HP Universal CMDB

Versione software: 10.10

Guida di riferimento per lo sviluppatore

Data di rilascio del documento: Novembre 2013

Data di rilascio del software: Novembre 2013

Informazioni legali

Garanzia

Le uniche garanzie riconosciute per i prodotti e servizi HP sono stabilite nelle dichiarazioni di garanzia esplicite allegate a tali prodotti e servizi. Nulla di quanto contenuto nel presente documento potrà essere interpretato in modo da costituire una garanzia aggiuntiva. HP non è responsabile di errori e omissioni editoriali o tecnici contenuti nel presente documento.

Le informazioni contenute nella presente documentazione sono soggette a modifiche senza preavviso.

Legenda dei diritti riservati

Questo software per computer è riservato. Per il possesso, l'uso o la copia è necessario disporre di una licenza HP valida. In conformità con le disposizioni FAR 12.211 e 12.212, il software commerciale, la documentazione del software e i dati tecnici per gli articoli commerciali sono concessi in licenza al governo degli Stati Uniti alle condizioni di licenza commerciale standard del fornitore.

Informazioni sul copyright

Informazioni sui marchi

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Aggiornamenti della documentazione

La pagina del titolo del presente documento contiene le seguenti informazioni di identificazione:

- Numero di versione software, che indica la versione del software.
- Data di rilascio del documento, che varia ad ogni aggiornamento del documento.
- Data di rilascio del software, che indica la data di rilascio di questa versione del software.

Per verificare l'esistenza di aggiornamenti recenti o per accertarsi di utilizzare la versione più recente del documento, visitare il sito:
http://h20230.www2.hp.com/selfsolve/manuals

Questo sito richiede la registrazione e l'accesso come utente HP Passport. Per registrarsi come utente HP Passport, andare all'indirizzo:
http://h20229.www2.hp.com/passport-registration.html

Oppure fare clic sul collegamento **New user registration** nella pagina di accesso di HP Passport.

È inoltre possibile ricevere versioni nuove o aggiornate abbonandosi all'apposito servizio di assistenza. Per informazioni, contattare il rappresentante commerciale di HP.

Assistenza

Visitare il sito Web dell'assistenza online HP Software all'indirizzo: **http://www.hp.com/go/hpsoftwaresupport**

Questo sito Web fornisce informazioni di contatto e dettagli sui prodotti, servizi e assistenza offerti da HP Software.

L'assistenza online di HP Software fornisce ai clienti funzionalità di auto-risoluzione dei problemi e costituisce un modo efficiente e veloce per accedere agli strumenti di assistenza tecnica interattiva necessari per gestire il proprio business. Nel sito Web dell'assistenza è possibile usufruire dei seguenti vantaggi:

- Ricerca di documenti nelle Knowledge Base
- Invio e consultazione di casi di assistenza e richieste di miglioramenti
- Download di patch software
- Gestione di contratti di assistenza
- Ricerca di recapiti di assistenza HP
- Esame delle informazioni relative ai servizi disponibili
- Partecipazione a forum di discussione con altri utenti del software
- Ricerca e iscrizione a eventi di formazione software

La maggior parte delle aree di assistenza richiede la registrazione e l'accesso come utente HP Passport. In molti casi è inoltre necessario disporre di un contratto di assistenza. Per registrarsi come utente HP Passport, andare all'indirizzo:

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

Per ulteriori informazioni sui livelli di accesso, visitare:

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

HP Software Solutions Now accede al Portale HPSW Solution and Integration. Questo sito permette di consultare la pagine di HP Product Solutions, che comprendono l'elenco completo delle integrazioni fra i prodotti HP e un elenco di processi ITIL. L'URL di questo sito Web è
http://h20230.www2.hp.com/sc/solutions/index.jsp

Informazioni sulla versione in PDF della Guida in linea

Questo documento è la versione in PDF della guida in linea. Questo file PDF viene fornito per consentire la stampa dei diversi argomenti della guida o leggere la guida online in formato PDF. Questo contenuto è stato creato per essere visualizzato come Guida in linea in un browser Web, pertanto il formato di alcuni argomenti potrebbe non essere corretto. Alcuni argomenti interattivi potrebbero non essere inclusi in questa versione in PDF. Questi argomenti possono comunque essere stampati dalla guida in linea.

Sommario

Creazione di adattatori di integrazione e individuazione

Capitolo 1: Sviluppo e scrittura degli adattatori

Questo capitolo comprende:

Panoramica dello sviluppo e della scrittura degli adattatori

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

Questo capitolo :

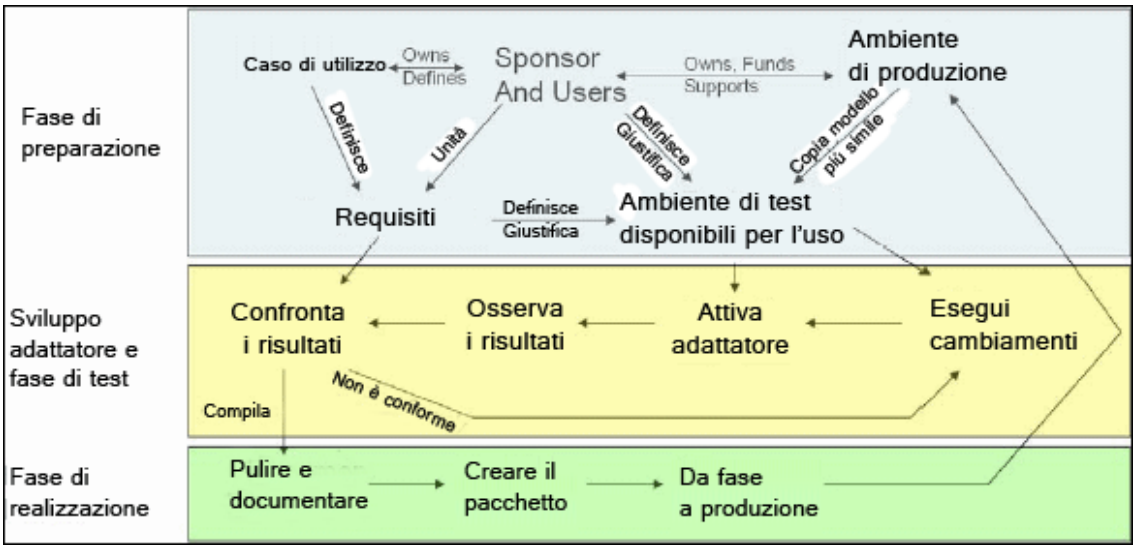
- Presuppone una conoscenza del funzionamento diHP Universal CMDB e una dimestichezza di base con gli elementi del sistema. Il suo scopo è quello di assistere durante il processo di apprendimento e non intende rappresentare una guida completa.
- Tratta le fasi di pianificazione, ricerca e implementazione 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 *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 gli argomenti seguenti:

Il 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.

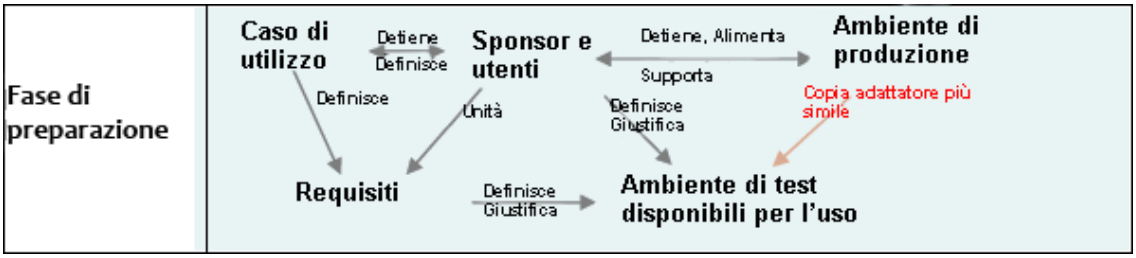


Ciascuna fase di sviluppo dell'adattatore si basa sulla precedente.

