine dati esterna.

Questo compito include i passaggi seguenti:

- "Prerequisiti" a pag. 247
- "Definire relazioni valide per le relazioni virtuali" a pag. 248
- "Definire una configurazione adattatore" a pag. 249
- "Definire le classi supportate" a pag. 252
- "Implementare l'adattatore" a pag. 253
- "Definire le regole di riconciliazione o implementare il motore di mappatura" a pag. 253
- "Aggiungere i Jar richiesti per l'implementazione sul percorso classe" a pag. 253
- "Distribuire l'adattatore" a pag. 254
- "Aggiornare l'adattatore" a pag. 255

1 Prerequisiti

Le classi dell'adattatore supportate dal modello per CI e relazioni nel modello dati UCMDB: lo sviluppatore dell'adattatore ha il compito di:

- conoscere la gerarchia dei tipi CI UCMDB per comprendere la relazione esistente tra CIT esterni e CIT UCMDB
- modellare i CIT esterni nel modello classe UCMDB
- aggiungere le definizione per i nuovi tipi CI e le relative relazioni

➤ definire relazioni valide nel modello di classe UCMDB per le relazioni valide tra le classi interne dell'adattatore. (I CIT possono essere posizionati a qualsiasi livello della struttura del modello classe UCMDB).

La modellazione deve essere la stessa, indipendentemente dal tipo di federazione (in tempo reale o replica). Per i dettagli sull'aggiunta di nuove definizioni CIT nel modello di classe UCMDB consultare"Utilizzo del selettore CI" nella *Guida alla modellazione di HP Universal CMDB*.

Per consentire all'adattatore di supportare attributi federati su CIT, aggiungere questo CIT alle classi supportate con gli attributi supportati e la regola di riconciliazione per questo CIT.

2 Definire relazioni valide per le relazioni virtuali

Nota: questa sezione riguarda esclusivamente la federazione.

Per recuperare i CIT federati connessi ai CIT locali di CMDB, deve esistere una definizione di collegamenti validi tra i due CIT nel CMDB.

a Creare un file XML dei collegamenti validi contenente tali collegamenti (se non sono già esistenti).

b Aggiungere il file XML dei collegamenti al pacchetto dell'adattatore nella cartella **\validlinks**. Per i dettagli consultare "Gestione pacchetti" nella *Guida all'amministrazione di HP Universal CMDB*.

Esempio di definizione relazione valida:

Nell'esempio seguente, la relazione di tipo containment tra le istanze di tipo node e le istanze di tipo myclass1 è una definizione di relazione valida.

```
<Valid-Links>
    <Valid-Link>
        <Class-Ref class-name="containment"/>
        <End1 class-name="node"/>
        <End2 class-name="myclass1"/>
        <Valid-Link-Qualifiers/>
    </Valid-Link>
</Valid-Links>
```

3 Definire una configurazione adattatore

a Passare a **Gestione adattatore**.

b Fare clic sul pulsante **Crea nuova risorsa**.

c Nella finestra di dialogo Nuovo adattatore, selezionare **Integrazione** e **Adattatore Java**.

d Fare clic con il pulsante destro del mouse sull'adattatore creato e selezionare **Modifica origine adattatore** dal menu di scelta rapida.

e Modificare i seguenti tag XML:

```
<pattern xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="newAdapterIdName"
xsi:noNamespaceSchemaLocation="../../Patterns.xsd" description="Adapter Description"
schemaVersion="9.0" displayName="New Adapter Display Name">
    <deletable>true</deletable>
    <discoveredClasses>
        <discoveredClass>link</discoveredClass>
        <discoveredClass>object</discoveredClass>
    </discoveredClasses>
    <taskInfo className="com.hp.ucmdb.discovery.probe.services.dynamic.core.AdapterService">
        <params
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.AdapterServiceParams"
enableAging="true" enableDebugging="false" enableRecording="false" autoDeleteOnErrors="success"
recordResult="false" maxThreads="1" patternType="java_adapter" maxThreadRuntime="25200000">
            <className >com.yourCompany.adapter.MyAdapter.MyAdapterClass</className>
        </params>
```

```
        <destinationInfo className="com.hp.ucmdb.discovery.probe.tasks.BaseDestinationData">
            <!-- check -->
            <destinationData name="adapterId" description="">${ADAPTER.adapter_id}</
destinationData>
            <destinationData name="attributeValues" description="">${SOURCE.attribute_values}</
destinationData>
            <destinationData name="credentialsId" description="">${SOURCE.credentials_id}</
destinationData>
            <destinationData name="destinationId" description="">${SOURCE.destination_id}</
destinationData>
        </destinationInfo>
        <resultMechanism isEnabled="true">
            <autoDeleteCITs isEnabled="true">
                <CIT>link</CIT>
                <CIT>object</CIT>
            </autoDeleteCITs>
        </resultMechanism>
    </taskInfo>
    <adapterInfo>
        <adapter-capabilities>
            <support-federated-query>
                <!--<supported-classes/> <!—see the section about supported classes-->
            <topology>
                <pattern-topology /> <!—or <one-node-topology> -->
            </topology>
            </support-federated-query>
            <!--<support-replicatioin-data>
            <source>
                <changes-source/>
            </source>
    <target/>
        </adapter-capabilities>
        <default-mapping-engine/>
            <queries />
    <removedAttributes />
        <full-population-days-interval>-1</full-population-days-interval>
    </adapterInfo>
    <inputClass>destination_config</inputClass>
    <protocols />
```

```
<parametri>
    <!--The description attribute may be written in simple text or HTML.-->
    <!--The host attribute is treated as a special case by UCMDB-->
    <!--and will automatically select the probe name (if possible)-->
    <!--according to this attribute's value.-->
    <parameter name="credentialsId" description="Special type of property, handled by UCMDB for
credentials menu" type="integer" display-name="Credentials ID" mandatory="true" order-index="12" />
    <parameter name="host" description="The host name or IP address of the remote machine"
type="string" display-name="Hostname/IP" mandatory="false" order-index="10" />
    <parameter name="port" description="The remote machine's connection port" type="integer"
display-name="Port" mandatory="false" order-index="11" />
    </parameters>
<parameter name="myatt" description="is my att true?" type="string" display-name="My Att"
mandatory="false" order-index="15" valid-values="True;False"/>True</parameters>
    <collectDiscoveredByInfo>true</collectDiscoveredByInfo>
    <integration isEnabled="true">
        <category >My Category</category>
    </integration>
    <overrideDomain>${SOURCE.probe_name}</overrideDomain>
    <inputTQL>
        <resource:XmlResourceWrapper xmlns:resource="http://www.hp.com/ucmdb/1-0-0/
ResourceDefinition" xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:tql="http://
www.hp.com/ucmdb/1-0-0/TopologyQueryLanguage">
            <resource xsi:type="tql:Query" group-id="2" priority="low" is-live="true" owner="Input TQL"
name="Input TQL">
                <tql:node class="adapter_config" id="-11" name="ADAPTER" />
                <tql:node class="destination_config" id="-10" name="SOURCE" />
                <tql:link to="ADAPTER" from="SOURCE" class="fcmdb_conf_aggregation" id="-12"
name="fcmdb_conf_aggregation" />
            </resource>
        </resource:XmlResourceWrapper>
    </inputTQL>
    <permissions />
</pattern>
```

Per i dettagli sui tag XML consultare "Proprietà e tag di configurazione XML" a pag. 259.

4 Definire le classi supportate

Definire le classi supportate da entrambi i codici adattatore implementando il metodo *getSupportedClasses()* o utilizzando il file XML della sequenza.

```
<supported-classes>
    <supported-class name="HistoryChange" is-derived="false"
is-reconciliation-supported=”false” federation-not-supported=”false”
is-id-reconciliation-supported=”false”>
    <supported-conditions>
        <attribute-operators attribute-name="change_create_time">
            <operator>GREATER</operator>
            <operator>LESS</operator>
            <operator>GREATER_OR_EQUAL</operator>
            <operator>LESS_OR_EQUAL</operator>
            <operator>CHANGED_DURING</operator>
        </attribute-operators>
    </supported-conditions>
 </supported-class>
```

name	Il nome del tipo CI.
is-derived	Specifica se la definizione include tutti i figli in eredità
is-reconciliation-supported	Specifica se questa classe viene utilizzata per la riconciliazione
is-id-reconciliation-supported	Specifica se questa classe viene utilizzata per id-reconciliation
federation-not-supported	Specifica se questo CIT non deve essere autorizzato alla federazione (bloccando alcuni CIT, ad esempio, un CIT definito solo per la federazione)
<supported-conditions>	Specifica le condizioni supportate su ciascun attributo

5 Implementare l'adattatore

Selezionare la classe di implementazione adattatore corretta in base alla relative capacità definite. Tale classe implementa le interfacce appropriate in base alle capacità definite.

6 Definire le regole di riconciliazione o implementare il motore di mappatura

Se l'adattatore supporta query TQL federate, sono disponibili tre opzioni per definire il motore di mappatura:

- Utilizzare il motore di mappatura predefinito di CMDB 9.0x , che usa le regole interne di riconciliazione di CMDB per la mappatura. Per utilizzarlo, lasciare vuoto il tag XML **<default-mapping-engine/>**.

  Per i dettagli consultare "File reconciliation_types.txt" a pag. 193.

- Utilizzare il motore di mappatura di CMDB 8.0x. A tale scopo, utilizzare il seguente tag XML:
  <default-mapping-engine>com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine</default-mapping-engine>

  Per i dettagli consultare "File reconciliation_rules.txt (per la contabilità inversa)" a pag. 194.

- Scrivere il motore di mappatura implementando l'interfaccia motore di mappatura e posizionando il JAR con il resto del codice adattatore. A tale scopo, utilizzare il seguente tag XML:
  <default-mapping-engine>com.yourcompany.map.MyMappingEngine</default-mapping-engine>

7 Aggiungere i Jar richiesti per l'implementazione sul percorso classe

Per implementare le classi, aggiungere il file **federation_api.jar** al percorso classe dell'editor di codice.

8 Distribuire l'adattatore

a Distribuire il pacchetto dell'adattatore. Per i dettagli generali sulla distribuzione di un pacchetto consultare "Gestione pacchetti" nella *Guida all'amministrazione di HP Universal CMDB*.

Il pacchetto deve contenere le seguenti entità:

- Definizione nuovi CIT (facoltativa):

  Utilizzata solo se l'adattatore supporta nuovi tipi CI non ancora esistenti in UCMDB.

  Le definizioni dei nuovi CIT si trovano nella cartella class nel pacchetto.

- Definizione nuovi tipi di dati (facoltativa):

  Utilizzata solo se i nuovi CIT richiedono nuovi tipi di dati.

  Le definizioni dei nuovi tipi di dati si trovano nella cartella typedef del pacchetto.

- Definizione nuove relazioni valide (facoltativa):

  Utilizzata solo se l'adattatore supporta la TQL federata.

  Le definizioni delle nuove relazioni valide si trovano nella cartella validlinks nel pacchetto.

- Il file XML della configurazione della sequenza deve trovarsi nella cartella discoveryPatterns nel pacchetto.

- **Descrittore.** Definisce le definizioni del pacchetto.

- Posizionare le classi compilate (normalmente in un file jar) nel pacchetto nella cartella **adapterCode\<adapter id>**.

Nota: il nome della cartella adapter id ha lo stesso valore della configurazione adattatore.

- Se viene creato il file di configurazione, è necessario posizionare il file nel pacchetto nella cartella **adapterCode\<adapter id>**.

9 Aggiornare l'adattatore

È possibile eseguire cambiamenti su qualunque file non binario dell'adattatore nel modulo Gestione adattatore. I cambiamenti ai file di configurazione nel modulo Gestione adattatore fanno sì che l'adattatore si ricarichi con le nuove configurazioni.

Gli aggiornamenti possono inoltre essere eseguiti modificando i file nel pacchetto (sia binari sia non binari) e distribuendo successivamente il pacchetto tramite Gestione pacchetti. Per i dettagli consultare "Distribuire un pacchetto" nella *Guida all'amministrazione di HP Universal CMDB*.

Implementare il motore di mappatura

La configurazione del motore di mappatura dipende dal motore che si sta utilizzando.

Questo compito include i passaggi seguenti:

- "Configurare il file reconciliation_types.txt (per il motore di mappatura predefinito di UCMDB 9.0x)" a pag. 255
- "Configurare il file reconciliation_rules.txt (per il motore di mappatura predefinito di UCMDB 8.0x)" a pag. 256

1 Configurare il file reconciliation_types.txt (per il motore di mappatura predefinito di UCMDB 9.0x)

Il file viene utilizzato per definire i tipi CI utilizzati per la riconciliazione nell'adattatore.

Scrivere ciascun tipo CI per la riconciliazione su una riga singola, come di seguito indicato:

```
node
business_application
```

Posizionare il file nel pacchetto adattatore nella cartella **adapterCode**\<**AdapterID**>\ **META-INF**\.

2 Configurare il file reconciliation_rules.txt (per il motore di mappatura predefinito di UCMDB 8.0x)

Il file viene utilizzato per configurare le regole di riconciliazione. Ciascuna riga del file rappresenta una regola. Ad esempio:

```
reconcilition_type[node] expression[^node.name OR ip_address.name]
end1_type[node] end2_type[ip_address] link_type[containment]
```

Il parametro **reconcilition_type** viene riempito con il tipo di CI su cui viene eseguita la riconciliazione (il nome classe UCMDB connesso alla classe federata nel TQL).

Il parametro **expression** è la logica che decide se due oggetti di riconciliazione sono uguali (un oggetto di riconciliazione dal lato UCMDB e l'altro dal lato adattatore federato).

L'espressione è composta da OR e AND.

La convenzione riguardante i nomi degli attributi nella parte dell'espressione è **[className].[attributeName]**. Ad esempio, l'attributo **ip_address** nella classe **ip** è scritto **ip.ip_address**.

È possibile definire corrispondenze ordinate. La corrispondenza ordinata controlla la prima sottoespressione OR. Se i due oggetti di riconciliazione hanno il valore negli attributi della sottoespressione che restituisce false (gli oggetti di riconciliazione non sono uguali), la seconda sottoespressione OR non viene confrontata.

Per una corrispondenza ordinata, utilizzare **ordered expression** invece di **expression**.

L'accento circonflesso (^) viene utilizzato per ignorare la distinzione tra maiuscole e minuscole durante i confronti.

Gli altri parametri (**end1_type**, **end2_type** e **link_type**) vengono utilizzati solo se i dati di riconciliazione contengono due nodi e non solo il nodo del tipo di riconciliazione (i dati di riconciliazione topologici). In questo caso, i dati di riconciliazione sono **end1_type -(link_type)> end2_type**.

Non è necessario aggiungere il relativo layout poiché viene recuperato dall'espressione.

Per eseguire la riconciliazione tramite ID UCMDB, utilizzare **cmdb_id** come nome attributo nell'espressione.

Posizionare il file nel pacchetto adattatore nella cartella **adapterCode**\<**AdapterID**>\ **META-INF**\.

Esempi:

- È possibile aggiungere una regola di riconciliazione solo per un CIT del nodo. Ciò dipende dal fatto che solo i CIT del nodo hanno relazioni valide con i CIT esterni. Ad esempio, viene creata una corrispondenza tra un CI del nodo in CMDB e un CI del nodo in ServiceCenter tramite l'attributo node.name o l'attributo ip_address.name.
- La regola di riconciliazione in questo caso è una regola topologica e l'espressione viene ordinata. La regola esegue i seguenti controlli sui CI oggetto del confronto:
  - Se l'attributo node.name è uguale, la regola corrisponde ai nodi.
  - Se l'attributo node.name non è uguale, la regola non corrisponde ai nodi.
  - Se l'attributo node.name è null in uno dei CI confrontati, la regola controlla l'attributo ip_address.name. Se l'attributo ip_address.name è uguale, la regola corrisponde ai nodi.

Creare un adattatore di esempio

Questo esempio illustra come creare un adattatore di esempio.

Questo compito include i passaggi seguenti:

- "Selezionare la logica dell'adattatore" a pag. 258
- "Caricare il progetto" a pag. 258

1 Selezionare la logica dell'adattatore

Quando si implementa un adattatore, è necessario scegliere la modalità di gestione della logica della condizione nell'implementazione (condizioni proprietà, condizioni ID, condizioni riconciliazione e condizioni collegamento).

a Recuperare tutti i dati nella memoria dell'adattatore consentendo di selezionare o filtrare le istanze CI necessarie.

b Convertire tutte le condizioni nella lingua dell'origine dati consentendo di filtrare e selezionare i dati. Ad esempio:

- ➤ Convertire la condizione in una query SQL.
- ➤ Convertire la condizione in un oggetto filtro API Java.

c Una fase intermedia prevede di filtrare alcuni dati sul servizio remoto, facendo sì che l'adattatore selezioni e filtri i rimanenti.

Nell'esempio MyAdapter, viene utilizzata la logica nel passaggio a.

2 Caricare il progetto

Copiare i file dalla cartella **C:\hp\UCMDB\UCMDBServer\tools\adapter-dev-kit\SampleAdapters** e seguire le istruzioni nei file Leggimi.

Nota: se si utilizza un adattatore con grandi insiemi di dati, potrebbe essere necessario eseguire la memorizzazione nella cache e l'indicizzazione per migliorare la prestazione della federazione.

La documentazione javadoc online è disponibile su:

C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\doc_lib\DevRef_guide\DBAdapterFramework_JavaAPI\index.html

Riferimenti

Proprietà e tag di configurazione XML

id="newAdapterIdName"		Definisce il nome effettivo dell'adattatore. Utilizzato per le ricerche in cartelle e registri
displayName="New Adapter Display Name"		Definisce il nome visualizzato dell'adattatore, così come viene visualizzato nell'interfaccia utente.
<className>…</className>		Definisce l'interfaccia dell'adattatore che implementa la classe Java.
<category >My Category</ category>		Definisce la categoria dell'adattatore.
<parametri>		Definisce le proprietà per la configurazione disponibili nell'interfaccia utente al momento dell'impostazione di un nuovo punto di integrazione.
	name	Il nome della proprietà (principalmente usato in base al codice)
	description	Il suggerimento visualizzato della proprietà
	type	Stringa o intero (utilizzare valori validi con la stringa per Boleano).
	display-name	Il nome della proprietà nell'interfaccia utente.
	mandatory	Specifica se questa proprietà di configurazione è obbligatoria per l'utente.
	order-index	L'ordine di posizionamento della proprietà (small = up)
	valid-values	Un elenco di possibili valori validi separati dai caratteri ‘;’ (ad esempio valid-values="Oracle;SQLServer;MySQL” o valid-values=”True;False”).
<adapterInfo>		Contiene la definizione delle capacità e impostazioni statiche dell'adattatore.
	<support-federated-query>	Definisce questo adattatore come capace di federazione.
	<one-node-topology>	La capacità di federare query con un nodo query federata.

	<pattern-topology>	La capacità di federare query complesse.
	<support-replicatioin-data>	Definisce la capacità di eseguire l'invio dati e i flussi di popolamento.
	<source>	Questo adattatore può essere utilizzato per i flussi di popolamento.
	<changes-source/>	Questo adattatore può essere utilizzato per i flussi dei cambiamenti del popolamento.
	<target>	Questo adattatore può essere utilizzato per i flussi di invio dati.
	<default-mapping-engine>	Consente la definizione di un motore di mappatura per l'adattatore (per impostazione predefinita, l'adattatore utilizza il motore di mappatura predefinito). Per qualsiasi altro motore di mappatura, immettere il nome della classe di implementazione del motore di mappatura (per il motore di mappatura di UCMDB 8.0x utilizzare: com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine)
	<removedAttributes>	Forza la rimozione di attributi specifici dal risultato.
	<full-population-days-interval>	Specifica quando eseguire un processo di popolamento completo invece di un processo differenziale (ogni 'x' giorni). Utilizza il meccanismo di aging con il flusso dei cambiamenti.