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 "Package Manager"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



La fase di Ricerca e preparazione comprende i casi di utilizzo e le esigenze aziendali principali, nonché la predisposizione delle risorse necessarie 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 di Gestione flusso di dati di HP Universal CMDB*.

2. Individuare il metodo che l'adattatore dovrà utilizzare per raccogliere i dati:

   - Utilizzare 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. Decomprimere il contenuto del pacchetto 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 "Package Manager" 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 "Package Manager" 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 presente 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 prodotto:

- 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 Assistenza HP Software.

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

Tipo di integrazione	Esempio di riferimento da riutilizzare
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 del mapping dei componenti di sistema ai CI e della loro aggiunta a CMDB è quello di fornire valore aziendale.

Non è sempre possibile utilizzare il contenuto per il mapping 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 domande seguenti:

- 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?
- Come dovrebbe essere il mapping ideale?
  - 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 in CMDB.

Un metodo potrebbe essere quello di fornire un questionario al proprietario dell'applicazione (anche 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. Il mapping di grandi quantità di dati fornisce mappe più imponenti ma che possono essere poco chiare e richiedere molto tempo di sviluppo.

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 "Panoramica dello sviluppo e della scrittura degli adattatori" a pagina 12.

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 Assistenza HP Software 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 in 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 non 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 tipi 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 alcune entità, relazioni o attributi 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 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 in **Gestione flusso di dati > Impostazioni Data Flow Probe > Riquadro domini e sonde**. Per dettagli su ogni protocollo, vedere la sezione che descrive i protocolli supportati in *HP UCMDB Discovery and Integrations Content Guide*.

È possibile aggiungere nuovi protocolli aggiungendo nuovi CI al modello. Per i dettagli contattare Assistenza HP Software.

> **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 in 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 un mapping 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 in CMDB? I dati devono essere monitorati tramite cronologia? L'origine è inaffidabile?

  Se la risposta a queste domande è si, è necessario il **Popolamento**.

- 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 in CMDB è troppo grande ma la quantità dei dati richiesta è solitamente piccola?

  Se la risposta a queste domande è si, è necessaria la **Federazione**.

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

  Se la risposta a queste domande è si, è necessario l'**Invio dati**.

> **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 Integration Studio nella *Guida di Gestione flusso di dati di HP Universal CMDB*.

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

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

   Per i dettagli consultare "Sviluppo degli adattatori Jython" a pagina 38.
2. 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 che si connettono a una singola origine o destinazione dati.

Per i dettagli consultare "Sviluppo degli adattatori Java" a pagina 149.

3. Adattatore DB generico:

   - Un adattatore astratto basato sull'adattatore Java, 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 diUCMDB.
   - Adatto per processi che si connettono a una singola origine dati.

   Per i dettagli consultare "Sviluppo degli adattatori generici del database" a pagina 71.

4. 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 sufficiente definire il mapping tra le classi di UCMDB e XML, nonché uno script Jython che invia i dati alla destinazione.
   - Adatto per processi che si connettono a una singola destinazione dati.
   - Utilizzato per l'invio dati.

   Per i dettagli consultare "Sviluppo degli adattatori Push" a pagina 186.

5. Adattatore Push generico avanzato:

   - Tutte le caratteristiche sopraelencate dell'adattatore Push generico
   - Un adattatore basato sull'elemento radice
   - Esegue il mapping tra una struttura dati ad albero di UCMDB e una struttura dati ad albero di destinazione

Per i dettagli consultare "Sviluppo degli adattatori Push generici estesi" a pagina 240.

La tabella seguente visualizza le capacità di ciascun adattatore:

Flusso/Adattatore	Adattatore Jython	Adattatore Java	Adattatore DB generico	Adattatore Push generico	Adattatore Push generico avanzato
Popolamento	✓	✓	✓	×	×
Federazione	×	✓	✓	×	×
Invio dati	×	✓	×	✓	✓

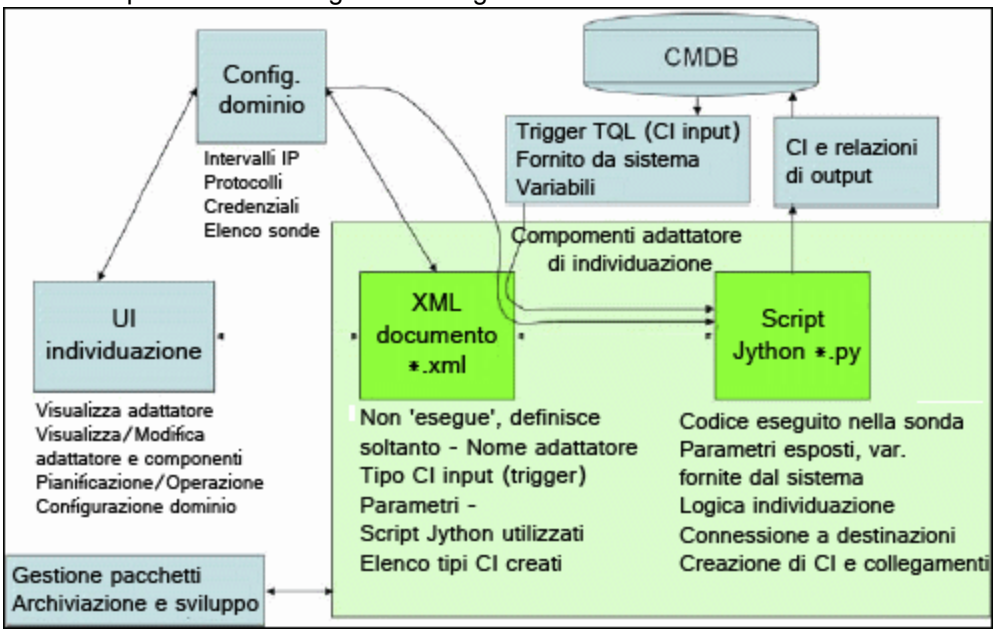
Sviluppo del contenuto di individuazione

In questa sezione vengono trattati gli argomenti seguenti:

- "Adattatori di individuazione e componenti correlati " nel seguito
- "Separazione degli adattatori" alla pagina successiva

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 in 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. Durante la scrittura dell'adattatore si utilizzano le pagine Universal Discovery, Gestione adattatori e Impostazioni Data Flow Probe.

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

Separazione degli adattatori

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 questo mapping, viene mappato un componente di connessione JDBC. L'adattatore B 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.

Implementare un adattatore di individuazione

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

1. **Creare un adattatore.**

   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 "Passaggio 1: Creare un adattatore" a pagina 28.

2. **Creare un processo di individuazione.**

   È possibile configurare un processo con le informazioni di pianificazione e una query trigger. Per i dettagli consultare "Passaggio 2: Assegnare un processo all'adattatore" a pagina 35.

3. **Modificare il 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 un codice Jython" a pagina 36.

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.

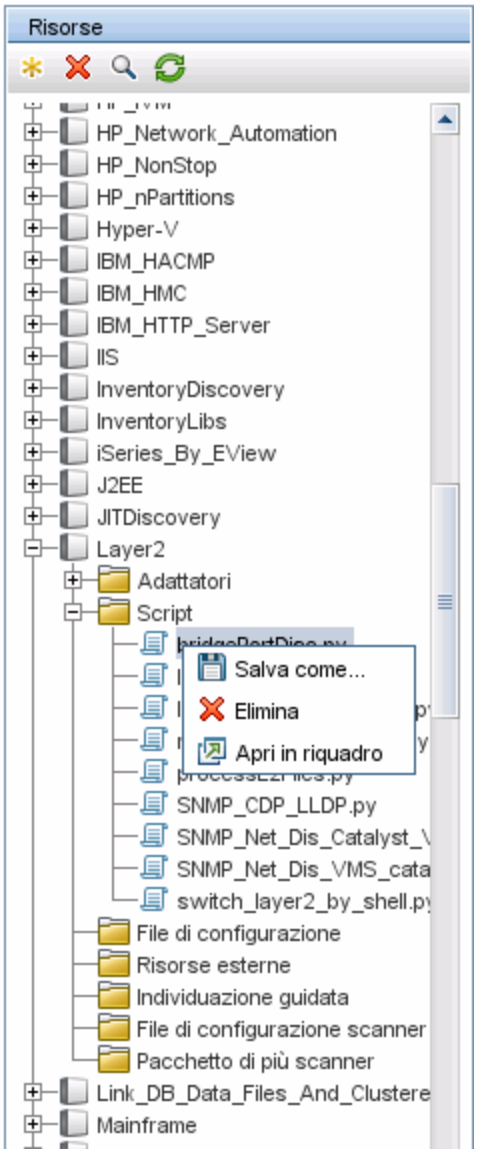
Codice adattatore

L'implementazione effettiva della connessione al sistema remoto, dell'interrogazione dei dati e del

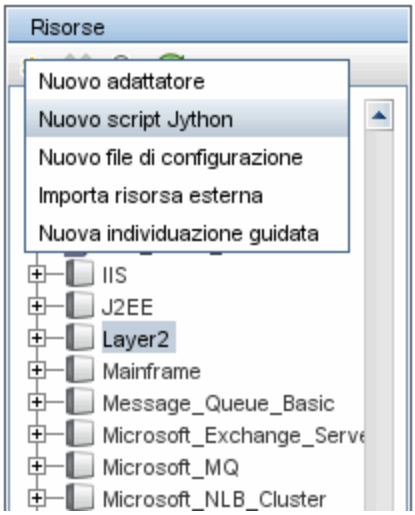
mapping come dati di CMDB viene eseguita dal codice Jython. Ad esempio, il codice contiene la logica per la connessione a un database e per l'estrazione dei dati da quest'ultimo. In tal caso, il codice si aspetta di ricevere un URL JDBC, un nome utente, una password, una porta e così via. Questi parametri 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.

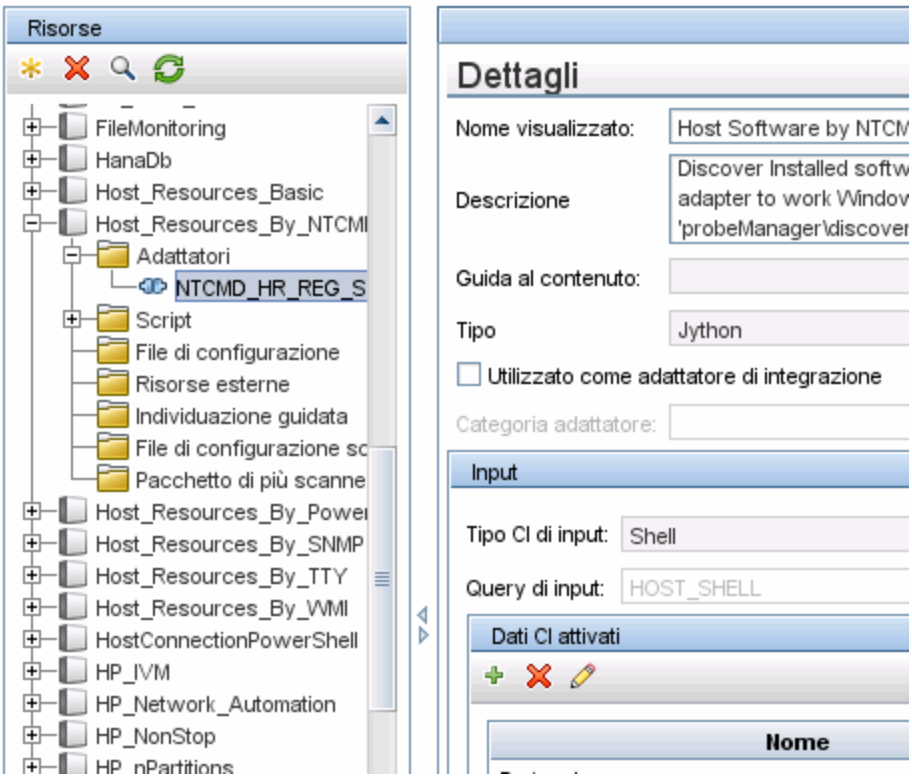
Un adattatore può contenere un elenco di script da utilizzare durante l'esecuzione dell'individuazione. Quando si crea un nuovo adattatore, viene creato solitamente un nuovo script che viene assegnato all'adattatore. Un nuovo script include i modelli di base, tuttavia è possibile utilizzare uno degli altri script come modello 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 un codice Jython" a pagina 36. 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. Talvolta vengono passati al codice anche dati di CI correlati. Per i dettagli consultare Finestra CI correlati nella *Guida di Gestione 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 Concetti di Gestione flusso di dati nella *Guida di Gestione flusso di dati di HP Universal CMDB*.

In questa sezione vengono trattati i seguenti argomenti:

1. ## 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 a 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 al TQL di input. Per i dettagli consultare Come aggiungere nodi query e relazioni a una query TQL 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.

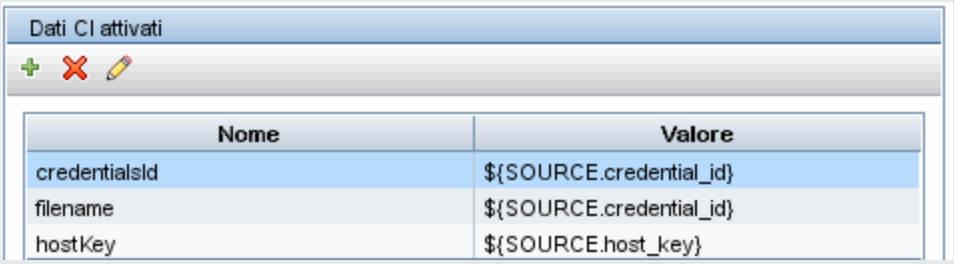
- Se il valore dei dati di destinazione è un elenco, è possibile definire il numero di elementi che verranno inviati alla sonda dall'elenco. Per definirlo, aggiungere due punti dopo il valore predefinito, seguiti dal numero di elementi. Se non vi è un valore predefinito per i dati di destinazione, immettere due volte due punti.

  Ad esempio, se vengono immessi i dati di destinazione seguenti: `name=portId, value= ${PHYSICALPORT.root_id:NA:1}` o `name=portId, value= ${PHYSICALPORT.root_id::1}`, verrà inviata alla sonda solo la prima porta dell'elenco.