7

Sviluppo degli adattatori Push

Questo capitolo comprende:

Concetti

Compiti

Riferimenti

Concetti

Sviluppo degli adattatori Push - Panoramica

L'adattatore Push generico fornisce una piattaforma che consente lo sviluppo rapido di integrazioni che inviano i dati di UCMDB 9.0x a repository di dati esterni (database e applicazioni di terze parti). Lo sviluppo di un'integrazione personalizzata basata su un adattatore Push generico richiede:

- un file di mappatura XML tra i tipi di collegamento CI di UCMDB e gli elementi di dati esterni.
- uno script Jython per inviare gli elementi di dati al repository di dati esterno.

Sincronizzazione differenziale

Se il metodo **DiscoveryMain** nello script Jython sul quale è basato l'adattatore Push restituisce un'istanza **OSHVResult** vuota, l'adattatore non supporterà la sincronizzazione differenziale. Ciò significa che, anche quando viene eseguito un processo di sincronizzazione differenziale, in effetti viene eseguita una sincronizzazione completa. Pertanto, non è possibile aggiornare o rimuovere alcun dato sul sistema remoto poiché tutti i dati vengono aggiunti al CMDB durante ciascuna sincronizzazione.

Affinché l'adattatore Push supporti la sincronizzazione differenziale, la funzione **DiscoveryMain** deve restituire un oggetto che implementi l'interfaccia **DataPushResults** contenente le mappature tra gli ID che lo script Jython riceve dall'XML e gli ID che lo script Jython crea sul computer remoto. Questi ultimi ID sono di tipo ExternalId.

Compiti

Preparare i file di mappatura

Esistono due modi diversi per preparare i file di mappatura:

- È possibile preparare un file di mappatura singolo e globale.

  Tutte le mappature vengono posizionate in un file singolo denominato **mappings.xml**.

- È possibile preparare un file separato per ciascuna query di invio.

  Ciascun file di mappatura è denominato <**nome query**>**.xml**.

Per i dettagli consultare "Schema del file di mappatura" a pag. 271.

Questo compito include i passaggi seguenti:

- "Creare il file di mappatura" a pag. 263
- "Mappare i CI" a pag. 264
- "Mappare i collegamenti" a pag. 264

1 Creare il file di mappatura

La struttura del file di mappatura è la seguente

```
<?xml version="1.0" encoding="UTF-8"?>
<integration>
    <info>
        <source name="UCMDB"  versions="9.x" vendor="HP" />
        <!-- for example: -->
        <target name="Oracle" versions="11g" vendor="Oracle" />
    </info>
    <targetcis>
        <!--- CI Mappings --->
    </targetcis>
    <targetrelations>
        <!--- Link Mappings --->
    </ targetrelations>
</integration>
```

2 Mappare i CI

Ciascun CIT CMDB viene mappato come nell'esempio seguente:

```
<source_ci_type name="node" mode="update_else_insert">
    <apioutputseq>1</apioutputseq>
    <target_ci_type  name="host">
        <targetprimarykey><pkey>host_key</pkey></targetprimarykey>
        <target_attribute name="host_os" datatype="STRING">
            <map type="direct" source_attribute="discovered_os_name" />
        </target_attribute>
        <!-- more target attributes --->
    </target_ci_type>
</source_ci_type>
```

Nota: i valori possibili della modalità dipendono dall'implementazione dello script.

---

3 Mappare i collegamenti

Ciascun collegamento valido viene mappato come nell'esempio seguente:

```
<link source_link_type="dependency" target_link_type="dependency"
mode="update_else_insert" source_ci_type_end1="webservice"
source_ci_type_end2="sap_gateway">
    <target_ci_type_end1 name="webservice" />
    <target_ci_type_end2 name="sap_gateway" />
</link>
```

Scrivere gli script Jython

Lo script di mappatura è un normale script Jython e deve seguire le regole degli script Jython. Per i dettagli consultare "Sviluppo degli adattatori Jython" a pag. 63.

Lo script deve contenere la funzione **DiscoveryMain** che potrebbe restituire un'istanza **OSHVResult** o **DataPushResults** a operazione riuscita.

Per segnalare eventuali errori, lo script deve sollevare un'eccezione, ad esempio:

```
raise Exception('Failed to insert to remote UCMDB using TopologyUpdateService. See log of the remote UCMDB')
```

Nella funzione **`DiscoveryMain`**, gli elementi dei dati da inviare o eliminare dall'applicazione esterna possono essere ottenuti come segue:

```
# get add/update/delete result objects (in XML format) from the Framework
addResult = Framework.getTriggerCIData('addResult')
updateResult = Framework.getTriggerCIData('updateResult')
deleteResult = Framework.getTriggerCIData('deleteResult')
```

L'oggetto client per l'applicazione esterna può essere ottenuto come segue:

```
oracleClient = Framework.createClient()
```

Questo oggetto client utilizza automaticamente l'ID delle credenziali, il nome host e il numero porta passati dall'adattatore tramite il Framework.

Se è necessario utilizzare i parametri di connessione definiti per l'adattatore (per i dettagli consultare il passaggio 2 in "Creare un pacchetto adattatore" a pag. 270), utilizzare il codice seguente:

```
propValue = str(Framework.getDestinationAttribute('<Connection Property Name'))
```

Ad esempio:

```
serverName = Framework.getDestinationAttribute('ip_address')
```

Questa sezione include inoltre:

Utilizzo dei risultati di mappatura

L'adattatore Push generico crea stringhe XML che descrivono i dati da aggiungere, aggiornare o eliminare dal sistema di destinazione. Lo script Jython deve analizzare questo XML e, successivamente, esegue l'operazione di aggiunta, aggiornamento o eliminazione sulla destinazione.

Nell'XML dell'operazione di aggiunta ricevuto dallo script Jython, l'attributo mamId per gli oggetti e i collegamenti è sempre l'identificatore di UCMDB per il collegamento o l'oggetto originale prima che il relativo tipo, attributo o altre informazioni vengano cambiate sullo schema del sistema remoto.

Nell'XML delle operazioni di aggiornamento o rimozione, l'attributo mamId di ciascun oggetto o collegamento contiene la rappresentazione di stringa dello stesso ExternalId restituito dallo script Jython della precedente sincronizzazione.

Esempio del risultato XML

```
<root>
    <data>
        <objects>
            <Object mode="update_else_insert" name="ip" operation="add"
mamId="2ebdc7a93dc7f5bcb33a444763c2a16c">
                <field name="root_lastaccesstime" key="false" datatype="DATE"
length="">1275469266</field>
                <field name="display_label" key="false" datatype="STRING"
length="">16.59.61.67</field>
                <field name="ip_probename" key="false" datatype="STRING"
length="">VMUCMDB05</field>
            </Object>
        </objects>
        <links>
            <link targetRelationshipClass="contained" targetParent="nt"
targetChild="ip" operation="add" mode="update_else_insert"
mamId="8c0a38d53c74c3cc972d6254fb50adba">
                <field name="DiscoveryID1">d5aac653aff428b4a3780111f6389d53</
field>
                <field
name="DiscoveryID2">2ebdc7a93dc7f5bcb33a444763c2a16c</field>
            </link>
        </links>
    </data>
</root>
```

Gestione della verifica connessione nello script

È possibile richiamare uno script Jython per verificare la connessione con un'applicazione esterna. In tal caso, l'attributo di destinazione testConnection sarà true. Questo attributo può essere ottenuto dal Framework come indicato di seguito:

```
testConnection = Framework.getTriggerCIData('testConnection')
```

Durante l'esecuzione in modalità di verifica connessione, uno script deve sollevare un'eccezione se non è possibile stabilire una connessione all'applicazione esterna. In caso contrario, se la connessione è riuscita, la funzione **DiscoveryMain** deve restituire un **OSHVResult** vuoto.

Supportare la sincronizzazione differenziale

Importante: se si sta implementando la sincronizzazione differenziale su un adattatore esistente creato nella versione 9.00 o 9.01, è necessario utilizzare il file push-adapter.zip file della versione 9.02 o successive per ricreare il pacchetto adattatore. Per i dettagli consultare "Creare un pacchetto adattatore" a pag. 270.

Questo compito consente all'adattatore Push di eseguire la sincronizzazione differenziale. Per i dettagli consultare "Sincronizzazione differenziale" a pag. 262.

Lo script Jython restituisce l'oggetto **DataPushResults** contenente due mappe Java, una per le mappature ID dell'oggetto (le chiavi e i valori sono oggetti di tipo ExternalCiId) e uno per gli ID del collegamento (le chiavi e i valori sono oggetti di tipo ExternalRelationId).

➤ Aggiungere le seguenti istruzioni **from** allo script Jython:

```
from com.hp.ucmdb.federationspi.data.query.types import ExternalIdFactory
from com.hp.ucmdb.adapters.push import DataPushResults
from com.hp.ucmdb.adapters.push import DataPushResultsFactory
from com.mercury.topaz.cmdb.server.fcmdb.spi.data.query.types import
ExternalIdUtil
```

➤ Utilizzare la classe di valori predefiniti **DataPushResultsFactory** per ottenere l'oggetto **DataPushResults** dalla funzione **DiscoveryMain**.

```
# Create the UpdateResult object
updateResult = DataPushResultsFactory.createDataPushResults(objectMappings,
linkMappings);
```

➤ Utilizzare i seguenti comandi per creare le mappe Java per l'oggetto **DataPushResults**:

```
# Prepare the maps to store the mappings if IDs
objectMappings = HashMap()
linkMappings = HashMap()
```

➤ Utilizzare la classe **ExternalIdFactory** per creare i seguenti ID ExternalId:

  ➤ ExternalId per gli oggetti o i collegamenti che hanno origine in CMDB (ad esempio, tutti i CI in un'operazione di aggiunta derivano dal CMDB):

```
externaCIId = ExternalIdFactory.createExternalCmdbCiId(ciType, ciIDAsString)
externalRelationId = ExternalIdFactory.createExternalCmdbRelationId(linkType,
end1ExternalCIId, end2ExternalCIId, linkIDAsString)
```

  ➤ ExternalId per gli oggetti o i collegamenti che non hanno origine in CMDB (solitamente, ogni operazione di aggiornamento e rimozione contiene questi oggetti):

```
myIDField = TypesFactory.createProperty("systemID", "1")
myExternalId = ExternalIdFactory.createExternalCiId(type, myIDField)
```

Nota: se lo script Jython ha aggiornato informazioni esistenti e l'ID dell'oggetto (o collegamento) cambia, è necessario restituire una mappatura tra l'ID esterno precedente e quello nuovo.

- Utilizzare i metodi **restoreCmdbCiIDString** o **restoreCmdbRelationIDString** dalla classe **ExternalIdFactory** per recuperare la stringa ID di UCMDB da un ID esterno di un oggetto o un collegamento con origine in UCMDB.
- Utilizzare i metodi **restoreExternalCiId** e **restoreExternalRelationId** dalla classe **ExternalIdUtil** per ripristinare l'oggetto **ExternalId** dal valore dell'attributo mamId dell'XML delle operazioni di aggiornamento o rimozione.

Nota: gli oggetti ExternalId sono di fatto un array di proprietà. Ciò significa che è possibile utilizzare un oggetto ExternalId per archiviare qualsiasi informazione potenzialmente necessaria che identificherà i dati sul sistema remoto.

Creare un pacchetto adattatore

1. Estrarre il contenuto di **C:\hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip** in una cartella temporanea.
2. Modificare il file **discoveryPatterns\push_adapter.xml**.
   a. Modificare il tag <pattern> con un nuovo ID e una etichetta visualizzata. Sostituire:

```
<pattern id="PushAdapter" xsi:noNamespaceSchemaLocation="../../
Patterns.xsd" description="Discovery Pattern Description" schemaVersion="9.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

   con