Esempio di sostituzione delle variabili con i dati effettivi:

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

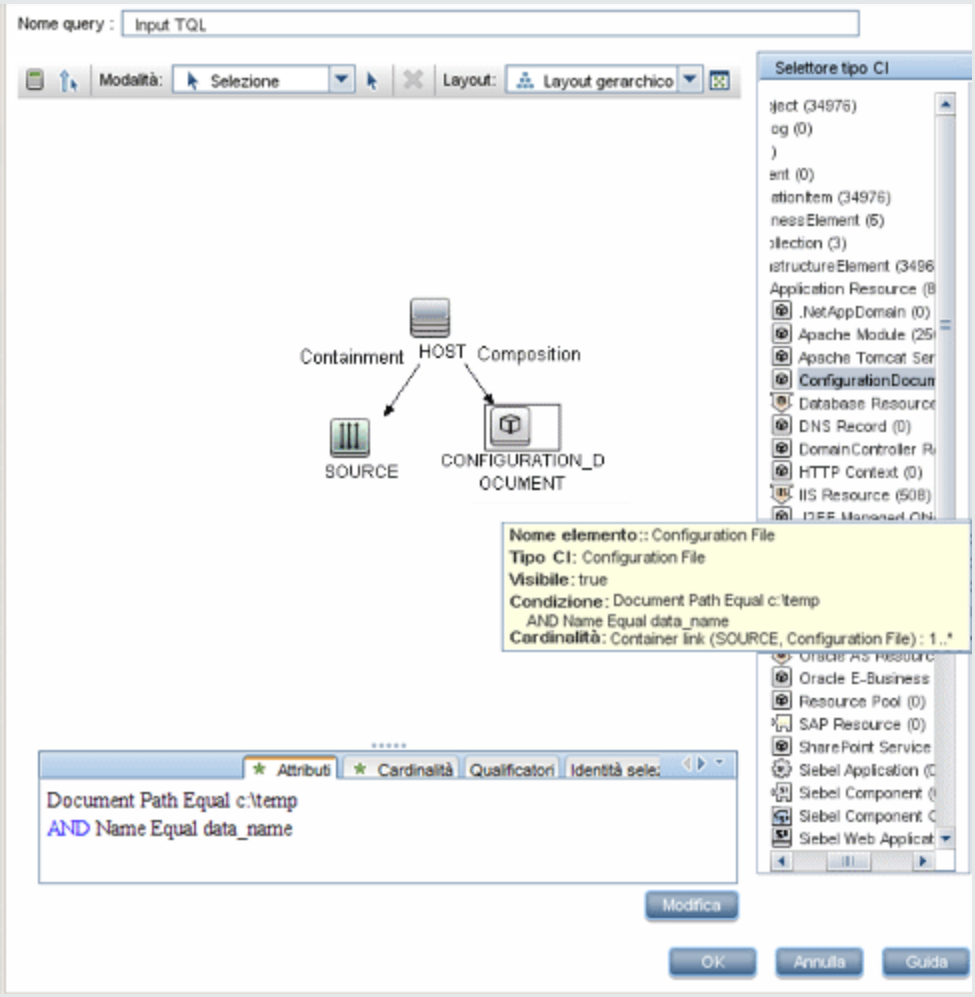
I dati CI attivati per il CI **IpAddress** includono una variabile `fileName`. Questa variabile consente la sostituzione del nodo **CONFIGURATION_DOCUMENT** nel TQL di input con i valori effettivi del file di configurazione ubicato in un host:



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 DFM Framework per recuperare i dati effettivi delle variabili definite:

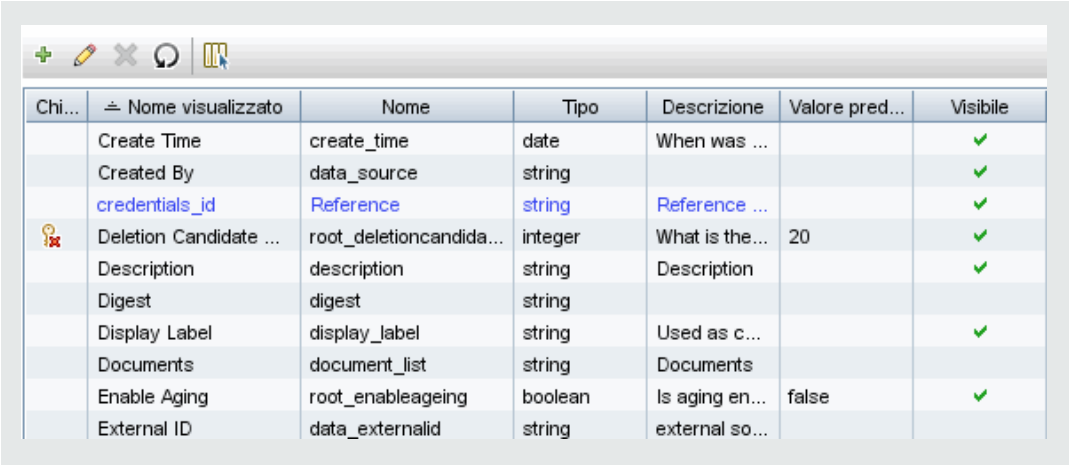
```
Framework.getTriggerCIData ('ip_address')
```

Le variabili `fileName` e `path` utilizzano gli attributi `data_name` e `document_path` del nodo del file **CONFIGURATION_DOCUMENT** (definito nell'Editor di query di input - vedere l'esempio precedente).



Fare clic sulla miniatura per visualizzare l'immagine a dimensioni intere.

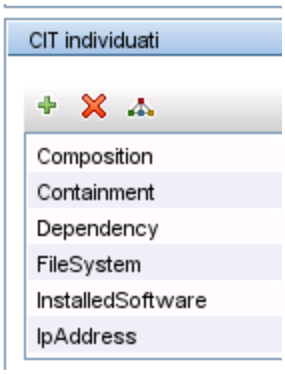
Le variabili `Protocol`, `credentialsId` e `ip_address` utilizzano gli attributi `root_class`, `credentials_id` e `application_ip`:



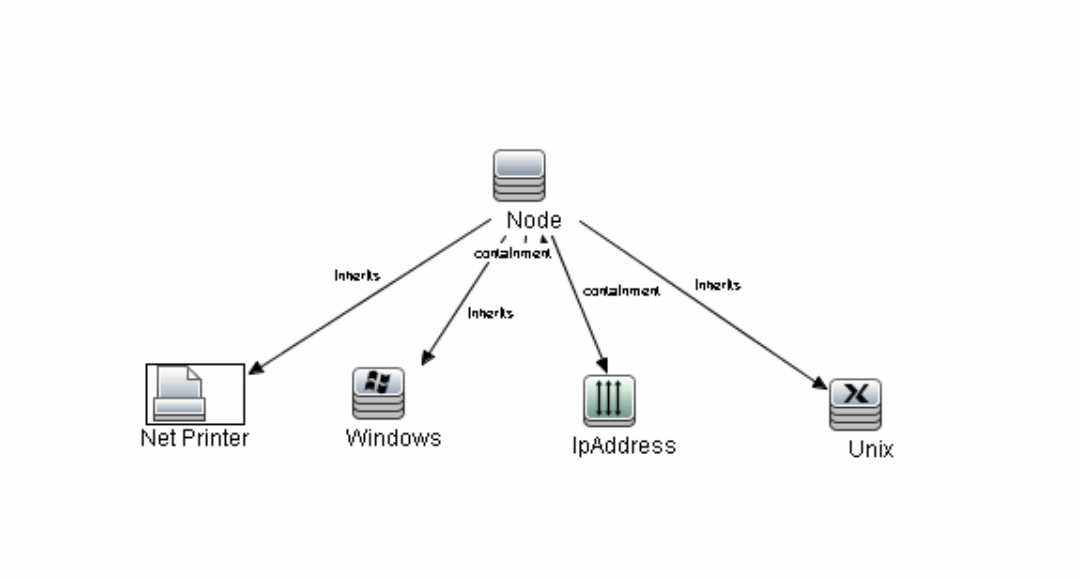
Chi...	Nome visualizzato	Nome	Tipo	Descrizione	Valore pred...	Visibile
	Create Time	create_time	date	When was ...		✔
	Created By	data_source	string			✔
	credentials_id	Reference	string	Reference ...		✔
	Deletion Candidate ...	root_deletioncandida...	integer	What is the...	20	✔
	Description	description	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	✔
	External ID	data_externalid	string	external so...		

2. **Definire l'output dell'adattatore**

   L'output dell'adattatore è un elenco dei CI individuati (**Gestione flusso di dati > Gestione adattatori > scheda 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 pagina 43.

Esempio di output dell'adattatore:

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

a. Accedere a **Gestione flusso di dati > Gestione adattatori**.

b. Nel riquadro Risorse, selezionare **Rete > Adattatori > NSLOOKUP_on_Probe**.

c. Nella scheda Definizione adattatore, individuare il riquadro CIT individuati.

d. Vengono elencati i CIT che devono far parte dell'output dell'adattatore. Aggiungere i CIT o rimuoverli dall'elenco. Per i dettagli consultare Scheda definizione adattatore nella *Guida di Gestione flusso di dati di HP Universal CMDB*.

3. ## 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.

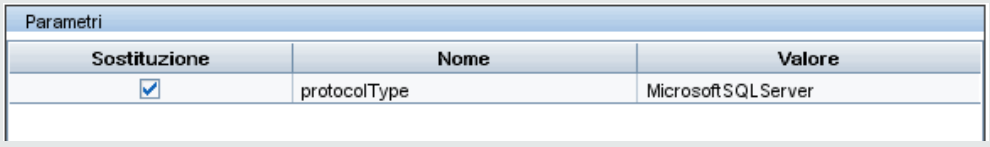
a. Accedere a **Gestione flusso di dati > Gestione adattatori**.

b. Nel riquadro Risorse, selezionare **Database di base > Adattatori > SQL_NET_Dis_Connection**.

c. Nella scheda Definizione adattatore, individuare il riquadro **Parametri adattatore**. Il parametro `protocolType` ha il valore **all**:

d. 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**.

e. 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, l'eliminazione e la modifica dei parametri, consultare Scheda definizione adattatore nella *Guida di Gestione flusso di dati di HP Universal CMDB*.

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

4. Sostituire la selezione della sonda - facoltativo

Nel server UCMDB esiste un meccanismo di invio che utilizza i CI trigger ricevuti da UCMDB e sceglie automaticamente quale sonda deve eseguire il processo per ogni CI trigger in base a una delle opzioni seguenti:

- **Per il tipo CI dell'indirizzo IP:** sceglie la sonda definita per questo IP.
- **Per il tipo CI del software in esecuzione:** utilizza gli attributi **application_ip** e **application_ip_domain** e sceglie la sonda definita per l'IP nel dominio pertinente.
- **Per altri tipi di CI:** sceglie l'IP del nodo in base al nodo correlato del CI (se esistente).

La selezione automatica della sonda viene eseguita in base al nodo correlato del CI. Una volta ottenuto il nodo correlato del CI, il meccanismo di invio sceglie uno degli IP del nodo e la sonda in base alle definizioni dell'ambito rete della sonda.

Nel seguenti casi, è necessario specificare la sonda manualmente e non ricorrere al meccanismo di invio automatico:

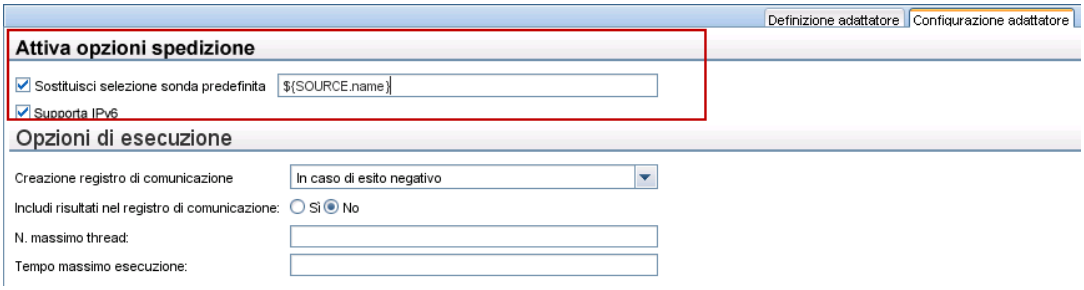
- Si conosce già quale sonda deve essere eseguita per l'adattatore e non è necessario ricorrere al meccanismo di invio automatico per selezionarla (ad esempio se il CI trigger è il gateway sonda).
- La selezione automatica della sonda potrebbe non riuscire. Questo può accadere nelle seguenti situazioni:
  - Un CI trigger non dispone di un nodo correlato (come il CIT network)
  - Il nodo di un CI trigger ha più IP, ciascuno appartenente ad una sonda diversa.

Per specificare manualmente quale sonda utilizzare con l'adattatore:

- Selezionare l'adattatore e fare clic sulla scheda **Configurazione adattatore**.
- In **Attiva opzioni spedizione**, selezionare **Sostituisci selezione sonda predefinita.**
- Nella casella, immettere la sonda in uno dei formati seguenti:

Nome sonda	Nome della sonda.
Indirizzo IP	Indirizzo IP della sonda, definito in formato IPv4 o IPv6
Dominio IP	Formato IPv4: **16.59.63.86,DefaultDomain** Formato IPv6: **2001:0:9d46:953c:34a9:1e6b:f2ff:fffe,CustomDomain**
Nome dominio	Dominio dal quale deve essere selezionata la sonda.

Ad esempio:



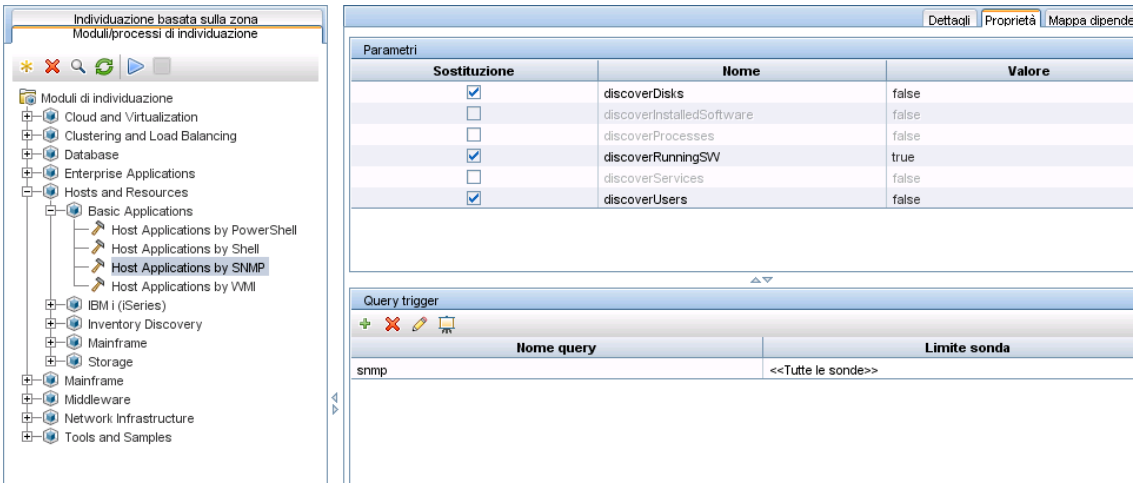
5. **Configurare un classpath per un processo remoto - facoltativo**

   Per i dettagli consultare "Esecuzione della configurazione del processo remoto" alla pagina successiva.

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, dove possono essere attivati dall'utente, come visualizzato nell'immagine seguente.