```
<pattern id="MyPushAdapter" displayLabel="My Push Adapter"
xsi:noNamespaceSchemaLocation="../../Patterns.xsd" description="Discovery
Pattern Description" schemaVersion="9.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
```

   b. Aggiornare l'elenco dei parametri in modo che tale elenco rifletta gli attributi di connessione richiesti. Non rimuovere l'attributo **probeName**.
3. Rinominare la cartella **adapterCode\PushAdapter** con l'ID adattatore utilizzato nel passaggio 2 (ad esempio, **adapterCode\MyPushAdapter**).
4. Sostituire **discoveryScripts\pushScript.py** con lo script scritto (per i dettagli consultare"Scrivere gli script Jython" a pag. 264). Se si rinomina lo script, la proprietà jythonScript.name in **adapterCode\<adapter ID>\push.properties** deve essere aggiornata di conseguenza.
5. Sostituire il file **adapterCode\<adapter ID>\mappings\mappings.xml** file con i file di mappatura preparati (per i dettagli consultare "Preparare i file di mappatura" a pag. 263.

   Se si desidera utilizzare un file di mappatura per ciascun metodo TQL, assegnare il nome del TQL corrispondente a ciascun file XML, seguito da **.xml**. In tal caso, il file **mappings.xml** verrà utilizzato come predefinito se non viene trovato alcun file di mappatura specifico per il nome TQL corrente. Il nome del file di mappatura predefinito può essere modificato cambiando la proprietà mappingFile.default in **adapterCode\<adapter ID>\push.properties**.

Riferimenti

Schema del file di mappatura

Elemento		Attributi
Nome e percorso	**Descrizione**	
integration	Definisce il contenuto di mappatura del file. Deve essere il blocco più esterno nel file ad eccezione della riga di inizio e di eventuali commenti.	
info (integration)	Definisce le informazioni sui repository di dati integrati	
source (integration > info)	Definisce le informazioni sul repository di dati di origine	**Nome**. type **Descrizione.** Nome del repository di dati di origine. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. versions **Descrizione.** Versione(i) dei repository di dati di origine. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. vendor **Descrizione.** Fornitore del repository di dati di origine. **Obbligatorietà.** Obbligatorio **Tipo.** String

Elemento		Attributi
Nome e percorso	**Descrizione**	
target (integration > info)	Definisce le informazioni sul repository di dati di destinazione	**Nome.** type **Descrizione.** Nome del repository di dati di origine. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. versions **Descrizione.** Versione(i) del repository di dati di origine. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. vendor **Descrizione.** Fornitore del repository di dati di origine. **Obbligatorietà.** Obbligatorio **Tipo.** String
targetcis (integration)	Elemento contenitore per tutte le mappature CIT	

Elemento		Attributi
Nome e percorso	**Descrizione**	
source_ci_type (integration > targetcis)	Definisce un CIT di origine	**Nome.** name **Descrizione.** Nome del CIT di origine. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. mode **Descrizione.** Il tipo di aggiornamento richiesto per il tipo CI corrente. **Obbligatorietà.** Obbligatorio **Tipo.** Una delle stringhe seguenti: ➤ **insert** - utilizza questo solo se il CI non è già esistente. ➤ **update** - utilizza questo solo se si è certi dell'esistenza del CI. ➤ **update_else_insert** - se il CI esiste, aggiornarlo, in caso contrario creare un nuovo CI. ➤ **ignore** - non eseguire alcuna operazione con questo tipo CI.

Elemento		Attributi
Nome e percorso	**Descrizione**	
target_ci_type (integration > targetcis > source_ci_type)	Definisce un CIT di destinazione	**Nome.** name **Descrizione.** Nome del tipo CI di destinazione. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. schema **Descrizione.** Il nome dello schema che verrà utilizzato per archiviare questo tipo CI sulla destinazione. **Obbligatorietà.** Non obbligatorio **Tipo.** String
		Nome. namespace **Descrizione.** Indica lo spazio dei nomi di questo tipo CI sulla destinazione **Obbligatorietà.** Non obbligatorio **Tipo.** String
targetprimarykey (integration > targetcis > source_ci_type -OPPURE- integration > targetrelations > link)	Identifica gli attributi chiave principali del CIT di destinazione	
pkey (integration > targetcis > source_ci_type > targetprimarykey -OPPURE- integration > targetrelations > link > targetprimarykey)	Identifica un attributo chiave principale Richiesto solo se la modalità è **update** o **insert_else_update**	

Elemento		Attributi
Nome e percorso	**Descrizione**	
target_attribute (integration > targetcis > source_ci_type -OPPURE- integration > targetrelations > link)	Definisce l'attributo del CIT di destinazione	**Nome.** name **Descrizione.** Nome dell'attributo del CIT di destinazione. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. datatype **Descrizione.** Tipo dati dell'attributo del CIT di destinazione. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. length **Descrizione.** Per i tipi di dati stringa/ caratteri, dimensione intera dell'attributo di destinazione. **Obbligatorietà.** Non obbligatorio **Tipo.** Integer
		Nome. option **Descrizione.** La funzione di conversione da applicare al valore. **Obbligatorietà.** False **Tipo.** Una delle stringhe seguenti: ➤ **uppercase** - converte in maiuscolo ➤ **lowercase** - converte in minuscolo ➤ Se questo attributo è vuoto, non verrà applicata alcuna funzione di conversione.

Elemento		Attributi
Nome e percorso	**Descrizione**	
map (integration > targetcis > source_ci_type > target_attribute -OPPURE- integration > targetrelations > link > target_attribute)	Specifica come ottenere il valore dell'attributo del CIT di origine	**Nome.** type **Descrizione.** Il tipo di mappatura tra i valori di origine e quelli di destinazione. **Obbligatorietà.** Obbligatorio **Tipo.** Una delle stringhe seguenti: ➤ **direct** - specifica una mappatura 1-to-1 dal valore dell'attributo di origine al valore dell'attributo di destinazione ➤ **compoundstring** - i sottoelementi vengono uniti in una stringa singola e viene impostato il valore dell'attributo di destinazione ➤ **childattr** - i sottoelementi sono uno o più attributi del CIT figlio. I CIT figli sono definiti come quelli con relazione **container_f** o **contained**. ➤ **constant** - stringa statica
		Nome. value **Descrizione.** Stringa costante per tipo=**constant** **Obbligatorietà.** Richiesto solo quando tipo=**constant** **Tipo.** String
		Nome. attr **Descrizione.** Nome attributo di origine per tipo=**direct** **Obbligatorietà.** Richiesto solo quando tipo=**direct** **Tipo.** String

Elemento		Attributi
Nome e percorso	**Descrizione**	
aggregation (integration > targetcis > source_ci_type > target_attribute > map -OPPURE- integration > targetrelations > link > target_attribute > map Valido solo quando il tipo di mappa è **childattr**)	Specifica come i valori degli attributi del CI figlio del CI di origine vengono combinati in un singolo valore per la mappatura sull'attributo del CI di destinazione. Facoltativo.	**Nome.** type **Descrizione.** Il tipo di funzione di aggregazione **Obbligatorietà.** Obbligatorio **Tipo.** Una delle stringhe seguenti: ➤ **csv** - concatena tutti i valori inclusi un elenco separato da virgole (numerico o stringa/carattere). ➤ **count** - restituisce un conteggio numerico di tutti i valori inclusi. ➤ **sum** - restituisce un conteggio numerico di tutti i valori inclusi. ➤ **average** - restituisce una media numerica di tutti i valori inclusi. ➤ **min** - restituisce il più basso valore numerico/carattere incluso. ➤ **max** - restituisce il più alto valore numerico/carattere incluso.

Elemento		Attributi
Nome e percorso	**Descrizione**	
validation (integration > targetcis > source_ci_type > target_attribute > map -OPPURE- integration > targetrelations > link > target_attribute > map Valido solo quando il tipo di mappa è **childatt**)	Consente il filtro di esclusione dei CI figli del CI di origine basati sui valori degli attributi. Utilizzato con il sottoelemento di aggregazione per raggiungere la granularità esatta di quali attributi figli vengono mappati sul valore attributo del CIT di destinazione. Facoltativo.	**Nome.** minlength **Descrizione.** Esclude le stringhe più brevi rispetto al valore fornito. **Obbligatorietà.** Non obbligatorio **Tipo.** Integer
		Nome. maxlength **Descrizione.** Esclude le stringhe più lunghe rispetto al valore fornito. **Obbligatorietà.** Non obbligatorio **Tipo.** Integer
		Nome. minvalue **Descrizione.** Esclude i numeri più piccoli rispetto al valore specificato. **Obbligatorietà.** Non obbligatorio **Tipo.** Number
		Nome. maxvalue **Descrizione.** Esclude i numeri più grandi rispetto al valore specificato. **Obbligatorietà.** Non obbligatorio **Tipo.** Number
targetrelations (integration)	Elemento contenitore per tutte le mappature delle relazioni. Facoltativo.	

Elemento		Attributi
Nome e percorso	**Descrizione**	
link (integration > targetrelations)	Esegue la mappatura di una relazione di origine su una relazione di destinazione. Obbligatorio solo se **targetrelation** è presente.	**Nome.** source_link_type **Descrizione.** Nome relazione di origine. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. target_link_type **Descrizione.** Nome relazione di destinazione. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. nameSpace **Descrizione.** Lo spazio dei nomi per il collegamento che verrà creato sulla destinazione. **Obbligatorietà.** Non obbligatorio **Tipo.** String
		Nome. mode **Descrizione.** Il tipo di aggiornamento richiesto per il collegamento corrente. **Obbligatorietà.** Obbligatorio **Tipo.** Una delle stringhe seguenti: ➤ **insert** - utilizza questo solo se il CI non è già esistente. ➤ **update** - utilizza questo solo se si è certi dell'esistenza del CI. ➤ **update_else_insert** - se il CI esiste, aggiornarlo, in caso contrario creare un nuovo CI. ➤ **ignore** - non eseguire alcuna operazione con questo tipo CI.

Elemento		Attributi
Nome e percorso	**Descrizione**	
link (continua)		**Nome.** source_ci_type_end1 **Descrizione.** Tipo CI End1 della relazione di origine **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. source_ci_type_end2 **Descrizione.** Tipo CI End2 della relazione di origine **Obbligatorietà.** Obbligatorio **Tipo.** String
target_ci_type_end1 (integration > targetrelations > link)	Tipo CI End1 della relazione di destinazione	**Nome.** name **Descrizione.** Nome del tipo CI End1 della relazione di destinazione. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. superclass **Descrizione.** Nome della superclasse del tipo CI End1. **Obbligatorietà.** Non obbligatorio **Tipo.** String

Elemento		Attributi
Nome e percorso	**Descrizione**	
target_ci_type_end2 (integration > targetrelations > link)	Tipo CI End2 della relazione di destinazione	**Nome.** name **Descrizione.** Nome del tipo CI End2 della relazione di destinazione. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. superclass **Descrizione.** Nome della superclasse del tipo CI End2. **Obbligatorietà.** Non obbligatorio **Tipo.** String

Mappatura dello schema dei risultati

Elemento		Attributi
Nome e percorso	**Descrizione**	
root	La radice del documento dei risultati	
data (root)	La radice dei dati stessi	
objects (root > data)	L'elemento radice per gli oggetti da aggiornare	

Elemento		Attributi
Nome e percorso	**Descrizione**	
Object (root > data > objects)	Descrive l'operazione di aggiornamento per un singolo oggetto e per tutti i relativi attributi	**Nome.** name **Descrizione.** Nome del Tipo CI **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. mode **Descrizione.** Il tipo di aggiornamento richiesto per il tipo CI corrente. **Obbligatorietà.** Obbligatorio **Tipo.** Una delle stringhe seguenti: ➤ **insert** - utilizza questo solo se il CI non è già esistente. ➤ **update** - utilizza questo solo se si è certi dell'esistenza del CI. ➤ **update_else_insert** - se il CI esiste, aggiornarlo, in caso contrario creare un nuovo CI. ➤ **ignore** - non eseguire alcuna operazione con questo tipo CI.

Elemento		Attributi
Nome e percorso	**Descrizione**	
Object (continua)		**Nome.** operation **Descrizione.** L'operazione da eseguire con questo CI. **Obbligatorietà.** Obbligatorio **Tipo.** Una delle stringhe seguenti: ➤ **add** - il CI deve essere aggiunto ➤ **update** - il CI deve essere aggiornato ➤ **delete** - il CI deve essere eliminato Se non è impostato alcun valore, viene utilizzato il valore predefinito di **add**.
		Nome. mamId **Descrizione.** L'ID dell'oggetto sul CMDB di origine. **Obbligatorietà.** Obbligatorio **Tipo.** String

Elemento		Attributi
Nome e percorso	**Descrizione**	
field (root > data > objects > Object -OPPURE- root > data > links > link)	Descrive il valore di un campo singolo per un oggetto. Il testo del campo è il nuovo valore nel campo e, se il campo contiene un collegamento, il valore è l'ID di una delle estremità. Ciascun ID di fine viene visualizzato come oggetto (in <objects>).	**Nome.** name **Descrizione.** Nome del campo. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. key **Descrizione.** Specifica se questo campo è una chiave per l'oggetto. **Obbligatorietà.** Obbligatorio **Tipo.** Boolean
		Nome. datatype **Descrizione.** Il tipo di campo. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. length **Descrizione.** Per i tipi di dati stringa/caratteri, questa è la dimensione intera dell'attributo di destinazione. **Obbligatorietà.** Non obbligatorio **Tipo.** Integer

Elemento		Attributi
Nome e percorso	**Descrizione**	
links (root > data)	L'elemento radice per i collegamenti da aggiornare	**Nome.** targetRelationshipClass **Descrizione.** Il nome della relazione (collegamento) nel sistema di destinazione. **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. targetParent **Descrizione.** Il tipo della prima estremità del collegamento (padre). **Obbligatorietà.** Obbligatorio **Tipo.** String
		Nome. targetChild **Descrizione.** Il tipo della seconda estremità del collegamento (figlio). **Obbligatorietà.** Obbligatorio **Tipo.** String

Elemento		Attributi
Nome e percorso	**Descrizione**	
links (continua)		**Nome.** mode **Descrizione.** Il tipo di aggiornamento richiesto per il tipo CI corrente. **Obbligatorietà.** Obbligatorio **Tipo.** Una delle stringhe seguenti: ➤ **insert** - utilizza questo solo se il CI non è già esistente. ➤ **update** - utilizza questo solo se si è certi dell'esistenza del CI. ➤ **update_else_insert** - se il CI esiste, aggiornarlo, in caso contrario creare un nuovo CI. ➤ **ignore** - non eseguire alcuna operazione con questo tipo CI.
		Nome. operation **Descrizione.** L'operazione da eseguire con questo CI. **Obbligatorietà.** Obbligatorio **Tipo.** Una delle stringhe seguenti: ➤ **add** - il CI deve essere aggiunto ➤ **update** - il CI deve essere aggiornato ➤ **delete** - il CI deve essere eliminato Se non è impostato alcun valore, viene utilizzato il valore predefinito di **add**.
		Nome. mamId **Descrizione.** L'ID dell'oggetto sul CMDB di origine. **Obbligatorietà.** Obbligatorio **Tipo.** String

Parte II

Utilizzo delle API

8

Introduzione alle API

Questo capitolo comprende:

Concetti

Concetti

API - Panoramica

Le API seguenti sono incluse con HP Universal CMDB:

- **API servizio Web UCMDB.** Consente di scrivere le definizioni dell'elemento di configurazione e le relazioni topologiche con UCMDB (Universal Configuration Management database) e l''interrogazione sulle informazioni mediante query TQL e ad-hoc. Per i dettagli consultare "API servizio Web di HP Universal CMDB" a pag. 291.
- **API Java UCMDB.** Spiega come strumenti di terze parti e personalizzati possono utilizzare l'API Java per estrarre i dati e i calcoli e scrivere i dati in UCMDB (Universal Configuration Management database). Per i dettagli consultare "HP Universal CMDB API" a pag. 373.

9

API servizio Web di HP Universal CMDB

Questo capitolo comprende:

Concetti

Compiti

Riferimenti

Concetti

Convenzioni

In questo capitolo vengono utilizzate le seguenti convenzioni:

- **UCMDB** si riferisce a Universal Configuration Management Database. **HP Universal CMDB** si riferisce all'applicazione.
- Gli elementi e gli argomenti del metodo di UCMDB sono immessi con l'iniziale maiuscola o minuscola specificata nello schema. Un elemento o argomento di un metodo non è in maiuscolo. Ad esempio, una relation è un elemento di tipo Relation passato a un metodo.

API servizio Web di HP Universal CMDB - Panoramica

Utilizzare il presente capitolo insieme alla documentazione dello schema UCMDB, disponibile nella Libreria della documentazione online.

L'API servizio Web di HP Universal CMDB è utilizzata per integrare le applicazioni con HP Universal CMDB (UCMDB). L'API fornisce dei metodi per:

- aggiungere, rimuovere e aggiornare CI e relazioni in CMDB
- recuperare le informazioni sul modello di classe
- recuperare le analisi impatto
- recuperare informazioni sulle relazioni e sugli elementi di configurazione
- gestire credenziali: visualizzare, aggiungere, aggiornare e rimuovere
- gestire processi: visualizzare lo stato, attivare e disattivare
- gestire gli intervalli della sonda: visualizzare, aggiungere e aggiornare
- gestire trigger: aggiungere o rimuovere un CI trigger e aggiungere, rimuovere o disabilitare una TQL trigger
- visualizzare dati generali su domini e sonde

I metodi di recupero delle informazioni sulle relazioni e sugli elementi di configurazione in genere utilizzano il Topology Query Language (TQL). Per i dettagli consultare "Topology Query Language" nella *Guida alla modellazione di HP Universal CMDB*.

Gli utenti dell'API servizio Web di HP Universal CMDB devono conoscere:

- Le specifiche SOAP
- Un linguaggio di programmazione orientato all'oggetto come C++, C# o Java
- HP Universal CMDB
- Gestione flusso di dati

In questa sezione vengono trattati i seguenti argomenti:

- "Utilizzi dell'API" a pag. 293
- "Autorizzazioni" a pag. 294

Utilizzi dell'API

L'API viene utilizzata per soddisfare un certo numero di requisiti aziendali. Ad esempio:

- Un sistema di terze parti può interrogare il modello di classe per ottenere informazioni sugli elementi di configurazione (CI) disponibili.
- Uno strumento di gestione asset di terze parti può aggiornare il CMDB con le informazioni disponibili solo su tale strumento, unificando pertanto i relativi dati con i dati raccolti dalle applicazioni HP.
- Diversi sistemi di terze parti possono popolare il CMDB per creare un CMDB centrale che rilevi i cambiamenti ed esegua l'analisi impatto.
- Un sistema di terze parti può creare entità e relazioni in base alla relativa logica e scrivere quindi i dati nel CMDB per usufruire delle capacità di interrogazione del CMDB.
- Altri sistemi, come ad esempio il sistema Release Control (CCM), possono utilizzare i metodi dell'analisi impatto per cambiare l'analisi.

Autorizzazioni

L'amministratore fornisce le credenziali di accesso per la connessione al servizio Web. Le credenziali richieste dipendono dal fatto che si utilizzi o meno HP Universal CMDB come applicazione standalone o dall'interno di Business Service Management:

- **HP Universal CMDB standalone.** Accedere utilizzando le credenziali di un utente di UCMDB con autorizzazioni per le risorse di individuazione e integrazione.

  Per i dettagli consultare "Pagina Gestione protezione" nella*Guida all'amministrazione di HP Universal CMDB*.

- **HP Universal CMDB incorporato in Business Service Management.** Accedere utilizzando le credenziali di un utente di Business Service Management. L'utente deve disporre della autorizzazioni pertinenti per la risorsa di HP Universal CMDB in Business Service Management.

Quando le autorizzazioni vengono assegnate tramite HP Universal CMDB, i livelli di autorizzazione sono View, Update ed Execute. Quando vengono assegnate utilizzando Business Service Management, i livelli sono View e Update, dove Update include anche Execution. Per visualizzare le autorizzazioni richieste per ciascuna operazione, consultare la documentazione di richiesta di ciascuna operazione e consultare *Riferimento per lo schema di Gestione flusso di dati*.

Riferimento API servizio Web HP Universal CMDB

Per la documentazione completa sulle strutture di richiesta e risposta, fare riferimento a Riferimento API servizio Web di HP UCMDB. Questi file si trovano nella cartella seguente:

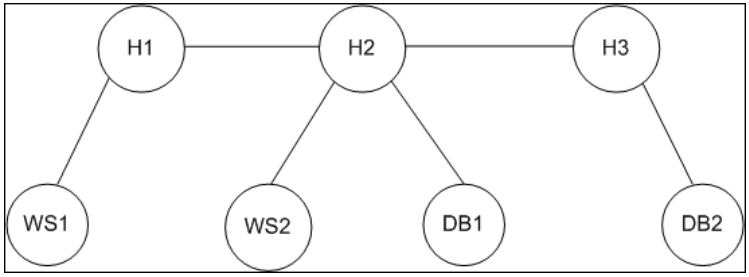
C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\doc_lib\DevRef_guide\CMDB_Schema\webframe.html

Restituzione di elementi della mappa topologica non ambigui

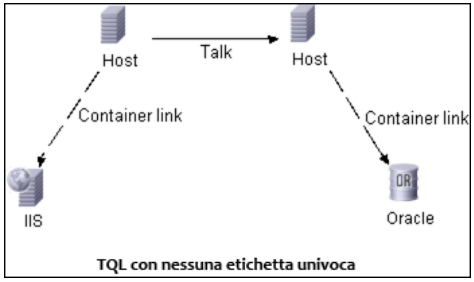
I metodi di query che restituiscono i dati negli elementi topology o topologyMap cercano nel sistema una corrispondenza di una query TQL. I diagrammi seguenti illustrano come le strutture topology e topologyMap risultanti sono influenzate dall'utilizzo di etichette univoche nella query.

Le etichette sono nomi specificati dall'utente nella query per le relazioni e gli elementi di configurazione in configurazioni specifiche. Le etichette specificate nella query sono utilizzate come etichette di nodo nella mappa restituita. Se non viene specificata alcuna etichetta, il Type Name CI o Relation viene utilizzato come etichetta nella mappa risultante. L'esempio seguente illustra come specificare le etichette IISHost e DBHost al posto dell'etichetta Host predefinita e le etichette ContainerIIS e ContainsDB al posto dell'etichetta Container Link predefinita.

L'esempio seguente rappresenta un modello universo IT di piccole dimensioni. Gli host disponibili sono tre: H1, H2, H3 che ospitano i server Web (WS) e le gestioni dei database (DB). WS1 risiede su H1. DB1 e WS2 risiedono entrambi su H2. DB2 risiede su H3.



Questa query è definita mediante le etichette predefinite:



Il risultato dell'esecuzione di questa query TQL sull'universo IT può essere l'elemento Topology o TopologyMap.

Risposta Topology

```
CIs: H1, H2, H3, WS1, WS2, DB1, DB2
Relations: H1-WS1, H1-H2, H2-H3, WS2-H2, DB1-H2, DB2-H3
```

Risposta TopologyMap

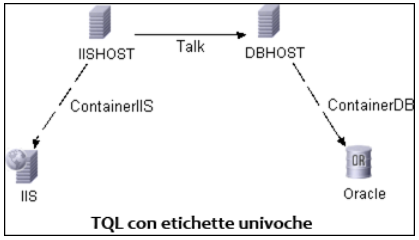
```
CINode:
    label: Host
    CIs: H1, H2
CINode:
    label: Host
    CIs: H2, H3
CINode:
    label: DB
    CIs: DB1, DB2
CINode:
    label: Webserver
CIs: IIS
relationNode:
    label: talk
    relations: H1-H2, H2-H3

relationNode:
    label: Container Link
    relations: WS1-H1, WS2-H2
relationNode:
    label: Container Link
    relations: DB2-H3, DB1-H2
```

Nella risposta TopologyMap suddetta, i primi due CINodecontengono etichette Host identiche, corrispondenti ai due CI Host nella query. Entrambi i CINode contengono l'host H2, senza alcuna indicazione sul motivo della duplicazione di H2.

Gli ultimi due relationNode contengono etichette Contained identiche, corrispondenti alle due relazioni Container link nella query.

Si verificano duplicazioni poiché non viene specificata alcuna etichetta univoca nella query; di conseguenza, nella mappa, vengono utilizzate le etichette predefinite (i nomi tipo Host e Container) Per estrarre una mappa più utilizzabile, definire le query con etichette univoche per ciascuna configurazione di cui trovare la corrispondenza, come mostrato nella query seguente:



Il risultato topology è identico a quello della TQL senza etichette univoche. Il risultato topologyMap è, tuttavia, diverso: ciascuna etichetta è ora univoca.