Scegliere una TQL trigger

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

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

Nota: un TQL trigger deve riferirsi agli stessi oggetti ai quali si riferisce il TQL di input. Ad esempio, se un TQL di input ricerca IP con SNMP in esecuzione, non è possibile definire un

TQL trigger (per lo stesso processo) per ricercare IP connessi a un host, poiché alcuni IP potrebbero non essere connessi a un oggetto SNMP, come richiesto dal TQL di input.

Impostare informazioni di 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 Discovery Scheduler Dialog Box nella *Guida di Gestione flusso di dati di HP Universal CMDB*.

Sostituire i parametri dell'adattatore

Quando si configura un processo, è possibile sostituire i parametri dell'adattatore. Per i dettagli consultare "Sostituire i parametri dell'adattatore" a pagina 32.

Passaggio 3: Creare un codice Jython

HP Universal CMDB utilizza script Jython per scrivere gli adattatori. 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 i seguenti siti Web (in inglese):

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

Per i dettagli consultare "Creare un codice Jython" a pagina 38.

Esecuzione della configurazione del processo remoto

È possibile eseguire un'individuazione in un processo separato rispetto a quello della Data Flow Probe.

Ad esempio, è possibile eseguirla in un processo remoto separato se utilizza librerie **.jar** di una versione diversa rispetto a quelle della sonda oppure che non sono compatibili con quelle della sonda.

È possibile inoltre eseguirla in un processo remoto separato qualora dovesse consumare potenzialmente molta memoria (recupera molti dati) e si desidera isolare la sonda da potenziali problemi di **insufficienza di memoria**.

Per configurare un processo da remoto, definire i seguenti parametri nel corrispondente file di configurazione dell'adattatore:

Parametro	Descrizione
remoteJVMArgs	Parametri JVM per il processo Java remoto.
runInSeparateProcess	Quando è impostato su **true**, l'individuazione viene eseguita in un processo separato.

remoteJVMClasspath	(Facoltativo) Consente la personalizzazione del classpath del processo remoto, sovrapponendo il classpath della sonda predefinita. Ciò risulta utile se ci sono versioni incompatibili tra i jar della sonda e quelli personalizzati necessari per l'individuazione definita dal cliente.
	
	Se il parametro **remoteJVMClasspath** non è definito o se questo campo viene lasciato vuoto, si utilizza il classpath della sonda predefinita.
	
	Se si sviluppa un nuovo processo di individuazione e si desidera garantire che la versione della libreria jar della sonda non sia in conflitto con le librerie jar del processo, è necessario utilizzare almeno il classmath minimo richiesto per eseguire l'individuazione di base. Il classpath minimo viene definito nel file **DataFlowProbe.properties** nel parametro **basic_discovery_minimal_classpath**.
	
	Esempi di personalizzazione di **remoteJVMClasspath**:
	
	• Per aggiungere o associare i jar personalizzati al classpath della sonda predefinita, personalizzare il parametro **remoteJVMClasspath** come segue:
	
	`custom1.jar;%classpath%;custom2.jar -`
	
	In questo caso, **custom1.jar** è ubicata prima del classpath della sonda predefinita e **custom2.jar** è associata al classpath della sonda.
	
	• Per utilizzare il classpath minimo, personalizzare il parametro **remoteJVMClasspath** come segue:
	
	`custom1.jar;%minimal_classpath%;custom2.jar`

Capitolo 2: Sviluppo degli adattatori Jython

Questo capitolo comprende:

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:

<directory di installazione di UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_JavaDoc\index.html

Creare un codice Jython

HP Universal CMDB utilizza script Jython per scrivere gli adattatori. 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 sviluppati per Universal Discovery devono essere compatibili con Jython versione 2.5.3.
- 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:

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 JAR 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.

Modificare gli 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_uti
ls)
```

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

```
# implement customer specific processing for 'servername' attribute of httpplugi
n
#
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 f
ind. this join can't be done with IP address:port
                # because multiple apps on a web server share the same IP:port f
or multiple websphere applications
                logger.debug('httpcontext_webapplicationserver attribute has bee
n changed from [' + servername + '] to [' + transportHN[:5] + '] to facilitate w
ebsphere enrichment')
                servername = transportHN[:5]
        return servername
```

Salvare lo script esterno nella cartella Risorse esterne. Per i dettagli consultare Resources Pane nella *Guida di Gestione 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 "Package Manager" 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:

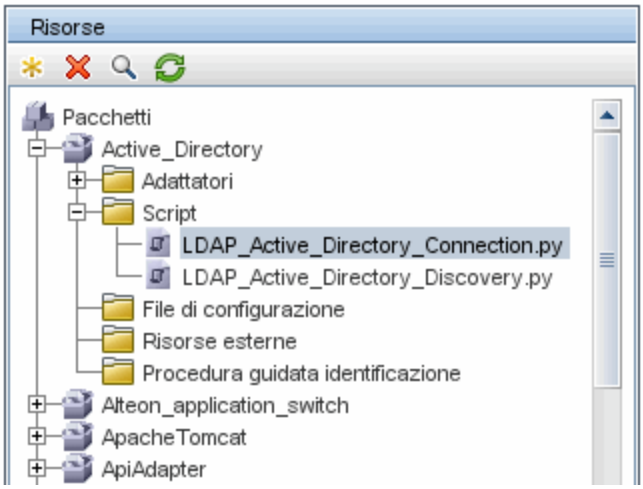
```
def DiscoveryMain(Framework):
```

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,str
ing'
    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.createClient(Framework.getTriggerCIData(BaseClien
t.CREDENTIALS_ID))
    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 adattatori > 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 in 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 in 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 in 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 "Link" nella *Guida di Gestione 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 CI Type Relationships 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 in 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" a pagina 64.

Invio di grandi quantità di dati

L'invio di grandi quantità di dati (solitamente più di 20 KB) è difficile da elaborare in UCMDB. Dati di queste dimensioni devono essere suddivisi in blocchi più piccoli prima di essere inviati a UCMDB. Per poter inserire tutti i blocchi correttamente in UCMDB, ogni blocco deve contenere le informazioni di identificazione richieste per i CI nel blocco. Questo è uno scenario comune nello sviluppo delle integrazioni Jython. Il metodo sendObjects viene utilizzato per inviare i risultati in blocchi. Se lo script Jython invia un gran numero di risultati (il valore predefinito è 20.000, ma questo valore può essere configurato nel "file DataFlowProbe.properties" utilizzando la chiave **appilog.agent.local.maxTaskResultSize**), deve suddividerli in blocchi in base alla topologia. Questa suddivisione in blocchi deve essere eseguita in base alle regole di identificazione, così che i risultati vengano inseriti correttamente in UCMDB. Se lo script Jython non suddivide i risultati in

blocchi, la sonda proverà a suddividerli; tuttavia, questo può portare a prestazioni insufficienti per un insieme di risultati di grandi dimensioni.

Nota: La suddivisione in blocchi deve essere utilizzata per gli adattatori di integrazione Jython e non per i processi di individuazione regolari. Il motivo è che i processi di individuazione generalmente individuano informazioni riguardanti un trigger specifico e non inviano grandi quantità di informazioni. Con le integrazioni Jython, vengono individuate grandi quantità di dati sul singolo trigger dell'integrazione.

È possibile utilizzare la suddivisione in blocchi anche per un piccolo numero di risultati. In questo caso, esiste una relazione tra i CI in blocchi diversi e lo sviluppatore dello script Jython ha due opzioni:

- Inviare di nuovo l'intero CI e tutte le relative informazioni di identificazione in ogni blocco che contiene un collegamento a esso.
- Utilizzare l'ID UCMDB del CI. Per procedere in questo modo, lo script Jython deve aspettare che ogni blocco venga elaborato nel server UCMDB per poter ottenere gli ID di UCMDB. Per abilitare questa modalità (detta invio sincrono dei risultati), aggiungere il tag SendJythonResultsSynchronously all'adattatore. Questo tag garantisce che quando si è finito di inviare il blocco, gli ID UCMDB dei CI nel blocco sono già stati ricevuti dalla sonda. Lo sviluppatore dell'adattatore può utilizzare gli ID di UCMDB per generare il blocco successivo. Per utilizzare gli ID di UCMDB, utilizzare l'API framework getIdMapping.

Esempio di utilizzo di getIdMapping

Nel primo blocco vengono inviati nodi. Nel secondo blocco vengono inviati processi. Il contenitore radice del processo è un nodo. Invece di inviare l'intero objectStateHolder del nodo nell'attributo root_container del processo, è possibile ottenere l'ID UCMDB del nodo utilizzando l'API getIdMapping e usare solamente l'ID nodo nell'attributo root_container del processo per ridurre le dimensioni del blocco.

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 pagina 38.

Il corretto utilizzo dell'istanza Framework è passarlo come argomento ad ogni metodo che lo utilizza.

Esempio:

```
def DiscoveryMain(Framework):
         OSHVResult = helperMethod (Framework)
            return OSHVResult
```

```
def helperMethod (Framework):
    ....
     probe_name    = Framework.getDestinationAttribute('probe_name')
     ...
     return result
```

Questa sezione descrive gli utilizzi più importanti del Framework:

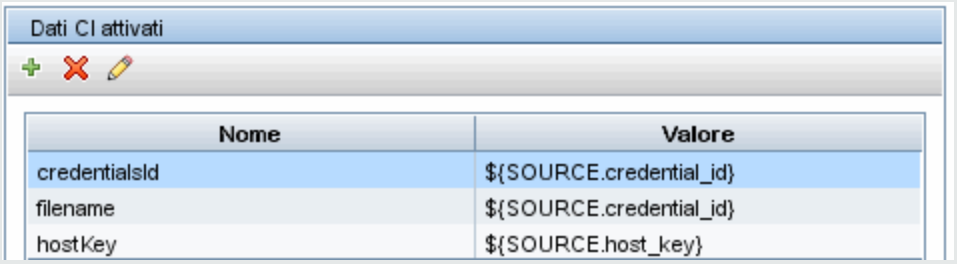
- "Framework.getTriggerCIData(String attributeName)" nel seguito
- "Framework.createClient(credentialsId, props)" nel seguito
- "Framework.getParameter (String parameterName)" a pagina 48
- "Framework.reportError(String message) e Framework.reportWarning(String message)" a pagina 48

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 sono definite nella classe ClientsConsts. Per i dettagli sulle credenziali e sui protocolli supportati consultare *HP UCMDB Discovery and Integrations Content Guide*.

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.

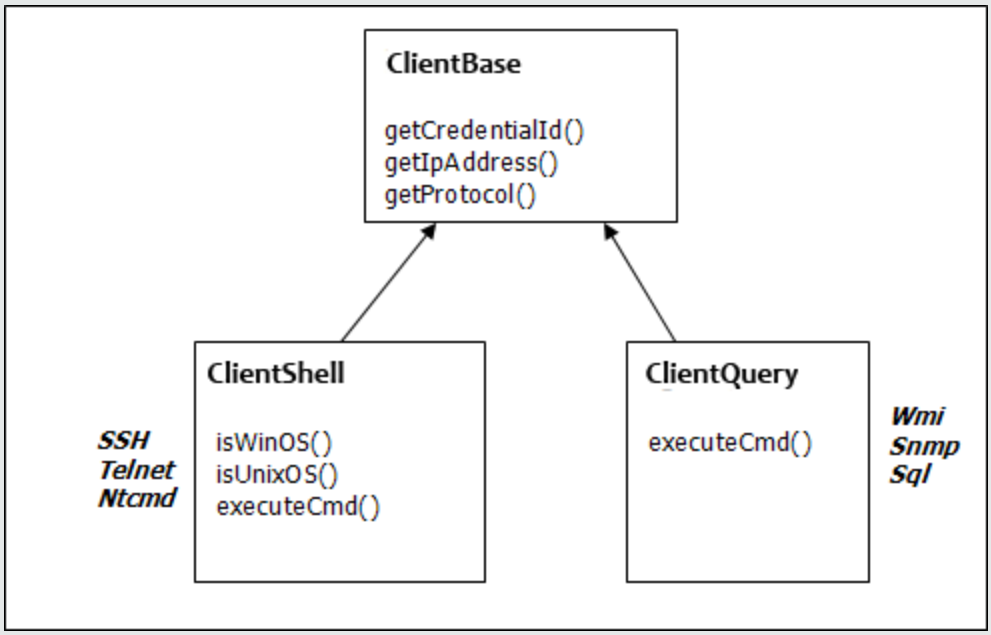
Esempio di creazione di un client WMI e di esecuzione di una query WMI:

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

```
wmiClient = Framework.createClient(credential)
resultSet = wmiClient. executeQuery("SELECT TotalPhysicalMemory
                                                    FROM    Win32_Logical
MemoryConfiguration")
```

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:

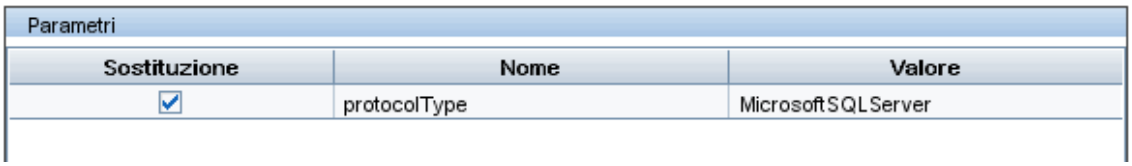


Per dettagli sui client e le API supportate, consultare BaseClient, ShellClient e QueryClient nel DFM Framework. Questi file si trovano nella cartella seguente:

<directory radice di UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_Schema\webframe.html

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.createClient(Framework.getTriggerCIData(BaseClient.CR
EDENTIALS_ID))

except:
    strException = str(sys.exc_info()[1]).strip()
    Framework. reportError ('Connection failed: %s' % strException)
```

È possibile utilizzare una delle API—`Framework.reportError(String message)`, `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 pagina 66.

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 è l'ID delle credenziali. Lo script di connessione ottiene l'accesso ai possibili insiemi di credenziali e li prova uno alla volta utilizzando il

metodo `Framework.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) a 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).

Nota: L'accesso ai dati sensibili (password, chiavi private e così via) è bloccato per i seguenti tipi di protocollo:

sshprotocol, ntadminprotocol, as400protocol, vmwareprotocol, wmiprotocol, vcloudprotocol, sapjmxprotocol, webspherepreprotocol, siebelgtwyprotocol, sapprotocol, ldapprotocol, udaprotocol, ntcmdprotocol, snmpprotocol, jbossprotocol, telnetprotocol, powershellprotocol, sqlprotocol, weblogicprotocol

Questi tipi di protocollo devono essere utilizzati con client dedicati.

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 la voce **Il paradigma di connessione a due fasi** in "Separazione degli adattatori" a pagina 24.