```
CINode:
    label: IISHOST
    CIs: H1, H2
CINode:
    label: DBHOST
    CIs: H2, H3

...

relationNode:
    label: ContainerIIS
    relations: WS1-H1, WS2-H2
relationNode:
    label: ContainerDB
    relations: DB2-H3, DB1-H2
```

In questa mappa, è chiaro il motivo per cui H2 viene restituito 2 volte. Le etichette univoche indicano che è stato restituito una volta come host del server Web e una volta come host del database.

Suggerimento: ogni volta che è possibile nel CMDB, applicare etichette univoche e definite dall'utente a configurazioni specifiche.

Compiti

Chiamare il servizio Web

È possibile utilizzare le tecniche di programmazione SOAP standard nel servizio Web di HP Universal CMDB per consentire la chiamate dei metodi lato server. Se non è possibile analizzare l'istruzione o è presente un problema di chiamata del metodo, i metodi API generano un'eccezione SoapFault. Quando viene generata un'eccezione SoapFault, UCMDB popola uno o più campi del messaggio di errore, codice di errore e messaggio di eccezione. Se non è presente alcun errore, vengono restituiti i risultati della chiamata.

I programmatori SOAP possono accedere al WSDL dall'indirizzo:

http://<server>[:port]/axis2/services/UcmdbService?wsdl

La specifica della porta è necessaria solo per le installazioni non standard. Consultare l'amministratore di sistema per il numero di porta corretto.

L'URL per la chiamata del servizio è:

http://<server>[:port]/axis2/services/UcmdbService

Per gli esempi di connessione al CMDB consultare "Casi di utilizzo" a pag. 331.

Interrogare il CMDB

Il CMDB viene interrogato utilizzando le API descritte in "Metodi di query UCMDB" a pag. 308.

Le query e gli elementi di CMDB restituiti contengono sempre gli ID reali di UMDB .

Per gli esempi di utilizzo dei metodi di query consultare "Esempio di query" a pag. 337.

In questa sezione vengono trattati i seguenti argomenti:

Calcolo tempestivo della risposta

Per tutti i metodi di query, il server UMDB calcola i valori richiesti dal metodo di query al momento della ricezione della richiesta e restituisce i risultati in base agli ultimi dati. Il risultato viene sempre calcolato al momento della ricezione della richiesta anche se la query TQL è attiva ed è presente un risultato calcolato in precedenza. Pertanto, i risultati di esecuzione di una query restituiti all'applicazione client potrebbero differire dai risultati della stessa query visualizzati sull'interfaccia utente.

Suggerimento: se l'applicazione utilizza i risultati di una query specifica più di una volta ed è stato previsto che i dati non cambieranno significativamente tra i diversi utilizzi dei dati dei risultati, è possibile migliorare le prestazioni facendo sì che l'applicazione client memorizzi i dati invece di eseguire ripetutamente la query.

Elaborazione delle risposte di grandi dimensioni

La risposta a una query include sempre le strutture dei dati richiesti dal metodo di query anche se non viene trasmesso alcun dato effettivo. Per molti metodi in cui i dati sono una raccolta o una mappa, la risposta include anche la struttura ChunkInfo, composta da chunksKey e numberOfChunks. Il campo numberOfChunks indica il numero di blocchi contenenti i dati da recuperare.

La dimensione massima di trasmissione dei dati è impostata dall'amministratore di sistema. Se i dati restituiti dalla query hanno una dimensione superiore a quella massima, le strutture dei dati nella prima risposta non contengono informazioni significative e il valore del campo numberOfChunks è 2 o superiore. Se i dati non superano la dimensione massima, il campo numberOfChunks è 0 (zero) e i dati vengono trasmessi nella prima risposta. Pertanto, durante l'elaborazione di una risposta, controllare prima il valore numberOfChunks. Se maggiore di 1, eliminare i dati nella trasmissione e richiedere i blocchi di dati. Altrimenti, utilizzare i dati nella risposta.

Per informazioni sulla gestione dei dati in blocchi consultare "pullTopologyMapChunks" a pag. 320 e "releaseChunks" a pag. 321.

Specifica delle proprietà da restituire

I CI e le relazioni hanno generalmente molte proprietà. Alcuni metodi che restituiscono raccolte o grafici di questi elementi accettano i parametri di input che specificano quali valori delle proprietà restituire con ciascun elemento corrispondente alla query. Il CMDB non restituisce proprietà vuote. Pertanto, la risposta a una query potrebbe avere meno proprietà di quelle richieste nella query.

Questa sezione descrive i tipi di set utilizzati per specificare le proprietà da restituire.

È possibile fare riferimento alle proprietà in due modi:

- Per nome
- Utilizzando i nomi delle regole delle proprietà predefinite. Le regole delle proprietà predefinite sono utilizzate da CMDB per creare un elenco dei nomi reali delle proprietà.

Quando un'applicazione fa riferimento alle proprietà per nome, passa un elemento PropertiesList.

Suggerimento: ogni volta che è possibile, utilizzare PropertiesList per specificare i nomi delle proprietà interessate invece di utilizzare un set basato su regole. L'utilizzo delle regole delle proprietà predefinite genera quasi sempre la restituzione di più proprietà del necessario, limitando pertanto le prestazioni.

Sono possibili due tipi di proprietà predefinite: proprietà del qualificatore e proprietà semplici.

- **Proprietà del qualificatore.** Utilizzare quando l'applicazione client deve passare un elemento QualifierProperties (un elenco di qualificatori che è possibile applicare alle proprietà). Il CMDB converte l'elenco dei qualificatori passato dall'applicazione client nell'elenco delle proprietà al quale è applicato almeno uno dei qualificatori. I valori di queste proprietà vengono restituiti con gli elementi CI o Relation.
- **Proprietà semplici.** Per utilizzare proprietà semplici basate su regole, l'applicazione client passa un elemento SimplePredefinedProperty o SimpleTypedPredefinedProperty. Questi elementi contengono il nome della regola attraverso la quale il CMDB genera l'elenco delle proprietà da restituire. Le regole che possono essere specificate in un elemento SimplePredefinedProperty o SimpleTypedPredefinedProperty sono CONCRETE, DERIVED e NAMING.

Proprietà concrete

Le proprietà concrete sono il set di proprietà definite per il CIT specificato. Le proprietà aggiunte dalle classi derivate non vengono restituite per le istanze di quelle classi derivate.

Un raccolta di istanze restituite da un metodo potrebbe essere composta dalle istanze di un CIT specificato nella chiamata del metodo e dalle istanze dei CIT ereditati da quel CIT. I CIT derivati ereditano le proprietà del CIT specificato. Inoltre, i CIT derivati estendono il CIT padre aggiungendo proprietà.

Esempio di proprietà concrete:

Il CIT T1 ha le proprietà P1 e P2. Il CIT T11 eredita da T1 ed estende T1 con le proprietà P21 e P22.

La raccolta di CI di tipo T1 include le istanze di T1 e T11. Le proprietà concrete di tutte le istanze di questa raccolta sono P1 e P2.

Proprietà derivate

Le proprietà derivate sono il set di proprietà definite dal CIT specificato e, per ciascun CIT derivato, le proprietà aggiunte dal CIT derivato.

Esempio di proprietà derivate:

Proseguendo con l'esempio delle proprietà concrete, le proprietà delle istanze di T1 sono P1 e P2. Le proprietà derivate delle istanze di T11 sono P1, P2, P21 e P22.

Proprietà Naming

Le proprietà di denominazione sono display_label e data_name.

Altri elementi di specifica delle proprietà

➤ **PredefinedProperties**

PredefinedProperties può contenere un elemento QualifierProperties e un elemento SimplePredefinedProperty per ciascuna delle altre regole possibili. Un set PredefinedProperties non contiene necessariamente tutti i tipi di elenco.

➤ **PredefinedTypedProperties**

PredefinedTypedProperties è utilizzato per applicare un set diverso di proprietà a ciascun CIT. PredefinedTypedProperties può contenere un elemento QualifierProperties e un elemento SimpleTypedPredefinedProperty per ciascuna delle altre regole applicabili. Poiché PredefinedTypedProperties viene applicato a ciascun CIT singolarmente, le proprietà derivate non sono rilevanti. Un set PredefinedProperties non contiene necessariamente tutti i tipi applicabili di elenco.

➤ **CustomProperties**

CustomProperties può contenere una combinazione dell'elenco di base PropertiesList e degli elenchi di proprietà basati su regole. Il filtro proprietà è l'unione di tutte le proprietà restituite da tutti gli elenchi.

➤ **CustomTypedProperties**

CustomTypedProperties può contenere una combinazione dell'elenco di base PropertiesList e degli elenchi di proprietà basati su regole applicabili. Il filtro proprietà è l'unione di tutte le proprietà restituite da tutti gli elenchi.

➤ **TypedProperties**

TypedProperties è utilizzato per passare un set diverso di proprietà per ciascun CIT. TypedProperties è una raccolta di coppie composta dai nomi tipo e dai set di proprietà di tutti i tipi. Ciascun set di proprietà viene applicato solo al tipo corrispondente.

Aggiornare UCMDB

È possibile aggiornare il CMDB con le API di aggiornamento. Per i dettagli dei metodi API consultare "Metodi di aggiornamento di UCMDB" a pag. 322.

Per gli esempi di utilizzo dei metodi di aggiornamento consultare "Esempio di aggiornamento" a pag. 353.

Questo compito include i passaggi seguenti:

➤ "Parametri di aggiornamento di UCMDB" a pag. 303

➤ "Utilizzo dei tipi ID con i metodi di aggiornamento" a pag. 304

➤ "Metodi di aggiornamento di UCMDB" a pag. 322

Parametri di aggiornamento di UCMDB

Questo argomento descrive i parametri utilizzati solo dai metodi di aggiornamento del servizio. Per i dettagli consultare la documentazione dello schema.

CIsAndRelationsUpdates

Il tipo CIsAndRelationsUpdates è composto da CIsForUpdate, relationsForUpdate, referencedRelations e referencedCIs. Un'istanza CIsAndRelationsUpdates non include necessariamente tutti e tre gli elementi.

CIsForUpdate è una raccolta di CI. relationsForUpdate è una raccolta Relations. Gli elementi CI e relation nelle raccolte hanno un elemento props. Quando si crea un CI o una relazione, le proprietà che hanno l'attributo requiredo l'attributo key nella definizione del tipo CI devono essere popolate con i valori. Le voci di queste raccolte vengono aggiornate o create dal metodo.

referencedCIs e referencedRelations sono raccolte di CI già definite nel CMDB. Gli elementi della raccolta sono definiti con un ID temporaneo in combinazione con tutte le proprietà chiave. Questi elementi vengono utilizzati per risolvere le identità dei CI e delle relazioni per l'aggiornamento. Non vengono mai creati o aggiornati dal metodo.

Ciascuno degli elementi CI e relation di queste raccolte ha una raccolta di proprietà. I nuovi elementi vengono creati con i valori delle proprietà di queste raccolte.

Utilizzo dei tipi ID con i metodi di aggiornamento

Quanto segue descrive i CIT ID, i CI e le relazioni. Quando l'ID non è un ID CMDB reale, sono necessari gli attributi tipo e chiave.

Eliminazione o aggiornamento degli elementi di configurazione

Un ID temporaneo o vuoto può essere utilizzato dal client quando si chiama un metodo per eliminare o aggiornare una voce. In tal caso, è necessario impostare il tipo CI e gli attributi chiave che identificano il CI.

Eliminazione o aggiornamento delle relazioni

Quando si eliminano o aggiornano le relazioni, l'ID della relazione può essere vuoto, temporaneo o reale.

Se un ID del CI è temporaneo, è necessario passare il CI nella raccolta referencedCIs e specificare i relativi attributi chiave. Per i dettagli consultare referencedCIs in "CIsAndRelationsUpdates" a pag. 304.

Inserimento dei nuovi elementi di configurazione nel CMDB

Per inserire un nuovo CI è possibile utilizzare un ID vuoto o un ID temporaneo. Tuttavia, se l'ID è vuoto, il server non può restituire l'ID CMDB reale nella struttura createIDsMap poiché non è presente alcun clientID. Per i dettagli consultare "addCIsAndRelations" a pag. 322 e "Metodi di query UCMDB" a pag. 308

Inserimento delle nuove relazioni nel CMDB

L'ID della relazione può essere temporaneo o vuoto. Tuttavia, se la relazione è nuova ma gli elementi di configurazione su una estremità della relazione sono già definiti nel CMDB, i CI già esistenti devono essere identificati da un ID CMDB reale o devono essere specificati in una raccolta referencedCIs.

Interrogare il modello di classe di UCMDB

I metodi del modello di classe restituiscono informazioni sui CIT e sulle relazioni. Il modello di classe viene configurato mediante Gestione tipo CI. Per i dettagli consultare "Gestione tipo CI" nella *Guida alla modellazione di HP Universal CMDB*.

Per gli esempi di utilizzo dei metodi del modello di classe consultare "Esempio di modello di classe" a pag. 357.

Questa sezione fornisce informazioni sui seguenti metodi che restituiscono informazioni sui CIT e sulle relazioni:

getClassAncestors

Il metodo getClassAncestors recupera il percorso tra il CIT fornito e la relativa radice, inclusa la radice.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
className	Nome tipo. Per i dettagli consultare "Type Name" a pag. 369.

Output

Parametro	Commenti
classHierarchy	Una raccolta di coppie di nomi classe e del nome della classe padre.
comments	Solo per uso interno.

getAllClassesHierarchy

Il metodo getAllClassesHierarchy recupera l'intera struttura del modello di classe.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.

Output

Parametro	Commenti
classesHierarchy	Una raccolta di coppie di nome della classe e nome della classe padre.
comments	Solo per uso interno.

getCmdbClassDefinition

Il metodo getCmdbClassDefinition recupera le informazioni sulla classe specificata.

Se si utilizza getCmdbClassDefinition per recuperare gli attributi chiave, è necessario interrogare anche le classi padre fino alla classe di base. getCmdbClassDefinition identifica come attributi chiave solo quegli attributi con ID_ATTRIBUTE impostato nella definizione di classe specificata da className. Gli attributi chiave ereditati non vengono riconosciuti come attributi chiave della classe specificata. Pertanto, l'elenco completo degli attributi chiave per la classe specificata è l'unione di tutte le chiavi della classe e di tutti i relativi padri, fino alla radice.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
className	Nome tipo. Per i dettagli consultare "Type Name" a pag. 369.

Output

Parametro	Commenti
cmdbClass	Definizione di classe composta da name, classType, displayLabel, description, parentName, qualificatori e attributi.
comments	Solo per uso interno.

Query per l'analisi impatto

L' Identificatore nei metodi di analisi impatto indica i dati di risposta del servizio. È univoco per la risposta corrente e viene eliminato dalla cache di memoria del server dopo 10 minuti di mancato utilizzo.

Per gli esempi di utilizzo dei metodi di analisi impatto consultare "Esempio di analisi impatto" a pag. 359.

Riferimenti

Metodi di query UCMDB

Questa sezione fornisce informazioni sui metodi seguenti:

executeTopologyQueryByName

Il metodo executeTopologyQueryByName recupera la mappa topologica corrispondente alla query specificata.

Suggerimento: la mappa contiene più informazioni ed è più facile da comprendere se l'etichetta per ciascunCINode e ciascun relationNode nel TQL è univoca. Per i dettagli consultare "Restituzione di elementi della mappa topologica non ambigui" a pag. 295.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
queryName	Nome della TQL nel CMDB con cui recuperare la mappa.
queryTypedProperties	Una raccolta di set di proprietà per recuperare gli elementi di un tipo di elemento di configurazione.

Output

Parametro	Commenti
topologyMap	Per i dettagli consultare "TopologyMap" a pag. 372.

executeTopologyQueryByNameWithParameters

Il metodo executeTopologyQueryByNameWithParameters recupera un elemento topologyMap corrispondente alla query con parametri specificata.

I valori per i parametri della query vengono passati nell'argomento parameterizedNodes argument. La TQL specificata deve avere etichette univoche definite per ciascun CINode e ciascun relationNode, in caso contrario, la chiamata del metodo non riesce.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
queryName	Nome della TQL con parametri nel CMDB per il quale recuperare la mappa.
parameterizedNodes	Le condizioni che ciascun nodo deve soddisfare per essere incluso nei risultati della query.
queryTypedProperties	Una raccolta di set di proprietà per recuperare gli elementi di un tipo di elemento di configurazione.

Output

Parametro	Commenti
topologyMap	Per i dettagli consultare "TopologyMap" a pag. 372.
chunkInfo	Per i dettagli consultare: "ChunkInfo" a pag. 372, "Elaborazione delle risposte di grandi dimensioni" a pag. 300.

executeTopologyQueryWithParameters

Il metodo executeTopologyQueryWithParameters recupera un elemento topologyMap corrispondente alla query con parametri specificata.

La query viene passata nell'argomento queryXML. I valori per i parametri della query vengono passati nell'argomento parameterizedNodes argument. La TQL deve avere etichette definite per ciascun CINode e ciascun relationNode.

Il metodo executeTopologyQueryWithParameters viene utilizzato per passare le query ad-hoc, invece di accedere a una query definita in CMDB. È possibile accedere a questo metodo quando non si dispone dell'accesso all'interfaccia UCMDB per definire una query o quando non si desidera salvare la query nel database.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
queryXML	Una stringa XML che rappresenta una TQL senza tag di risorse.
parameterizedNodes	Le condizioni che ciascun nodo deve soddisfare per essere incluso nei risultati della query.

Output

Parametro	Commenti
topologyMap	Per i dettagli consultare "TopologyMap" a pag. 372.
chunkInfo	Per i dettagli consultare "ChunkInfo" a pag. 372 e "Elaborazione delle risposte di grandi dimensioni" a pag. 300

 getChangedCIs

Il metodo getChangedCIs restituisce i dati del cambiamento per tutti i CI correlati ai CI specificati.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
ids	L'elenco degli ID dei CI radice i cui CI correlati vengono controllati per rilevare eventuali cambiamenti. Solo gli ID CMDB reali sono validi in questa raccolta.
fromDate	L'inizio del periodo in cui controllare se i CI sono cambiati.
toDate	La fine del periodo in cui controllare se i CI sono cambiati.

Output

Parametro	Commenti
changeDataInfo	Zero o più raccolte di elementi ChangedDataInfo.

getCINeighbours

Il metodo getCINeighbours restituisce i vicini immediati del CI specificato.

Ad esempio, se la query si trova sui vicini del CI A e il CI A contiene il CI B che utilizza il CI C, viene restituito il CI B ma non il CI C. Ovvero, vengono restituiti solo i vicini del tipo specificato.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
ID	L'ID del CI con cui recuperare i vicini. Questo deve essere un ID CMDB reale.
neighbourType	Il nome CIT dei vicini da recuperare. Vengono restituiti i vicini del tipo specificato e dei tipi derivati da quel tipo. Per i dettagli consultare "Type Name" a pag. 369.
CIProperties	I dati da restituire su ciascun elemento di configurazione, denominati Layout query nell'interfaccia utente. Per i dettagli consultare "TypedProperties" a pag. 303.
relationProperties	I dati da restituire su ciascuna relazione (denominati Layout query nell'interfaccia utente). Per i dettagli consultare "TypedProperties" a pag. 303

Output

Parametro	Commenti
topology	Per i dettagli consultare "Topology" a pag. 371.
comments	Solo per uso interno.

getCIsByID

Il metodo getCIsByID recupera gli elementi di configurazione tramite i relativi ID CMDB .

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
CIsTypedProperties	Raccolta typedproperties. Per i dettagli consultare "Altri elementi di specifica delle proprietà" a pag. 302.
IDs	Solo gli ID CMDB reali sono validi in questa raccolta.

Output

Parametro	Commenti
CIs	Raccolta di elementi CI.
chunkInfo	Per i dettagli consultare: "ChunkInfo" a pag. 372, "Elaborazione delle risposte di grandi dimensioni" a pag. 300.

getCIsByType

Il metodo getCIsByType restituisce la raccolta degli elementi di configurazione del tipo specificato e di tutti i tipi che ereditano dal tipo specificato.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.

Parametro	Commenti
type	Il nome classe. Per i dettagli consultare "Type Name" a pag. 369.
properties	I dati da restituire su ciascun elemento di configurazione. Per i dettagli consultare "CustomProperties" a pag. 303.

Output

Parametro	Commenti
CIs	Raccolta di elementi CI.
chunkInfo	Per i dettagli consultare: "ChunkInfo" a pag. 372, "Elaborazione delle risposte di grandi dimensioni" a pag. 300.

getFilteredCIsByType

Il metodo getFilteredCIsByType recupera i CI del tipo specificato che soddisfano le condizioni utilizzate dal metodo. Una condizione è composta da:

- un campo nome contenente il nome di una proprietà
- un campo operatore contenente un operatore di confronto
- un campo valore facoltativo contenente un valore o un elenco di valori

Tutti insieme formano un'espressione boleana:

```
<item>.property.value [operator] <condition>.value
```

Ad esempio, se il nome della condizione è root_actualdeletionperiod, il valore della condizione è 40 e l'operatore è Equal, l'istruzione boleana è:

```
<item>.root_actualdeletionperiod.value = = 40
```

La query restituisce tutti gli elementi il cui root_actualdeletionperiod è 40, presupponendo che non ci siano altre condizioni.

Se l'argomento conditionsLogicalOperator è AND, la query restituisce gli elementi che soddisfano tutte le condizione nella raccolta condizioni. Se l'argomento conditionsLogicalOperator è OR, la query restituisce gli elementi che soddisfano almeno una delle condizioni nella raccolta conditions.

La tabella seguente elenca gli operatori di confronto:

Operatore	Tipo di condizione/Commenti
ChangedDuring	Data Si tratta di un controllo a intervallo. Il valore della condizione è espresso in ore. Se il valore della proprietà della data rientra nell'intervallo di tempo in cui il metodo viene chiamato più o meno il valore della condizione, la condizione è true. Ad esempio, se il valore della condizione è 24, la condizione è true se il valore della proprietà della data è tra ieri a quest'ora e domani a quest'ora. **Nota:** il nome ChangedDuring viene mantenuto per conservare la compatibilità con le versioni precedenti. Nelle versioni precedenti, l'operatore è stato utilizzato solo con le proprietà della data/dell'ora di creazione e modifica.
Equal	Stringa e numerico
EqualIgnoreCase	Stringa
Greater	Numerico
GreaterEqual	Numerico
In	Stringa, numerico ed elenco Il valore della condizione è un elenco. La condizione è true se il valore della proprietà è uno dei valori nell'elenco.
InList	Elenco Il valore della condizione e il valore della proprietà sono elenchi. La condizione è true se tutti i valori nell'elenco della condizione vengono visualizzati anche nell'elenco proprietà dell'elemento. Ci possono essere più valori proprietà rispetto a quanto specificato nella condizione senza influenzare la verità della condizione.

Operatore	Tipo di condizione/Commenti
IsNull	Stringa, numerico ed elenco La proprietà dell'elemento non ha alcun valore. Quando viene utilizzato l'operatore IsNull, il valore della condizione viene ignorato e, in alcuni casi, può essere nullo.
Less	Numerico
LessEqual	Numerico
Like	Stringa Il valore della condizione è una sottostringa del valore della proprietà. Il valore della condizione deve essere racchiuso tra parentesi con il simbolo della percentuale (%). Ad esempio %Bi% corrisponde a Bismark e Bay of Biscay ma non a biscuit.
LikeIgnoreCase	Stringa Utilizzare l'operatore LikeIgnoreCase quando si utilizza l'operatore Like. La corrispondenza, tuttavia, non rispetta le maiuscole/minuscole. Pertanto, %Bi% corrisponde a biscuit.
NotEqual	Stringa e numerico
UnchangedDuring	Data Si tratta di un controllo a intervallo. Il valore della condizione è espresso in ore. Se il valore della proprietà della data rientra nell'intervallo di tempo in cui il metodo viene chiamato più o meno il valore della condizione, la condizione è false. Se non rientra in quell'intervallo, la condizione è true. Ad esempio, se il valore della condizione è 24, la condizione è true se il valore della proprietà della data è l'altro ieri a quest'ora e dopodomani a quest'ora. **Nota:** il nome UnchangedDuring viene mantenuto per preservare la compatibilità con le versioni precedenti. Nelle versioni precedenti, l'operatore è stato utilizzato solo con le proprietà della data/dell'ora di creazione e modifica.

Esempio di configurazione di una condizione:

```
FloatCondition fc = new FloatCondition();
FloatProp fp = new FloatProp();
fp.setName("attr_name");
fp.setValue(11);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.floatOperatorEnum.Equal);
```

Esempio di interrogazione per le proprietà ereditate:

Il CI di destinazione è sample con due attributi, name e size. sampleII estende il CI con due attributi, level e grade. Questo esempio configura una query per le proprietà di sampleII ereditate da sample specificandole per nome.

```
GetFilteredCIsByType request = new GetFilteredCIsByType()
request.setCmdbContext(cmdbContext);
request.setType("sampleII")
CustomProperties customProperties = new CustomProperties();
PropertiesList propertiesList = new PropertiesList();
propertiesList.addPropertyName("name");
propertiesList.addPropertyName("name");
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties);
```

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
type	Il nome classe. Per i dettagli consultare "Type Name" a pag. 369. Il tipo può essere uno dei tipi definiti utilizzando Gestione tipo CI. Per i dettagli consultare "Gestione tipo CI" nella *Guida alla modellazione di HP Universal CMDB*.
properties	I dati da restituire su ciascun CI (denominati Layout query nell'interfaccia utente). Per i dettagli consultare "CustomProperties" a pag. 303.

Parametro	Commenti
conditions	Una raccolta di coppie nome-valore e di operatori che relazionano l'una all'altra. Ad esempio host_hostname like QA.
conditionsLogicalOperator	➤ **AND.** Tutte le condizioni devono essere soddisfatte. ➤ **OR**. Almeno una delle condizioni deve essere soddisfatta.

Output

Parametro	Commenti
CIs	Raccolta di elementi CI.
chunkInfo	Per i dettagli consultare "ChunkInfo" a pag. 372 e "Elaborazione delle risposte di grandi dimensioni" a pag. 300

 getQueryNameOfView

Il metodo getQueryNameOfView recupera il nome della TQL sulla quale è basata la vista specificata.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
viewName	Il nome di una vista, ovvero un sottoinsieme del modello classe in CMDB.

Output

Parametro	Commenti
queryName	Nome della TQL nel CMDB su cui si basa la vista.

getTopologyQueryExistingResultByName

Il metodo getTopologyQueryExistingResultByName recupera il risultato più recente dell'esecuzione della TQL specificata. La chiamata non esegue la TQL. Se non sono presenti risultati da un'esecuzione precedente, non viene restituito nulla.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
queryName	Il nome di una TQL.
queryTypedProperties	Una raccolta di set di proprietà per recuperare gli elementi di un tipo di elemento di configurazione specifico.

Output

Parametro	Commenti
queryName	Nome della TQL nel CMDB su cui si basa la vista.

getTopologyQueryResultCountByName

Il metodo getTopologyQueryResultCountByName recupera il numero di istanza di ciascun nodo corrispondente alla query specificata.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
queryName	Il nome di una TQL.
countInvisible	Se true, l'output include i CI definiti come invisibili nella query.

Output

Parametro	Commenti
queryName	Nome della TQL nel CMDB su cui si basa la vista.

pullTopologyMapChunks

Il metodo pullTopologyMapChunks recupera uno dei blocchi contenenti la risposta a un metodo.

Ciascun blocco contiene un elemento topologyMap che fa parte della risposta. Il primo blocco ha il numero 1, pertanto il contatore del ciclo di recupero passa da 1 a *<response object>*.getChunkInfo().getNumberOfChunks().

Per i dettagli consultare "ChunkInfo" a pag. 372 e "Interrogare il CMDB" a pag. 299

L'applicazione client deve essere in grado di gestire le mappe parziali. Vedere il seguente esempio di gestione di una raccolta CI e l'esempio di unione di blocchi in una mappa in "Esempio di query" a pag. 337.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
ChunkRequest	Il numero del blocco da recuperare e il ChunkInfo restituito dal metodo di query.

Output

Parametro	Commenti
topologyMap	Per i dettagli consultare "TopologyMap" a pag. 372.
comments	Solo per uso interno.

Esempio di gestione blocchi:

```
GetCIsByType request =
    new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
    ucmdbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1 ; j < response.getChunkInfo().getNumberOfChunks() ; j++) {
     chunkRequest.setChunkNumber(j);
     PullTopologyMapChunks req = new PullTopologyMapChunks(cmdbContext,
chunkRequest);
     PullTopologyMapChunksResponse res =
          ucmdbService.pullTopologyMapChunks(req);
          for(int m=0 ;
               m < res.getTopologyMap().getCINodes().sizeCINodeList() ;
               m++) {
               CIs cis =
                    res.getTopologyMap().getCINodes().getCINode(m).getCIs();
               for(int i=0 ; i < cis.sizeCIList() ; i++) {
                    // your code to process the CIs
               }
          }
}
```

releaseChunks

Il metodo releaseChunks libera la memoria dei blocchi che contengono i dati della query.

Suggerimento: il server elimina i dati dopo dieci minuti. La chiamata di questo metodo di eliminazione dati subito dopo la lettura conserva le risorse del server.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
chunksKey	L'identificatore dei dati sul server con blocchi. La chiave è un elemento di ChunkInfo.

Metodi di aggiornamento di UCMDB

Questa sezione fornisce informazioni sui metodi seguenti:

- "addCIsAndRelations" a pag. 322
- "addCustomer" a pag. 323
- "deleteCIsAndRelations" a pag. 324
- "removeCustomer" a pag. 324
- "updateCIsAndRelations" a pag. 324

addCIsAndRelations

Il metodo addCIsAndRelations aggiunge o aggiorna i CI e le relazioni.

Se i CI o le relazioni non esistono nel CMDB, questi vengono aggiunti e le relative proprietà vengono impostate in base al contenuto dell'argomento CIsAndRelationsUpdates.

Se i CI o le relazioni esistono nel CMDB, questi vengono aggiornati con i nuovi dati se updateExisting è **true**.

Se updateExisting è **false**, CIsAndRelationsUpdates non può fare riferimento alle relazioni o agli elementi di configurazione esistenti. Eventuali tentativi di riferimento agli elementi esistenti quando updateExisting è false genereranno un'eccezione.

Se updateExisting è **true**, l'operazione di aggiunta o aggiornamento viene eseguita senza convalidare i CI, indipendentemente dal valore di ignoreValidation.

Se updateExisiting è **false** e ignoreValidation è **true**, l'operazione di aggiunta viene eseguita senza eseguire la convalida dei CI.

Se updateExisiting è **false** e ignoreValidation è **false**, i CI vengono convalidati prima dell'operazione di aggiunta.

Le relazioni non vengono mai convalidate.

CreatedIDsMap è una mappa o un dizionario di tipo ClientIDToCmdbID che connette gli ID temporanei del client agli ID CMDB reali.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
updateExisting	Impostare *true* per aggiornare gli elementi esistenti in CMDB. Impostare su *false* per generare un'eccezione in caso di elemento già esistente.
CIsAndRelationsUpdates	Gli elementi da aggiornare o creare. Per i dettagli consultare "CIsAndRelationsUpdates" a pag. 304.
ignoreValidation	Se true, non viene eseguito alcun controllo prima di aggiornare il CMDB.

Output

Parametro	Commenti
CreatedIDsMap	La mappa degli ID client sugli ID CMDB. Per i dettagli consultare "addCIsAndRelations" a pag. 322.
comments	Solo per uso interno.

addCustomer

Il metodo addCustomer aggiunge un cliente.

Input

Parametro	Commenti
CustomerID	ID numerico del cliente.

deleteCIsAndRelations

Il metodo deleteCIsAndRelations rimuove le relazioni e gli elementi di configurazione specificati dal CMDB.

Quando un CI viene eliminato e il CI è un'estremità di uno o più elementi Relation, vengono eliminati anche questi elementi Relation.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
CIsAndRelationsUpdates	Gli elementi da eliminare. Per i dettagli consultare "CIsAndRelationsUpdates" a pag. 304.

removeCustomer

Il metodo removeCustomer elimina un record cliente.

Input

Parametro	Commenti
CustomerID	ID numerico del cliente.

updateCIsAndRelations

Il metodo updateCIsAndRelations aggiorna le relazioni e i CI specificati.

L'aggiornamento utilizza i valori proprietà dell'argomento CIsAndRelationsUpdates. Se un CI o una relazione non esiste nel CMDB, viene generata un'eccezione.

CreatedIDsMap è una mappa o un dizionario di tipo ClientIDToCmdbID che connette gli ID temporanei del client agli ID CMDB reali.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
CIsAndRelationsUpdates	Gli elementi da aggiornare. Per i dettagli consultare "CIsAndRelationsUpdates" a pag. 304.
ignoreValidation	Se true, non viene eseguito alcun controllo prima di aggiornare il CMDB.

Output

Parametro	Commenti
CreatedIDsMap	La mappa degli ID client sugli ID CMDB. Per i dettagli consultare "addCIsAndRelations" a pag. 322.

Metodi dell'analisi impatto di UCMDB

Questa sezione fornisce informazioni sui metodi seguenti:

- "calculateImpact" a pag. 325
- "getImpactPath" a pag. 326
- "getImpactRulesByNamePrefix" a pag. 327

calculateImpact

Il metodo calculateImpact calcola quali CI sono influenzati da un dato CI in base alle regole definite nel CMDB.

Questo mostra l'effetto di un'attivazione evento della regola. L'output identifier di calculateImpact viene utilizzato come input per getImpactPath.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
impactCategory	Il tipo di evento che attiverà la regola simulata.
IDs	Raccolta di elementi ID.
impactRulesNames	Raccolta di elementi ImpactRuleName.
severity	Gravità dell'evento di attivazione.

Output

Parametro	Commenti
impactTopology	Per i dettagli consultare "Topology" a pag. 371.
identifier	La chiave per la risposta del server.

 getImpactPath

Il metodo getImpactPath recupera il grafico topologico del percorso tra i CI interessati e il CI che lo influenza.

L'output identifier di calculateImpact è utilizzato come argomento di input identifier di getImpactPath.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
identifier	La chiave per la risposta del server restituita da calculateImpact.
relation	Una Relation basata su una delle ShallowRelationrestituite da calculateImpact nell'elemento impactTopology.

Output

Parametro	Commenti
impactPathTopology	Una raccolta CI e una raccolta ImpactRelations.
comments	Solo per uso interno.

Un elemento ImpactRelations è composto da un ID, type, end1ID, end2ID, una rule e una action.

 getImpactRulesByNamePrefix

Il metodo getImpactRulesByNamePrefix recupera le regole utilizzando un filtro prefisso.

Questo metodo viene applicato alle regole di impatto denominate con un prefisso indicante il contesto di applicazione, ad esempio SAP_myrule, ORA_myrule e così via. Questo metodo filtra tutti i nomi delle regole di impatto per quelle regole che iniziano con il prefisso specificato dall'argomento ruleNamePrefixFilter.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pag. 367.
ruleNamePrefixFilter	Stringa contenente le prime lettere dei nomi delle regole di cui trovare la corrispondenza.

Output

Parametro	Commenti
impactRules	impactRules è composto da zero o più impactRule. Un impactRule, che specifica l'effetto di un cambiamento, è composto da ruleName, description, queryName e isActive.

Gestione flusso di dati

Questa sezione contiene un elenco delle operazioni del servizio Web e un breve riepilogo del relativo utilizzo. Per la documentazione completa della richiesta e della risposta di ciascuna operazione consultare *Riferimento per lo schema di Gestione flusso di dati*.

In questa sezione vengono trattati i seguenti argomenti:

Gestione dei metodi di processo GFD

- **activateJob**

  Attiva il processo specificato.

- **deactivateJob**

  Disattiva il processo specificato.

- **dispatchAdHocJob**

  Invia un processo sulla sonda ad-hoc. Il processo deve essere attivo e contenere il CI trigger specificato.

- **getDiscoveryJobsNames**

  Restituisce l'elenco dei nomi di processo.

- **isJobActive**

  Controlla se il processo è attivo.

Gestione dei metodi di attivazione

- **addTriggerCI**

  Aggiunge un nuovo CI trigger al processo specificato.

➤ **addTriggerTQL**

Aggiunge una nuova TQL trigger al processo specificato.

➤ **disableTriggerTQL**

Impedisce alla TQL di attivare il processo ma non lo rimuove permanentemente dall'elenco delle query che attivano il processo.

➤ **removeTriggerCI**

Rimuove il CI specificato dall'elenco dei CI che attivano il processo.

➤ **removeTriggerTQL**

Rimuove la TQL specificata dall'elenco delle query che attivano il processo.

➤ **setTriggerTQLProbesLimit**

Limita le sonde in cui la TQL è attiva nel processo all'elenco specificato.

Dominio e metodi dei dati della sonda

➤ **getDomainType**

Restituisce il tipo di dominio.

➤ **getDomainsNames**

Restituisce i nomi dei domini correnti.

➤ **getProbeIPs**

Restituisce gli indirizzi IP della sonda specificata.

➤ **getProbesNames**

Restituisce i nomi delle sonde nel dominio specificato.

➤ **getProbeScope**

Restituisce la definizione ambito della sonda specificata.

➤ **isProbeConnected**

Controlla se la sonda specificata è connessa.

➤ **updateProbeScope**

Imposta l'ambito della sonda specificata, sostituendo l'ambito esistente.

Metodi dei dati delle credenziali

➤ **addCredentialsEntry**

Aggiunge una voce delle credenziali al protocollo specificato per il dominio specificato.

➤ **getCredentialsEntriesIDs**

Restituisce gli ID delle credenziali definite per il protocollo specificato.

➤ **getCredentialsEntry**

Restituisce le credenziali definite per il protocollo specificato. Gli attributi crittografati vengono restituiti vuoti.

➤ **removeCredentialsEntry**

Rimuove le credenziali specificate dal protocollo.

➤ **updateCredentialsEntry**

Imposta i nuovi valori per le proprietà della voce delle credenziali specificate.

Metodi di aggiornamento dati

➤ **rediscoverCIs**

Individua i trigger che hanno rilevato gli oggetti del CI specificato e riesegue questi trigger. (Tenere presente che il comando di riesecuzione ha una priorità superiore rispetto agli altri elementi pianificati).

rediscoverCIs viene eseguito in modo asincrono. Chiamare **checkDiscoveryProgress** per determinare quando la reindividuazione è completa.

➤ **checkDiscoveryProgress**

Restituisce lo stato di avanzamento della chiamata **rediscoverCIs** più recente sugli ID specificati. La risposta è un valore compreso tra 0 e 1. Quando la risposta è 1, la chiamata **rediscoverCIs** è completata.

➤ **rediscoverViewCIs**

Individua i trigger che hanno creato i dati per popolare la vista specificata e riesegue quei trigger. (Tenere presente che il comando di riesecuzione ha una priorità superiore rispetto agli altri elementi pianificati).

rediscoverViewCIs viene eseguito in modo asincrono. Chiamare **checkViewDiscoveryProgress** per determinare quando la reindividuazione è completa.

- **checkViewDiscoveryProgress**

  Restituisce lo stato di avanzamento della chiamata **rediscoverViewCIs** più recente sulla vista specificata. La risposta è un valore compreso tra 0 e 1. Quando la risposta è 1, la chiamata **rediscoverCIs** è completata.

Casi di utilizzo

I seguenti casi di utilizzo presuppongono due sistemi:

- Server HP Universal CMDB
- Un sistema di terze parti contenente un repository degli elementi di configurazione

In questa sezione vengono trattati i seguenti argomenti:

- "Popolamento del CMDB" a pag. 331
- "Interrogazione del CMDB" a pag. 332
- "Interrogazione del modello di classe" a pag. 332
- "Analisi dell'impatto di cambiamento" a pag. 332

Popolamento del CMDB

Casi di utilizzo:

- Una gestione asset di terze parti aggiorna il CMDB con le informazioni disponibili solo nella gestione asset
- Diversi sistemi di terze parti popolano il CMDB per creare un CMDB centrale che rilevi i cambiamenti ed esegua l'analisi impatto
- Un sistema di terze parti crea gli elementi di configurazione e le relazioni in base alla logica aziendale di terze parti per sfruttare le capacità di query di CMDB

Interrogazione del CMDB

Casi di utilizzo:

- Un sistema di terze parti ottiene gli elementi di configurazione e le relazioni che rappresentano il sistema SAP recuperando i risultati della TQL SAP.
- Un sistema di terze parti ottiene l'elenco dei server Oracle aggiunti o cambiati nelle ultime cinque ore
- Un sistema di terze parti ottiene l'elenco dei server il cui nome host contiene la sottostringa *lab*
- Un sistema di terze parti trova gli elementi correlati a un dato CI ottenendo i relativi vicini

Interrogazione del modello di classe

Casi di utilizzo:

- Un sistema di terze parti consente agli utenti di specificare un insieme di dati da recuperare dal CMDB. È possibile creare un'interfaccia utente sul modello di classe per mostrare agli utenti le possibili proprietà e richiedere loro i dati necessari. L'utente può scegliere le informazioni da recuperare.
- Un sistema di terze parti esplora il modello di classe quando l'utente non può accedere all'interfaccia utente di UCMDB

Analisi dell'impatto di cambiamento

Caso di utilizzo:

Un sistema di terze parti emette un elenco dei servizi aziendali che potrebbero essere impattati da un cambiamento su un host specifico.

Esempi

In questa sezione vengono trattati i seguenti argomenti:

Classe base di esempio

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.services.UcmdbService;
import com.hp.ucmdb.generated.services.UcmdbServiceStub;
import com.hp.ucmdb.generated.types.CmdbContext;
import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;
```

```
import org.apache.axis2.transport.http.HttpTransportProperties;

import java.net.MalformedURLException;
import java.net.URL;
```

```
/**
* User: hbarkai
* Date: Jul 12, 2007
*/
abstract class Demo {
```

```
UcmdbService stub;
CmdbContext context;
```

```
public void initDemo() {
    try {
        setStub(createUcmdbService("admin", "admin"));
        setContext();
    } catch (Exception e) {
        //handle exception
    }
}
```

```
public UcmdbService getStub() {
    return stub;
}
```

```
public void setStub(UcmdbService stub) {
    this.stub = stub;
}
```

```
public CmdbContext getContext() {
    return context;
}
```

```
public void setContext() {
    CmdbContext context = new CmdbContext();
    context.setCallerApplication("demo");
    this.context = context;
}
```

```
//connection to service  - for axis2/jibx client

private static final String PROTOCOL = "http";
private static final String HOST_NAME = "host_name";
private static final int PORT = 8080;
private static final String FILE = "/axis2/services/UcmdbService";

protected UcmdbService createUcmdbService
    (String username, String password) throws Exception{
        URL url;
        UcmdbServiceStub serviceStub;
```

```
        try {
             url = new URL
                 (Demo.PROTOCOL, Demo.HOST_NAME,
                 Demo.PORT, Demo.FILE);
             serviceStub = new UcmdbServiceStub(url.toString());
             HttpTransportProperties.Authenticator auth =
                 new HttpTransportProperties.Authenticator();
             auth.setUsername(username);
             auth.setPassword(password);
serviceStub._getServiceClient().getOptions().setProperty
    (HTTPConstants.AUTHENTICATE,auth);
```

```
    } catch (AxisFault axisFault) {
            throw new Exception
            ("Failed to create SOAP adapter for "
                + Demo.HOST_NAME , axisFault);
```

```
        } catch (MalformedURLException e) {

            throw new Exception
                ("Failed to create SOAP adapter for "
                    + Demo.HOST_NAME, e);
        }
        return serviceStub;
 }
}
```

Esempio di query

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.query.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.services.UcmdbService;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.props.*;

import java.rmi.RemoteException;
```

```
public class QueryDemo extends Demo{

    UcmdbService stub;
    CmdbContext context;
```

```
    public void getCIsByTypeDemo() {
        GetCIsByType request = new GetCIsByType();
        //set cmdbcontext
        CmdbContext cmdbContext = getContext();
        request.setCmdbContext(cmdbContext);
        //set CIs type
        request.setType("anyType");
        //set CIs propeties to be retrieved
        CustomProperties customProperties = new CustomProperties();
        PredefinedProperties predefinedProperties =
            new PredefinedProperties();
        SimplePredefinedProperty simplePredefinedProperty =
            new SimplePredefinedProperty();
        simplePredefinedProperty.setName
            (SimplePredefinedProperty.nameEnum.DERIVED);
        SimplePredefinedPropertyCollection
            simplePredefinedPropertyCollection =
                new SimplePredefinedPropertyCollection();
```

```
        simplePredefinedPropertyCollection.addSimplePredefinedProperty
            (simplePredefinedProperty);
        predefinedProperties.setSimplePredefinedProperties
            (simplePredefinedPropertyCollection);
        customProperties.setPredefinedProperties(predefinedProperties);
        request.setProperties(customProperties);
        try {
            GetCIsByTypeResponse response =
                getStub().getCIsByType(request);
            TopologyMap map =
                getTopologyMapResultFromCIs
                    (response.getCIs(), response.getChunkInfo());
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }
```

```
    public void getCIsByIdDemo() {
        GetCIsById request = new GetCIsById();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set ids
        ID id1 = new ID();
        id1.setBase("cmdbobjectidCIT1");
        ID id2 = new ID();
        id2.setBase("cmdbobjectidCIT2");
        IDs ids = new IDs();
        ids.addID(id1);
        ids.addID(id2);
        request.setIDs(ids);
        //set CIs properties to be retrieved
        TypedPropertiesCollection properties =
            new TypedPropertiesCollection();
```

```
    TypedProperties typedProperties1 =
            new TypedProperties();
        typedProperties1.setType("CIT1");
```

```
        CustomTypedProperties customProperties1 =
            new CustomTypedProperties();
        PredefinedTypedProperties predefinedProperties1 =
            new PredefinedTypedProperties();
        SimpleTypedPredefinedProperty simplePredefinedProperty1 =
            new SimpleTypedPredefinedProperty();
        simplePredefinedProperty1.setName
            (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
        SimpleTypedPredefinedPropertyCollection
            simplePredefinedPropertyCollection1 =
                new SimpleTypedPredefinedPropertyCollection();
        simplePredefinedPropertyCollection1
            .addSimpleTypedPredefinedProperty
                (simplePredefinedProperty1);
```

```
predefinedProperties1.
        setSimpleTypedPredefinedProperties
            (simplePredefinedPropertyCollection1);
        customProperties1.
            setPredefinedTypedProperties
                (predefinedProperties1);
        typedProperties1.setProperties(customProperties1);
        properties.addTypedProperties(typedProperties1);
```

```
        TypedProperties typedProperties2 =
            new TypedProperties();
        typedProperties2.setType("CIT2");
        CustomTypedProperties customProperties2 =
            new CustomTypedProperties();
        PredefinedTypedProperties predefinedProperties2 =
            new PredefinedTypedProperties();
        SimpleTypedPredefinedProperty simplePredefinedProperty2 =
            new SimpleTypedPredefinedProperty();
        simplePredefinedProperty2.setName
            (SimpleTypedPredefinedProperty.nameEnum.NAMING);
        SimpleTypedPredefinedPropertyCollection
            simplePredefinedPropertyCollection2 =
                new SimpleTypedPredefinedPropertyCollection();
```

```
        simplePredefinedPropertyCollection2.
            addSimpleTypedPredefinedProperty
                (simplePredefinedProperty2);
```

```
        predefinedProperties2.setSimpleTypedPredefinedProperties
            (simplePredefinedPropertyCollection2);
        customProperties2.setPredefinedTypedProperties
            (predefinedProperties2);
        typedProperties2.setProperties(customProperties2);
        properties.addTypedProperties(typedProperties2);
```

```
    request.setCIsTypedProperties(properties);
        try {
            GetCIsByIdResponse response =
                getStub().getCIsById(request);
            CIs cis = response.getCIs();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getFilteredCIsByTypeDemo() {
        GetFilteredCIsByType request = new GetFilteredCIsByType();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set CIs type
        request.setType("anyType");
        //sets Filter conditions
        Conditions conditions = new Conditions();
        IntConditions intConditions = new IntConditions();
        IntCondition intCondition = new IntCondition();
        IntProp intProp = new IntProp();
        intProp.setName("int_attr1");
```

```
intProp.setValue(100);
intCondition.setCondition(intProp);
intCondition.setIntOperator
    (IntCondition.intOperatorEnum.Greater);
intConditions.addIntCondition(intCondition);
```

```
conditions.setIntConditions(intConditions);
request.setConditions(conditions);
//set logical operator for conditions
request.setConditionsLogicalOperator
    (GetFilteredCIsByType.conditionsLogicalOperatorEnum.AND);
//set CIs properties to be retrieved
CustomProperties customProperties =
    new CustomProperties();
PredefinedProperties predefinedProperties =
    new PredefinedProperties();
SimplePredefinedProperty simplePredefinedProperty =
    new SimplePredefinedProperty();
simplePredefinedProperty.setName
    (SimplePredefinedProperty.nameEnum.NAMING);
```

```
SimplePredefinedPropertyCollection
    simplePredefinedPropertyCollection =
        new SimplePredefinedPropertyCollection();
simplePredefinedPropertyCollection.
    addSimplePredefinedProperty
        (simplePredefinedProperty);
predefinedProperties.setSimplePredefinedProperties
    (simplePredefinedPropertyCollection);
customProperties.setPredefinedProperties
    (predefinedProperties);
```

```
request.setProperties(customProperties);
try {
    GetFilteredCIsByTypeResponse response =
        getStub().getFilteredCIsByType(request);
    TopologyMap map =
        getTopologyMapResultFromCIs
            (response.getCIs(), response.getChunkInfo());
```

```
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void executeTopologyQueryByNameDemo() {
        ExecuteTopologyQueryByName request = new
ExecuteTopologyQueryByName();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set query name
        request.setQueryName("queryName");
```

```
        try {
            ExecuteTopologyQueryByNameResponse response =
                getStub().executeTopologyQueryByName(request);
            TopologyMap map =
                getTopologyMapResult
                    (response.getTopologyMap(), response.getChunkInfo());
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
// assume the follow query was defined at UCMDB
// Query Name: exampleQuery
// Query sketch:
//                                  Host
//                                  / \
//                                ip  Disk
// Query Parameters:
//     Host-
//          host_os (like)
//     Disk-
//          disk_failures (equal)
```

```
public void executeTopologyQueryByNameWithParametersDemo() {
    ExecuteTopologyQueryByNameWithParameters request =
        new ExecuteTopologyQueryByNameWithParameters();
    CmdbContext cmdbContext = getContext();
    //set cmdbcontext
    request.setCmdbContext(cmdbContext);
    //set query name
    request.setQueryName("queryName");
    //set parameters
    ParameterizedNode hostParametrizedNode =
        new ParameterizedNode();
    hostParametrizedNode.setNodeLabel("Host");
    CIProperties parameters = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp strProp = new StrProp();
    strProp.setName("host_os");
    strProp.setValue("%2000%");
    strProps.addStrProp(strProp);
    parameters.setStrProps(strProps);
    hostParametrizedNode.setParameters(parameters);
    request.addParameterizedNodes(hostParametrizedNode);
    ParameterizedNode diskParametrizedNode =
        new ParameterizedNode();
```

```
    diskParametrizedNode.setNodeLabel("Disk");
    CIProperties parameters1 = new CIProperties();
    IntProps intProps = new IntProps();
```

```
        IntProp intProp = new IntProp();
            intProp.setName("disk_failures");
            intProp.setValue(30);
            intProps.addIntProp(intProp);
            parameters1.setIntProps(intProps);
            diskParametrizedNode.setParameters(parameters1);
```

```
            request.addParameterizedNodes(diskParametrizedNode);
            try {
                ExecuteTopologyQueryByNameWithParametersResponse
                    response =
                    getStub().executeTopologyQueryByNameWithParameters
                        (request);
                TopologyMap map =
                    getTopologyMapResult
                        (response.getTopologyMap(), response.getChunkInfo());
            } catch (RemoteException e) {
                //handle exception
            } catch (UcmdbFaultException e) {
                //handle exception
            }

        }
```

```
/       // assume the follow query was defined at UCMDB
        // Query Name: exampleQuery
        // Query sketch:
        //                              Host
        //                              / \
        //                             ip  Disk
        // Query Parameters:
        //     Host-
        //         host_os (like)
        //     Disk-
        //         disk_failures (equal)
```

```
public void executeTopologyQueryWithParametersDemo() {
    ExecuteTopologyQueryWithParameters request =
        new ExecuteTopologyQueryWithParameters();
    CmdbContext cmdbContext = getContext();
    //set cmdbcontext
    request.setCmdbContext(cmdbContext);
    //set query definition
    String queryXml = "<xml that represents the query above>";
    request.setQueryXml(queryXml);
    //set parameters
    ParameterizedNode hostParametrizedNode =
        new ParameterizedNode();
```

```
hostParametrizedNode.setNodeLabel("Host");
    CIProperties parameters = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp strProp = new StrProp();
    strProp.setName("host_os");
    strProp.setValue("%2000%");
    strProps.addStrProp(strProp);
    parameters.setStrProps(strProps);
    hostParametrizedNode.setParameters(parameters);
    request.addParameterizedNodes(hostParametrizedNode);
    ParameterizedNode diskParametrizedNode =
        new ParameterizedNode();
    diskParametrizedNode.setNodeLabel("Disk");
    CIProperties parameters1 = new CIProperties();
    IntProps intProps = new IntProps();
    IntProp intProp = new IntProp();
    intProp.setName("disk_failures");
    intProp.setValue(30);
    intProps.addIntProp(intProp);
    parameters1.setIntProps(intProps);
    diskParametrizedNode.setParameters(parameters1);
    request.addParameterizedNodes(diskParametrizedNode);
```

```
        try {
            ExecuteTopologyQueryWithParametersResponse
            response = getStub().executeTopologyQueryWithParameters
                (request);
            TopologyMap map =
                getTopologyMapResult
                    (response.getTopologyMap(), response.getChunkInfo());
```

```
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getCINeighboursDemo() {
        GetCINeighbours request = new GetCINeighbours();
        //set cmdbcontext
        CmdbContext cmdbContext = getContext();
        request.setCmdbContext(cmdbContext);
        // set CI id
        ID id = new ID();
        id.setBase("cmdbobjectidCIT1");
        request.setID(id);
        //set neighbour type
        request.setNeighbourType("neighbourType");
        //set Neighbours CIs propeties to be retrieved
        TypedPropertiesCollection properties =
            new TypedPropertiesCollection();
        TypedProperties typedProperties1 = new TypedProperties();
        typedProperties1.setType("neighbourType");
        CustomTypedProperties customProperties1 =
            new CustomTypedProperties();
        PredefinedTypedProperties predefinedProperties1 =
            new PredefinedTypedProperties();
```

```
QualifierProperties qualifierProperties =
    new QualifierProperties();
qualifierProperties.addQualifierName("ID_ATTRIBUTE");
predefinedProperties1.setQualifierProperties(qualifierProperties);
customProperties1.setPredefinedTypedProperties
    (predefinedProperties1);
typedProperties1.setProperties(customProperties1);
properties.addTypedProperties(typedProperties1);
request.setCIProperties(properties);
```

```
TypedPropertiesCollection relationsProperties =
    new TypedPropertiesCollection();
TypedProperties typedProperties2 = new TypedProperties();
typedProperties2.setType("relationType");
CustomTypedProperties customProperties2 =
    new CustomTypedProperties();
```

```
PredefinedTypedProperties predefinedProperties2 =
    new PredefinedTypedProperties();
SimpleTypedPredefinedProperty simplePredefinedProperty2 =
    new SimpleTypedPredefinedProperty();
simplePredefinedProperty2.setName
```

```
    (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
SimpleTypedPredefinedPropertyCollection
    simplePredefinedPropertyCollection2 =
        new SimpleTypedPredefinedPropertyCollection();
simplePredefinedPropertyCollection2.
    addSimpleTypedPredefinedProperty
        (simplePredefinedProperty2);
predefinedProperties2.
    setSimpleTypedPredefinedProperties
        (simplePredefinedPropertyCollection2);
customProperties2.setPredefinedTypedProperties
    (predefinedProperties2);
typedProperties2.setProperties(customProperties2);
relationsProperties.addTypedProperties(typedProperties2);
request.setRelationProperties(relationsProperties);
```

```
        try {
            GetCINeighboursResponse response =
                getStub().getCINeighbours(request);
            Topology topology = response.getTopology();
    } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
//get Topology Map for chunked/non-chunked result

    private TopologyMap getTopologyMapResult(TopologyMap topologyMap, ChunkInfo
chunkInfo) {
        if(chunkInfo.getNumberOfChunks() == 0) {
            return topologyMap;
        } else {
```

```
            topologyMap = new TopologyMap();
            for(int i=1 ; i <= chunkInfo.getNumberOfChunks() ; i++) {
                ChunkRequest chunkRequest = new ChunkRequest();
                chunkRequest.setChunkInfo(chunkInfo);
                chunkRequest.setChunkNumber(i);
                PullTopologyMapChunks req =
                    new PullTopologyMapChunks();
                req.setChunkRequest(chunkRequest);
                req.setCmdbContext(getContext());
                PullTopologyMapChunksResponse res = null;
```

```
                try {
                    res = getStub().pullTopologyMapChunks(req);
                    TopologyMap map = res.getTopologyMap();
                    topologyMap = mergeMaps(topologyMap, map);
                } catch (RemoteException e) {
                    //handle exception
                } catch (UcmdbFaultException e) {
                    //handle exception
                }
            }
        }
        return topologyMap;

    }
```

```
    private TopologyMap getTopologyMapResultFromCIs(CIs cis, ChunkInfo chunkInfo)
{
        TopologyMap topologyMap = new TopologyMap();
        if(chunkInfo.getNumberOfChunks() == 0) {
            CINode ciNode = new CINode();
            ciNode.setLabel("");
            ciNode.setCIs(cis);
            CINodes ciNodes = new CINodes();
            ciNodes.addCINode(ciNode);
            topologyMap.setCINodes(ciNodes);
        } else {
```

```
            for(int i=1 ; i <= chunkInfo.getNumberOfChunks() ; i++) {
                ChunkRequest chunkRequest =
                    new ChunkRequest();
                chunkRequest.setChunkInfo(chunkInfo);
                chunkRequest.setChunkNumber(i);
                PullTopologyMapChunks req =
                    new PullTopologyMapChunks();
                req.setChunkRequest(chunkRequest);
                req.setCmdbContext(getContext());
                PullTopologyMapChunksResponse res = null;
```

```
                try {
                    res = getStub().pullTopologyMapChunks(req);
                } catch (RemoteException e) {
                    //handle exception
                } catch (UcmdbFaultException e) {
                    //handle exception
                }
                TopologyMap map = res.getTopologyMap();
                topologyMap = mergeMaps(topologyMap, map);
            }
```

```
            //release chunks
            ReleaseChunks req = new ReleaseChunks();
            req.setChunksKey(chunkInfo.getChunksKey());
            req.setCmdbContext(getContext());
```

```
            try {
                getStub().releaseChunks(req);
            } catch (RemoteException e) {
                //handle exception
            } catch (UcmdbFaultException e) {
                //handle exception
            }
        }
        return topologyMap;

    }
```

```
//=================================================
/*  WARNING merge will be correct only if a each node is given
    a unique name. This applies to both CI and Relation nodes .*/
//=================================================
    private TopologyMap mergeMaps(TopologyMap topologyMap, TopologyMap
newMap) {
        for(int i=0 ; i < newMap.getCINodes().sizeCINodeList() ; i++ ) {
            CINode ciNode = newMap.getCINodes().getCINode(i);
            boolean alreadyExist = false;
            if(topologyMap.getCINodes() == null) {
                topologyMap.setCINodes(new CINodes());
            }
```

```
        for(int j=0 ; j < topologyMap.getCINodes().sizeCINodeList() ; j++) {
            CINode ciNode2 = topologyMap.getCINodes().getCINode(j);
            if(ciNode2.getLabel().equals(ciNode.getLabel())){
```

```
                CIs cisTOAdd = ciNode.getCIs();
                CIs cis =
                    mergeCIsGroups
                    (topologyMap.getCINodes().getCINode(j).getCIs(),
                        cisTOAdd);
                topologyMap.getCINodes().getCINode(j).setCIs(cis);
                alreadyExist = true;
            }
        }
        if(!alreadyExist) {
            topologyMap.getCINodes().addCINode(ciNode);
        }

    }
```

```
    for(int i=0 ; i < newMap.getRelationNodes().sizeRelationNodeList() ; i++ ) {
        RelationNode relationNode =
            newMap.getRelationNodes().getRelationNode(i);
        boolean alreadyExist = false;
        if(topologyMap.getRelationNodes() == null) {
            topologyMap.setRelationNodes(new RelationNodes());
        }
```

```
            for(int j=0 ;
                    j < topologyMap.getRelationNodes().sizeRelationNodeList() ;
                    j++) {
                RelationNode relationNode2 =
                    topologyMap.getRelationNodes().getRelationNode(j);
                if(relationNode2.getLabel().equals(relationNode.getLabel())){
                    Relations relationsTOAdd = relationNode.getRelations();
                    Relations relations =
                        mergeRelationsGroups
                        (topologyMap.getRelationNodes().
                            getRelationNode(j).getRelations(),
                            relationsTOAdd);
                    topologyMap.getRelationNodes().
                        getRelationNode(j).setRelations(relations);
                    alreadyExist = true;
                }
            }
```

```
            if(!alreadyExist) {
                topologyMap.getRelationNodes().addRelationNode(relationNode);
            }
        }

        return topologyMap;

    }
```

```
    private Relations mergeRelationsGroups(Relations relations1, Relations relations2)
{
        for(int i=0 ; i < relations2.sizeRelationList() ; i++) {
            relations1.addRelation(relations2.getRelation(i));
        }
        return relations2;
    }
```

```
    private CIs mergeCIsGroups(CIs cis1, CIs cis2) {
        for(int i=0 ; i < cis2.sizeCIList() ; i++) {
            cis1.addCI(cis2.getCI(i));
        }
        return cis1;
    }


}
```

Esempio di aggiornamento

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.update.AddCIsAndRelations;
import com.hp.ucmdb.generated.params.update.AddCIsAndRelationsResponse;
import com.hp.ucmdb.generated.params.update.UpdateCIsAndRelations;
import com.hp.ucmdb.generated.params.update.DeleteCIsAndRelations;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.update.CIsAndRelationsUpdates;
import com.hp.ucmdb.generated.types.update.ClientIDToCmdbID;

import java.rmi.RemoteException;

public class UpdateDemo extends Demo{
```

```
    public void getAddCIsAndRelationsDemo() {
        AddCIsAndRelations request = new AddCIsAndRelations();
        request.setCmdbContext(getContext());
        request.setUpdateExisting(true);
        CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
        CIs cis = new CIs();
        CI ci = new CI();
        ID id = new ID();
        id.setBase("temp1");
        id.setTemp(true);
```

```
        ci.setID(id);
        ci.setType("host");
```

```
        CIProperties props = new CIProperties();
        StrProps strProps = new StrProps();
        StrProp strProp = new StrProp();
        strProp.setName("host_key");
        String value = "blabla";
        strProp.setValue(value);
```

```
        strProps.addStrProp(strProp);
        props.setStrProps(strProps);
        ci.setProps(props);
        cis.addCI(ci);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);
```

```
        try {
            AddCIsAndRelationsResponse response  =
                getStub().addCIsAndRelations(request);
            for(int i = 0  ;  i < response.sizeCreatedIDsMapList() ; i++) {
                ClientIDToCmdbID idsMap = response.getCreatedIDsMap(i);
                //do something
            }
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getUpdateCIsAndRelationsDemo() {
        UpdateCIsAndRelations request = new UpdateCIsAndRelations();
        request.setCmdbContext(getContext());
```

```
        CIsAndRelationsUpdates updates =
            new CIsAndRelationsUpdates();
        CIs cis = new CIs();
        CI ci = new CI();
        ID id = new ID();
```

```
        id.setBase("temp1");
        id.setTemp(true);
        ci.setID(id);
        ci.setType("host");
        CIProperties props = new CIProperties();
        StrProps strProps = new StrProps();
```

```
        StrProp hostKeyProp = new StrProp();
        hostKeyProp.setName("host_key");
        String hostKeyValue = "blabla";
        hostKeyProp.setValue(hostKeyValue);
        strProps.addStrProp(hostKeyProp);

        StrProp hostOSProp = new StrProp();
        hostOSProp.setName("host_os");
        String hostOSValue = "winXP";
        hostOSProp.setValue(hostOSValue);
        strProps.addStrProp(hostOSProp);

        StrProp hostDNSProp = new StrProp();
        hostDNSProp.setName("host_dnsname");
        String hostDNSValue = "dnsname";
        hostDNSProp.setValue(hostDNSValue);
        strProps.addStrProp(hostDNSProp);

        props.setStrProps(strProps);
        ci.setProps(props);
        cis.addCI(ci);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);

        try {
            getStub().updateCIsAndRelations(request);
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getDeleteCIsAndRelationsDemo() {
        DeleteCIsAndRelations request =
            new DeleteCIsAndRelations();
        request.setCmdbContext(getContext());
        CIsAndRelationsUpdates updates =
            new CIsAndRelationsUpdates();
        CIs cis = new CIs();
        CI ci = new CI();
        ID id = new ID();
        id.setBase("stam");
        id.setTemp(true);
        ci.setID(id);
        ci.setType("host");
```

```
        CIProperties props = new CIProperties();
        StrProps strProps = new StrProps();
        StrProp strProp1 = new StrProp();
        strProp1.setName("host_key");
        String value1 = "for_delete";
        strProp1.setValue(value1);
        strProps.addStrProp(strProp1);
        props.setStrProps(strProps);
        ci.setProps(props);
        cis.addCI(ci);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);
```

```
        try {
            getStub().deleteCIsAndRelations(request);
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }


}
```

Esempio di modello di classe

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.classmodel.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.classmodel.UcmdbClassModelHierarchy;
import com.hp.ucmdb.generated.types.classmodel.UcmdbClass;

import java.rmi.RemoteException;

public class ClassmodelDemo extends Demo{
```

```
    public void getClassAncestorsDemo() {
        GetClassAncestors request =
            new GetClassAncestors();
        request.setCmdbContext(getContext());
        request.setClassName("className");
```

```
        try {
            GetClassAncestorsResponse response  =
                getStub().getClassAncestors(request);
            UcmdbClassModelHierarchy hierarchy =
                response.getClassHierarchy();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getAllClassesHierarchyDemo() {
        GetAllClassesHierarchy request =
            new GetAllClassesHierarchy();
        request.setCmdbContext(getContext());
        try {
            GetAllClassesHierarchyResponse response  =
                getStub().getAllClassesHierarchy(request);
            UcmdbClassModelHierarchy hierarchy =
                response.getClassesHierarchy();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getCmdbClassDefinitionDemo() {
        GetCmdbClassDefinition request =
            new GetCmdbClassDefinition();
        request.setCmdbContext(getContext());
        request.setClassName("className");
```

```
        try {
            GetCmdbClassDefinitionResponse response  =
            getStub().getCmdbClassDefinition(request);
            UcmdbClass ucmdbClass = response.getUcmdbClass();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }


}
```

Esempio di analisi impatto

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.impact.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.impact.*;

import java.rmi.RemoteException;
```

```
/**
 * Date: Jul 17, 2007
 */
public class ImpactDemo extends Demo{

//Impact Rule Name : impactExample
//Impact Query:
//              Network
//                 |
//               Host
//                 |
//                IP
//Impact Action: network affect on ip ;severity 100% ; category: change
//
public void calculateImpactAndGetImpactPathDemo() {
    CalculateImpact request = new CalculateImpact();
    request.setCmdbContext(getContext());
    //set root cause ids
    IDs ids = new IDs();
    ID id = new ID();
    id.setBase("rootCauseCmdbID");
    ids.addID(id);
```

```
request.setIDs(ids);
//set impact category
request.setImpactCategory("change");
//set rule Names
ImpactRuleNames impactRuleNames = new ImpactRuleNames();
ImpactRuleName impactRuleName = new ImpactRuleName();
impactRuleName.setBase("impactExample");
impactRuleNames.addImpactRuleName(impactRuleName);
request.setImpactRuleNames(impactRuleNames);
//set severity
request.setSeverity(100);
CalculateImpactResponse response =
    new CalculateImpactResponse();
```

```
request.setIDs(ids);
//set impact category
request.setImpactCategory("change");
//set rule Names
ImpactRuleNames impactRuleNames = new ImpactRuleNames();
ImpactRuleName impactRuleName = new ImpactRuleName();
impactRuleName.setBase("impactExample");
impactRuleNames.addImpactRuleName(impactRuleName);
request.setImpactRuleNames(impactRuleNames);
//set severity
request.setSeverity(100);
CalculateImpactResponse response =
    new CalculateImpactResponse();
```

```
try {
    response  = getStub().calculateImpact(request);
} catch (RemoteException e) {
    //handle exception
```

```
    } catch (UcmdbFaultException e) {
        //handle exception
    }
    Identifier identifier= response.getIdentifier();
    Topology topology = response.getImpactTopology();
    Relation relation = topology.getRelations().getRelation(0);
    GetImpactPath request2 = new GetImpactPath();
    //set cmdb context
    request2.setCmdbContext(getContext());
    //set impact identifier
    request2.setIdentifier(identifier);
    //set shallowRelation
    ShallowRelation shallowRelation = new ShallowRelation();
    shallowRelation.setID(relation.getID());
    shallowRelation.setEnd1ID(relation.getEnd1ID());
    shallowRelation.setEnd2ID(relation.getEnd2ID());
    shallowRelation.setType(relation.getType());
    request2.setRelation(shallowRelation);
```

```
    try {
        GetImpactPathResponse response2 =
            getStub().getImpactPath(request2);
        ImpactTopology impactTopology =
            response2.getImpactPathTopology();
    } catch (RemoteException e) {
        //To change body of catch statement
        // use File | Settings | File Templates.
        e.printStackTrace();
    } catch (UcmdbFaultException e) {
        //To change body of catch statement
        // use File | Settings | File Templates.
        e.printStackTrace();
    }
}
```

```
public void getImpactRulesByGroupName() {
    GetImpactRulesByGroupName request =
        new GetImpactRulesByGroupName();
    //set cmdb context
    request.setCmdbContext(getContext());
    //set group names list
    request.addRuleGroupNameFilter("groupName1");
    request.addRuleGroupNameFilter("groupName2");
```

```
    try {
        GetImpactRulesByGroupNameResponse response  =
            getStub().getImpactRulesByGroupName(request);
        ImpactRules impactRules = response.getImpactRules();
    } catch (RemoteException e) {
        //handle exception
    } catch (UcmdbFaultException e) {
        //handle exception
    }

}
```

```
public void getImpactRulesByNamePrefix() {
        GetImpactRulesByNamePrefix request =
            new GetImpactRulesByNamePrefix();
        //set cmdb context
        request.setCmdbContext(getContext());
        //set prefixes list
        request.addRuleNamePrefixFilter("prefix1");
```

```
        try {
            GetImpactRulesByNamePrefixResponse response  =
                getStub().getImpactRulesByNamePrefix(request);
            ImpactRules impactRules = response.getImpactRules();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
}

}
```

Esempio di aggiunta di credenziali

```
import java.net.URL;

import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;

import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.DiscoveryService;
import com.hp.ucmdb.generated.services.DiscoveryServiceStub;
import com.hp.ucmdb.generated.types.BytesProp;
import com.hp.ucmdb.generated.types.BytesProps;
import com.hp.ucmdb.generated.types.CIProperties;
import com.hp.ucmdb.generated.types.CmdbContext;
import com.hp.ucmdb.generated.types.StrList;
import com.hp.ucmdb.generated.types.StrProp;
import com.hp.ucmdb.generated.types.StrProps;
```

```
public class test {
    static final String HOST_NAME = "hostname";
   static final int PORT = 8080;

    private static final String PROTOCOL = "http";
    private static final String FILE = "/axis2/services/DiscoveryService";

    private static final String PASSWORD = "admin";
    private static final String USERNAME = "admin";

    private static CmdbContext cmdbContext = new CmdbContext("ws tests");
```

```
    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest("Range IPs by ICMP",
cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);

        // Add credentilas entry for ntcmd protcol
        addNTCMDCredentialsEntry();
    }
```

```
    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(new
GetDomainsNamesRequest(cmdbContext)).getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create credentials");
            return;
        }
        String domainName = domains.getStrValue(0);

        // Create propeties with one byte param
        CIProperties newCredsProperties = new CIProperties();

        // Add password property - this is of type bytes
        newCredsProperties.setBytesProps(new BytesProps());
        setPasswordProperty(newCredsProperties);

        // Add user & domain properties - these are of type string
        newCredsProperties.setStrProps(new StrProps());
        setStringProperties("protocol_username", "test user", newCredsProperties);
        setStringProperties("ntadminprotocol_ntdomain", "test doamin",
newCredsProperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(new
AddCredentialsEntryRequest(domainName,  "ntadminprotocol",  newCredsProperties,
cmdbContext));

        System.out.println("new credentials craeted for domain: " + domainName + " in
ntcmd protocol");
    }
```

```
    private static void setPasswordProperty(CIProperties newCredsProperties) {
        BytesProp bProp = new BytesProp();
        bProp.setName("protocol_password");
        bProp.setValue(new byte[] {101,103,102,104});
        newCredsProperties.getBytesProps().addBytesProp(bProp);
    }
```

```
    private static void setStringProperties(String propertyName, String value,
CIProperties newCredsProperties) {
        StrProp strProp = new StrProp();
        strProp.setName(propertyName);
        strProp.setValue(value);
        newCredsProperties.getStrProps().addStrProp(strProp);
    }
```

```
    private static void getProbesInfo(DiscoveryService discoveryService) throws
Exception {
        GetDomainsNamesResponse result =
discoveryService.getDomainsNames(new GetDomainsNamesRequest(cmdbContext
));

        // Go over all the domains
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName = result.getDomainNames().getStrValue(0);
            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(new
GetProbesNamesRequest(domainName, cmdbContext));

            // Go over all the probes
            for (int i=0; i<probesResult.getProbesNames().sizeStrValueList(); i++) {
                String probeName = probesResult.getProbesNames().getStrValue(i);

                // Check if connected
                IsProbeConnectedResponce connectedRequest =
                    discoveryService.isProbeConnected(new
IsProbeConnectedRequest(domainName, probeName, cmdbContext));
                Boolean isConnected = connectedRequest.getIsConnected();

                // Do something …
                System.out.println("probe " + probeName + " isconnect=" +
isConnected);
            }
        }
    }
```

```
    private static DiscoveryService getDiscoveryService() throws Exception {
        DiscoveryService discoveryService = null;
        try {

            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub = new
DiscoveryServiceStub(url.toString());

            // Authenticate info
            HttpTransportProperties.Authenticator auth = new
HttpTransportProperties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);

serviceStub._getServiceClient().getOptions().setProperty(HTTPConstants.AUTHENTIC
ATE,auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service ", e);
        }

        return discoveryService;

    }
} // End class
```

Parametri generali di UCMD

Questa sezione descrive i parametri più comuni dei metodi del servizio. Per i dettagli consultare la documentazione dello schema.

In questa sezione vengono trattati i seguenti argomenti:

CmdbContext

Tutte le chiamate del servizio API del servizio Web di UCMDB richiedono un argomento CmdbContext. CmdbContext è una stringa callerApplication che identifica l'applicazione che richiama il servizio. CmdbContext è utilizzato per la registrazione e la risoluzione dei problemi.

ID

Tutti i CI e le relazioni hanno un campo ID. Questo è composto da una stringa ID che rispetta le minuscole/maiuscole e da un flag temp facoltativo, indicante se l'ID è temporaneo.

Attributi chiave

Per identificare un CI o una Relation in alcuni contesti, è possibile utilizzare gli attributi chiave al posto di un ID CMDB. Gli attributi chiave sono quegli attributi con ID_ATTRIBUTE impostato nella definizione della classe.

Nell'interfaccia utente, gli attributi chiave hanno un'icona Chiave accanto a essi nell'elenco degli Attributi del tipo elemento di configurazione dell'interfaccia utente. Per i dettagli consultare "Finestra di dialogo Modifica/Aggiungi attributo" nella *Guida alla modellazione di HP Universal CMDB*. Per informazioni sull'identificazione degli attributi chiave dall'interno dell'applicazione client API consultare "getCmdbClassDefinition" a pag. 306.

Tipi di ID

Un elemento ID può contenere un ID reale, un ID temporaneo o essere vuoto.

Un ID reale è una stringa assegnata dal CMDB che identifica un'entità all'interno del database. Un ID temporaneo può essere una stringa univoca all'interno della richiesta corrente. Un ID vuoto indica che non è assegnato alcun valore.

Un ID temporaneo può essere assegnato dal client e spesso rappresenta l'ID del CI come archiviato dal client. Non rappresenta necessariamente un'entità già creata nel CMDB. Quando un ID temporaneo viene passato dal client, se CMDB può identificare un elemento di configurazione dati esistente utilizzando le proprietà chiave del CI, tale CI viene utilizzato come adeguato per il contesto come se fosse stato identificato con un ID reale.

L'ID reale di un CI viene calcolato da CMDB in base a una combinazione del tipo CI e delle proprietà chiave. L'ID reale di una Relation è basato sul tipo di relazione, sugli ID di due CI facenti parte della relazione e sulle proprietà della chiave della relazione. Pertanto, i valori dell'attributo chiave devono essere impostati durante la creazione di CI o Relation. Se i valori delle proprietà chiave non vengono specificati durante la creazione di un CI, ci sono due possibilità:

- Se il CIT include un qualificatore RANDOM_GENERATED_ID, il server genera un ID univoco.
- Se il CIT CIT non ha un qualificatore RANDOM_GENERATED_ID, viene generata un'eccezione.

Per i dettagli consultare "Gestione tipo CI" nella *Guida alla modellazione di HP Universal CMDB*.

CIProperties

Un elemento CIProperties è composto da raccolte, ognuna della quali contenente una sequenza di elementi nome-valore che specificano le proprietà del tipo indicato dal nome della raccolta. Nessuna delle raccolte è obbligatoria, pertanto l'elemento CIProperties può contenere una combinazione di raccolte.

CIProperties sono utilizzate dagli elementi CI e Relation. Per i dettagli consultare "Elemento di configurazione (CI, Configuration item)" a pag. 369 e "Relation" a pag. 370.

Le raccolte di proprietà sono:

- datеProps - raccolta di elementi DateProp
- doubleProps - raccolta di elementi DoubleProp
- floatProps - raccolta di elementi FloatProp
- intListProps - raccolta di elementi intListProp
- intProps - raccolta di elementi IntProp
- strProps - raccolta di elementi StrProp
- strListProps - raccolta di elementi StrListProp
- longProps - raccolta di elementi LongProp
- bytesProps - raccolta di elementi BytesProp
- xmlProps - raccolta di elementi XmlProp

Type Name

Il type name è il nome classe di un tipo elemento di configurazione o di un tipo relazione. Il type name è utilizzato in un codice per riferirsi alla classe. Non deve essere confuso con il nome visualizzato che viene appunto visualizzato nell'interfaccia utente in cui la classe è menzionata ma che non ha alcun senso nel codice.

Elemento di configurazione (CI, Configuration item)

Un elemento CI è composto da un ID, un type e da una raccolta props.

Quando si utilizza Metodi di aggiornamento di UCMDB per aggiornare un CI, l'elemento ID può contenere un ID CMDB reale o una ID temporaneo assegnato dal client. Se viene utilizzando un ID temporaneo, impostare il flag temp su true. Quando si elimina un elemento l'ID può essere vuoto. Metodi di query UCMDB considera gli ID come parametri di input e restituisce ID reali nei risultati della query.

type può essere un qualsiasi type name definito nella Gestione tipo CI. Per i dettagli consultare "Gestione tipo CI" nella *Guida alla modellazione di HP Universal CMDB*.

L'elemento propsè una raccolta CIProperties. Per i dettagli consultare "CIProperties" a pag. 368.

Relation

Relation è un'entità che collega due elementi di configurazione. Un elemento Relation è composto da un ID, un type, gli identificatori dei due elementi collegati (end1ID e end2ID), e una raccolta props.

Quando si utilizza Metodi di aggiornamento di UCMDB per aggiornare una Relation, il valore dell'ID della Relation può essere un ID CMDB reale o un ID temporaneo. Quando si elimina un elemento, l'ID può essere vuoto. Metodi di query UCMDB considerano gli ID reale come parametri di input e restituiscono ID reali nei risultati della query.

Il tipo di relazione è il Type Name della classe UCMDB dalla quale viene creata l'istanza della relazione. Il tipo può essere uno dei tipi di relazione definiti nel CMDB. Per ulteriori informazioni sulle classi o sui tipi consultare "Interrogare il modello di classe di UCMDB" a pag. 305.

Per i dettagli consultare "Gestione tipo CI" nella *Guida alla modellazione di HP Universal CMDB*.

I due ID di estremità della relazione non devono essere ID vuoti poiché vengono utilizzati per creare l'ID della relazione corrente. Tuttavia, entrambi possono avere ID temporanei a loro assegnati dal client.

L'elemento propsè una raccolta CIProperties. Per i dettagli consultare "CIProperties" a pag. 368.

Parametri di output di UCMDB

Questa sezione descrive i parametri di output più comuni dei metodi del servizio. Per i dettagli consultare la documentazione dello schema.

In questa sezione vengono trattati i seguenti argomenti:

CIs

CIs è una raccolta di elementi CI.

ShallowRelation

ShallowRelation è un'entità che collega due elementi di configurazione, composta da un ID, un type e gli identificatori dei due elementi collegati (end1ID e end2ID). Il tipo di relazione è il Type Name della classe CMDB dalla quale viene creata l'istanza della relazione. Il tipo può essere uno dei tipi di relazione definiti in CMDB.

Topology

Topology è un grafico degli elementi CI e delle relazioni. Topology è composto da una raccolta CIs e da una raccolta Relations contenente uno o più elementi Relation.

CINode

CINode è composto da una raccolta CIs con una label. label in CINode è l'etichetta definita nel nodo della TQL utilizzato nella query.

RelationNode

RelationNode è un set di raccolte Relation con una label. label in RelationNode è l'etichetta definita nel nodo della TQL utilizzato nella query.

TopologyMap

TopologyMap è l'output di un calcolo di query corrispondente alla query TQL. labels in TopologyMap sono le etichette del nodo definite nella TQL utilizzato nella query.

I dati di TopologyMap vengono restituiti nella forma seguente:

- CINodes. Questo è uno o più CINode (consultare "CINode" a pag. 371).
- relationNodes. Questo è uno o più RelationNode (consultare "RelationNode" a pag. 372).

Le label in queste due strutture ordinano gli elenchi degli elementi di configurazione e delle relazioni.

ChunkInfo

Quando una query restituisce una grande quantità di dati, il server archivia i dati suddividendoli in segmenti chiamati blocchi. Le informazioni utilizzate dal client per recuperare i dati in blocchi si trovano nella struttura ChunkInfo restituita dalla query. ChunkInfo è composto dal numberOfChunks da recuperare e dal chunksKey. chunksKey è un identificatore univoco dei dati sul server per questa chiamata di query specifica.

Per i dettagli consultare "Elaborazione delle risposte di grandi dimensioni" a pag. 300.

10

HP Universal CMDB API

Questo capitolo comprende:

Concetti

Compiti

Riferimenti

Concetti

Convenzioni

In questo capitolo vengono utilizzate le seguenti convenzioni:

- **UCMDB** si riferisce a Universal Configuration Management Database. **HP Universal CMDB** si riferisce all'applicazione.
- Gli elementi e gli argomenti del metodo di UCMDB sono immessi con l'iniziale maiuscola o minuscola specificata nelle interfacce.

Utilizzo delle API HP Universal CMDB

Utilizzare il presente capitolo insieme all'API Javadoc, disponibile nella Libreria della documentazione online.

L'API HP Universal CMDBè utilizzata per integrare le applicazioni con Universal CMDB (CMDB). L'API fornisce dei metodi per:

- aggiungere, rimuovere e aggiornare CI e relazioni in CMDB
- recuperare le informazioni sul modello di classe
- eseguire scenari whatif
- recuperare informazioni sulle relazioni e sugli elementi di configurazione

I metodi di recupero delle informazioni sulle relazioni e sugli elementi di configurazione in genere utilizzano il Topology Query Language (TQL). Per i dettagli consultare "Topology Query Language" nella *Guida alla modellazione di HP Universal CMDB*.

Gli utenti dell'API HP Universal CMDB devono conoscere:

- Il linguaggio di programmazione Java
- HP Universal CMDB

In questa sezione vengono trattati i seguenti argomenti:

- "Utilizzi dell'API" a pag. 375
- "Autorizzazioni" a pag. 375

Utilizzi dell'API

L'API viene utilizzata per soddisfare un certo numero di requisiti aziendali. Ad esempio, un sistema di terze parti può richiedere al modello di classe informazioni sugli elementi di configurazione (CI) disponibili. Per ulteriori casi di utilizzo consultare "Casi di utilizzo" a pag. 380.

Autorizzazioni

L'amministratore fornisce le credenziali di accesso per la connessione all'API. Il client API richiede il nome utente e la password di un utente di integrazione definito nel CMDB. Tali utenti non sono utenti umani del CMDB, ma piuttosto applicazioni che si connettono al CMDB.

Per i dettagli consultare "Creare un utente di integrazione" a pag. 377.

Struttura generale di un'applicazione

C'è solo un valore predefinito statico, UcmdbServiceFactory. Tale valore predefinito rappresenta il punto di ingresso per un'applicazione. Il valore UcmdbServiceFactory espone i metodi getServiceProvider. Tali metodi restituiscono un'istanza dell'interfaccia **UcmdbServiceProvider**.

Il client crea altri oggetti utilizzando i metodi interfaccia. Ad esempio, per creare una nuova definizione query, il client:

1. ottiene il servizio query dall'oggetto servizio principale di CMDB
2. ottiene un oggetto valori predefiniti query da un oggetto servizio
3. ottiene una nuova definizione query dai valori predefiniti

```
UcmdbServiceProvider provider =
  UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
UcmdbService = provider.connect(provider.createCredentials(USERNAME,
  PASSWORD), provider.createClientContext("Test"));
TopologyQueryService queryService = ucmdbService.getTopologyQueryService();
TopologyQueryFactory factory = queryService.getFactory();
QueryDefinition queryDefinition = factory.createQueryDefinition("Test Query");
queryDefinition.addNode("Node").ofType("host");
Topology topology = queryService.executeQuery(queryDefinition);
System.out.println("There are " + topology.getAllCIs().size() + " hosts in uCMDB");
```

I servizi disponibili da **UcmdbService** sono:

Metodi servizio	Utilizzo
getClassModelService	Informazioni sui tipi di CI e relazioni
getDDMConfigurationService	Configurare il sistema di individuazione e gestione delle dipendenze
getDDMManagementService	Analizzare e visualizzare l'avanzamento, i risultati e gli errori del sistema di individuazione e gestione delle dipendenze
getImpactAnalysisService	Eseguire lo scenario dell'analisi impatto (anche noto come **correlazione**).
getQueryManagementService	Gestire l'accesso alle query: salvare, eliminare, elencare esistenti. Fornisce inoltre la convalida query e l'individuazione delle dipendenze delle query.
getResourceBundleManagement Service	Assegnazione di tag risorse (servizi di "creazione pacchetti". Consente la creazione esplicita di nuovi tag e la rimozione dei tag da tutte le risorse contrassegnate.
getSoftwareSignatureService	Definire gli elementi software che il sistema di individuazione e di gestione delle dipendenze deve individuare
getTopologyQueryService	Ottenere informazioni sull'universo IT
getTopologyUpdateService	Cambiare le informazioni nell'universo IT
getViewService	Visualizzare il servizio di esecuzione (eseguire definizione, eseguire salvati) e il servizio di gestione (salvare, eliminare, elencare esistenti). Fornisce inoltre la convalida e l'individuazione delle dipendenze della vista.
getViewArchiveService	Visualizzare i servizi di archiviazione risultati. Consente il salvataggio del risultato della vista corrente e il recupero dei risultati precedentemente salvati.

Il client comunica con il server tramite HTTP.

Compiti

Mettere il file .jar dell'API nel classpath

L'uso di questo insieme di API richiede il file **ucmdb-api.jar**. È possibile scaricare il file immettendo http://localhost:8080 in un browser Web e facendo clic sul collegamento **Download del client API**.

Mettere il file .jar nel classpath prima di compilare o eseguire l'applicazione.

Creare un utente di integrazione

È possibile creare un utente dedicato per le integrazioni tra gli altri prodotti e UCMDB. Questo utente consente a un prodotto che utilizza il client SDK di UCMDB di essere autentificato nel server SDK ed eseguire le API. Le applicazioni scritte con questo insieme di API devono accedere con le credenziali dell'utente di integrazione.

Attenzione: Solo un utente di integrazione può connettersi al CMDB tramite questo insieme di API. Un tentativo di connessione da parte di altri tipi di utente possono causare errori anche quando si utilizza la verifica LDAP.

Per creare un utente di integrazione:

1. Avviare il browser Web e immettere l'indirizzo del server come segue:

   http://localhost:8080/jmx-console.

   È necessario accedere con nome utente e password (i valori predefiniti sono sysadmin/sysadmin).

2. In UCMDB, fare clic su **service=Servizi di protezione UCMDB** per aprire la pagina JMX MBEAN View.

3. Individuare l'operazione **CreateIntegrationUser**. Questo metodo accetta i seguenti parametri:
   - **customerId**. L'ID utente.
   - **username**. Il nome dell'utente di integrazione.
   - **password**. La password dell'utente di integrazione.
   - **dataStoreOrigin**. Il nome del prodotto che verrà utilizzato da questo utente di integrazione.

   Le operazioni seguenti sono utili per la gestione dell'utente di integrazione:
   - **DeleteIntegrationUser**. Elimina l'utente di integrazione fornito.
   - **ExportIntegrationUser**. Esporta l'utente di integrazione in un file XML nel percorso fornito (sul computer server).
   - **getIntegrationUser**. Visualizza le informazioni dell'utente di integrazione.
   - **changeIntegrationUserPassword**. Cambia la password dell'utente di integrazione.
   - **canUserAuthenticate**. **isIntegrationUser** è **true**: l'utente di integrazione può eseguire l'autenticazione con le credenziali fornite?
4. Fare clic su **Richiama**.

   Fare clic su **Ritorna a MBean View** per creare più utenti o chiudere la console JMX.
5. Accedere a UCMDB come amministratore.
6. Dalla scheda **Amministrazione**, eseguire **Gestione pacchetti**.
7. Fare clic sull'icona **Nuovo**.
8. Immettere un nome per il nuovo pacchetto e fare clic su **Avanti**.
9. Nella scheda Selezione risorse in **Amministrazione**, fare clic su **Utenti di integrazione**.
10. Selezionare uno o più utenti da creare utilizzando la console JMX.
11. Fare clic su **Avanti** e quindi su **Fine**. Il nuovo pacchetto viene visualizzato nell'elenco Nome pacchetto in Gestione pacchetti.

12 Distribuire il pacchetto agli utenti che eseguiranno le applicazioni API.

Per i dettagli consultare "Distribuire un pacchetto" nella *Guida all'amministrazione di HP Universal CMDB*.

Nota:

L'utente di integrazione è per cliente. Per creare un utente di integrazione più complesso da utilizzare con più clienti, utilizzare **systemUser** con il flag **isSuperIntegrationUser** impostato su **true**. Utilizzare i metodi **systemUser** (**createSystemUser**, **removeSystemUser**, **showAllSystemUsers**, **changeSystemUserPassword**, **canSuperIntegrationUserAuthenticate** e così via).

Esistono due utenti di sistema predefiniti; si consiglia di cambiare le relative password dopo l'installazione attraverso il metodo **changeSystemUserPassword**

- **sysadmin**/**sysadmin**
- **UISysadmin**/**UISysadmin** (Questo è anche il superutente di integrazione **SuperIntegrationUser**).

Se si cambia la password utilizzando UISysadmin **changeSystemUserPassword**, è necessario eseguire il metodo seguente: nella console JMX, individuare il servizio **UCMDB-UI:name=Integrazione UCMDB**. Eseguire **setCMDBSuperIntegrationUser** con il nome utente e la nuova password dell'utente di integrazione.

Riferimenti

HP Universal CMDB Riferimenti API

Per la documentazione completa sulle API disponibili consultare "Introduzione alle API" a pag. 289.

Casi di utilizzo

I seguenti casi di utilizzo presuppongono due sistemi:

- Server HP Universal CMDB
- Un sistema di terze parti contenente un repository degli elementi di configurazione

In questa sezione vengono trattati i seguenti argomenti:

Popolazione di CMDB

Casi di utilizzo:

- Una gestione asset di terze parti aggiorna il CMDB con le informazioni disponibili solo nella gestione asset
- Diversi sistemi di terze parti popolano il CMDB per creare un CMDB centrale che rilevi i cambiamenti ed esegua l'analisi impatto
- Un sistema di terze parti crea gli elementi di configurazione e le relazioni in base alla logica aziendale di terze parti per sfruttare le capacità di query di UCMDB

Interrogazione del CMDB

Casi di utilizzo:

- Un sistema di terze parti ottiene gli elementi di configurazione e le relazioni che rappresentano il sistema SAP recuperando i risultati della TQL SAP.
- Un sistema di terze parti ottiene l'elenco dei server Oracle aggiunti o cambiati nelle ultime cinque ore
- Un sistema di terze parti ottiene l'elenco dei server il cui nome host contiene la sottostringa lab
- Un sistema di terze parti trova gli elementi correlati a un dato CI ottenendo i relativi vicini

Interrogazione del modello di classe

Casi di utilizzo:

- Un sistema di terze parti consente agli utenti di specificare un insieme di dati da recuperare dal CMDB. È possibile creare un'interfaccia utente sul modello di classe per mostrare agli utenti le possibili proprietà e richiedere loro i dati necessari. L'utente può scegliere le informazioni da recuperare.
- Un sistema di terze parti esplora il modello di classe quando l'utente non può accedere all'interfaccia utente di UCMDB

Analisi dell'impatto di cambiamento

Caso di utilizzo:

Un sistema di terze parti emette un elenco dei servizi aziendali che potrebbero essere impattati da un cambiamento su un host specifico.

Esempi

In questa sezione vengono trattati i seguenti argomenti:

Esempio punto di ingresso

```
final String HOST_NAME = "localhost";
final int PORT = 8080;
UcmdbServiceProvider provider =
    UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
final String USERNAME = "integration_user";
final String PASSWORD = "integration_password";
Credentials credentials =
    provider.createCredentials(USERNAME, PASSWORD),
ClientContext clientContext = provider.createClientContext("Example");
UcmdbService ucmdbService = provider.connect(credentials, clientContext);
```

Esempi di query

Gli esempi seguenti dimostrano l'ottenimento di una definizione di classe singola e l'ottenimento di un elenco di tutte le definizioni CIT e dei relativi attributi.

Recupero di una definizione di classe

```
ClassModelService classModelService
    = ucmdbService.getClassModelService();
String typeName = "disk";
ClassDefinition def =
    classModelService.getClassDefinition(typeName);
System.out.println("Type " + typeName + " is derived from type "
    + def.getParentClassName());
System.out.println("Has " + def.getChildClasses().size() +
    " derived types");
System.out.println("Defined and inherited attributes:");
for (Attribute attr : def.getAllAttributes().values()) {
    System.out.println("Attribute " + attr.getName() +
    " of type " + attr.getType());
}
```

Recupero dell'elenco di definizioni CIT e relativi attributi

Questo esempio esegue la query degli attributi di un CIT e stampa i relativi nomi e tipi.

```
ClassModelService classModelService =
     ucmdbService.getClassModelService();
for (ClassDefinition def : classModelService.getAllClasses()) {
    System.out.println("Type " + def.getName() +
        " (" + def.getDisplayName() + ") is derived from type "
        + def.getParentClassName());
    System.out.println
        ("Has " + def.getChildClasses().size() + " derived types");
    System.out.println
        ("Defined and inherited attributes:");
    for (Attribute attr : def.getAllAttributes().values()) {
        System.out.println
          ("Attribute " + attr.getName() +
        " of type " + attr.getType());
    }
}
```

Esempio di query di topologia

```
TopologyQueryService queryService =
    ucmdbService.getTopologyQueryService();
TopologyQueryFactory queryFactory =
    queryService.getFactory();
QueryDefinition queryDefinition =
    queryFactory.createQueryDefinition
        ("Get hosts with more than one network interface");
String hostNodeName = "Host";
QueryNode hostNode =

queryDefinition.addNode(hostNodeName).ofType("host").queryProperty("display_label"
);
QueryNode ipNode =
    queryDefinition.addNode("IP").ofType("ip").queryProperty("ip_address");
hostNode.linkedTo(ipNode).withLinkOfType("contained").atLeast(2);
Topology topology = queryService.executeQuery(queryDefinition);
Collection<TopologyCI> hosts = topology.getCIsByName(hostNodeName);
for (TopologyCI host : hosts) {
    System.out.println("Host " + host.getPropertyValue("display_label"));
    for (TopologyRelation relation : host.getOutgoingRelations()) {
        System.out.println
        ("  has IP " + relation.getEnd2CI().getPropertyValue("ip_address"));
    }
}
```

Esempio di aggiornamento topologia

```
TopologyUpdateService topologyUpdateService =
    ucmdbService.getTopologyUpdateService();
TopologyUpdateFactory topologyUpdateFactory =
    topologyUpdateService.getFactory();
TopologyModificationData topologyModificationData =
    topologyUpdateFactory.createTopologyModificationData();
CI host = topologyModificationData.addCI("host");
host.setPropertyValue("host_key", "test1");
CI ip = topologyModificationData.addCI("ip");
ip.setPropertyValue("ip_address", "127.0.0.10");
ip.setPropertyValue("ip_domain", "DefaultDomain");
topologyModificationData.addRelation("contained", host, ip);
topologyUpdateService.create
    (topologyModificationData, CreateMode.IGNORE_EXISTING);
```

Esempio di analisi impatto

```
ImpactAnalysisService impactAnalysisService =
    ucmdbService.getImpactAnalysisService();
ImpactAnalysisFactory impactFactory =
    impactAnalysisService.getFactory();
ImpactAnalysisDefinition definition =
    impactFactory.createImpactAnalysisDefinition();
definition.addTriggerCI(disk).withSeverity
    (impactFactory.getSeverityByName("Warning(2)"));
definition.useAllRules();
ImpactAnalysisResult impactResult =
    impactAnalysisService.analyze(definition);
AffectedTopology affectedCIs =
    impactResult.getAffectedCIs();
for (AffectedCI affectedCI : affectedCIs.getAllCIs()) {
     System.out.println("Affected " +
     affectedCI.getType() + " " + affectedCI.getId() +
     " - severity " + affectedCI.getSeverity());
}
```

Indice

F

G

H

I

J

T

U

V