Esempio

```
import logger
import netutils
import sys
import errorcodes
import errorobject

# Java imports
from java.util import Properties
from com.hp.ucmdb.discovery.common import CollectorsConstants
from appilog.common.system.types.vectors import ObjectStateHolderVector
from com.hp.ucmdb.discovery.library.clients import ClientsConsts
from com.hp.ucmdb.discovery.library.scope import DomainScopeManager

TRUE = 1
FALSE = 0

def mainFunction(Framework, isClient, ip_address = None):
       _vector = ObjectStateHolderVector()
       errStr = ''
       ip_domain  = Framework.getDestinationAttribute('ip_domain')
       # Get the Trigger CI data values
       ip_address = Framework.getDestinationAttribute('ip_address')

          if (ip_domain == None):
             ip_domain = DomainScopeManager.getDomainByIp(ip_address, None)
```

```
protocols = netutils.getAvailableProtocols(Framework, ClientsConst
s.SNMP_PROTOCOL_NAME, ip_address, ip_domain)
if len(protocols) == 0:
    errStr = 'No credentials defined for the triggered ip'
    logger.debug(errStr)
    errObj = errorobject.createError(errorcodes.NO_CREDENTIALS_FOR_
TRIGGERED_IP, [ClientsConsts.SNMP_PROTOCOL_NAME], errStr)
    return (_vector, errObj)

connected = 0
# Go over all the protocol credentials
for protocol in protocols:
    client = None
    try:
        try:
            logger.debug('try to get snmp agent for: %s:%s' % (ip_
address, ip_domain))
            if (isClient == TRUE):
                properties = Properties()
                properties.setProperty(CollectorsConstants.DESTINA
TION_DATA_IP_ADDRESS, ip_address)
                properties.setProperty(CollectorsConstants.DESTINA
TION_DATA_IP_DOMAIN, ip_domain)
                client = Framework.createClient(protocol, properti
es)
            else:
                properties = Properties()
                properties.setProperty(CollectorsConstants.DESTINA
TION_DATA_IP_ADDRESS, ip_address)
                client = Framework.createClient(protocol, properti
es)
            logger.debug('Running test connection queries')
            testConnection(client)
            Framework.saveState(protocol)
            logger.debug('got snmp agent for: %s:%s' % (ip_address,
ip_domain))
            isMultiOid = client.supportMultiOid()
            logger.debug('snmp server isMultiOid state=%s' %isMult
iOid)

         client.close()
         client = None

        except:
            if client != None:
                client.close()
                client = None
            logger.debugException('Unexpected SNMP_AGENT
```

```
          Exception:')
                      lastExceptionStr = str(sys.exc_info()[1]).strip()
              finally:
                  if client != None:
                      client.close()
                      client = None

        return (_vector, error)
```

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 buona norma stamparne la traccia dello stack nel registro e visualizzare un messaggio adeguato nell'interfaccia utente.

Nota: È molto importante importare la classe delle eccezioni di base di Java come mostrato nel seguente esempio, a causa della presenza della classe delle eccezioni di base in Python con lo stesso nome.

```
from java.lang import Exception as JException
try:
       client = Framework.createClient(Framework.getTriggerCIData(BaseClient.CREDENTIA
LS_ID))
except JException, ex:
       # process java exceptions only
       Framework.reportError('Connection failed')
       logger.debugException(str(ex))
        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.

Risoluzione dei problemi di migrazione da Jython versione 2.1 alla 2.5.3

Universal Discovery utilizza ora Jython versione 2.5.3. Tutti gli script predefiniti sono stati migrati correttamente. Se si sono sviluppati script Jython personalizzati per l'utilizzo da parte di Discovery prima di questo upgrade, è possibile riscontrare i problemi seguenti e dover apportare le correzioni indicate.

Nota: È necessario essere uno sviluppatore Jython esperto per apportare queste modifiche.

Formattazione stringhe

- **Messaggio di errore:** `TypeError: int argument required`
- **Causa possibile:** utilizzo della formattazione stringa a intero decimale da una variabile stringa contenente dati interi.
- **Codice Jython 2.1 problematico:**

```
variable = "43"
print "%d" % variable
```

- **Codice Jython 2.5.3 corretto:**

```
variable = "43"
print "%s" % variable
```

oppure

```
variable = "43"
print "%d" % int(variable)
```

Controllo tipo stringa

Il codice sottostante potrebbe non funzionare correttamente se l'input contiene stringhe unicode:

- **Codice Jython 2.1 problematico:** `isinstance(unicodeStringVariable,'')`
- **Codice Jython 2.5.3 corretto:** `isinstance(unicodeStringVariable,basestring)`

  Il confronto deve essere fatto con `basestring` per verificare se un oggetto è un'istanza di `str` o unicode.

Carattere non ASCII nel file

- **Messaggio di errore:**
  `SyntaxError: Non-ASCII character in file 'x', , but no encoding declared; see http://www.python.org/peps/pep-0263.html for details`
- **Codice Jython 2.5.3 corretto:** (da aggiungere nella prima riga del file)

```
# coding: utf-8
```

Importazione di sottopacchetti

- **Messaggio di errore:**
  `AttributeError: 'module' object has no attribute 'sub_package_name'`
- **Causa possibile:** viene importato un sottopacchetto senza specificare esplicitamente il nome del sottopacchetto nell'istruzione di importazione.

- **Codice Jython 2.1 problematico:**

  ```
  import a
  print dir(a.b)
  ```

  Il sottopacchetto non è importato esplicitamente.

- **Codice Jython 2.5.3 corretto:**

  ```
  import a.b
  ```

  oppure

  ```
  from a import b
  ```

Modifiche all'iteratore

A partire da Jython 2.2, viene utilizzato il metodo **__iter__** per eseguire un ciclo su una raccolta nell'ambito di un blocco **for-in**. L'iteratore deve implementare il metodo **next**, restituendo un elemento appropriato o l'errore **StopIteration** se è stata raggiunta la fine della raccolta. Se il metodo **__iter__** non è implementato, viene utilizzato invece il metodo **getitem**.

Generazione di eccezioni

- **Il metodo di Jython 2.1 per generare le eccezioni è obsoleto:**
  ```
  raise Exception, 'Impossibile leggere il contenuto del file'
  ```
- **Metodo di Jython 2.5.3 raccomandato per generare le eccezioni:**
  ```
  raise Exception('Impossibile leggere il contenuto del file')
  ```

Supporto della 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.

In questa sezione vengono trattati gli argomenti seguenti:

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)" nel seguito
- "Dichiarare e registrare l'oggetto Language" nel seguito

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 pagina 57.

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 adattatori. Per i dettagli consultare Adapter Management Window nella *Guida di Gestione flusso di dati di HP Universal CMDB*.

   a. Dichiarare la lingua come segue:

```
LANG_RUSSIAN = Language(LOCALE_RUSSIAN, 'rus', ('Cp866', 'Cp1251'), (104
9,), 866)
```

Per i dettagli sulla lingua della classe consultare "Riferimento API" a pagina 58. Per i dettagli sull'oggetto Class Locale consultare, 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" alla pagina precedente.

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 "HP Universal CMDB Panoramica API servizio Web" a pagina 269.

Questo codice assume solitamente la forma seguente:

```
client = Framework.createClient(protocol, properties)
shellUtils = shellutils.ShellUtils(client)
languageBundle = shellutils.getLanguageBundle ('langNetwork', shellUtils.osLangu
age, Framework)
strWindowsIPAddress = languageBundle.getString('windows_ipconfig_str_ip_addres
s')
ipconfigOutput = shellUtils.executeCommandAndDecode('ipconfig /all', strWindowsI
PAddress)
#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.osL
   anguage, Framework)
   ```

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

```
strWindowsIPAddress = languageBundle.getString('windows_ipconfig_str_ip_addr
ess')
```

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

   ```
   ipconfigOutput = shellUtils.executeCommandAndDecode('ipconfig /all', strWind
   owsIPAddress)
   ```

   Lo ShellUtils object 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 sul recupero di una parola chiave dal pacchetto delle risorse in "Definire un nuovo processo da utilizzare con i dati localizzati" alla pagina precedente.

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:

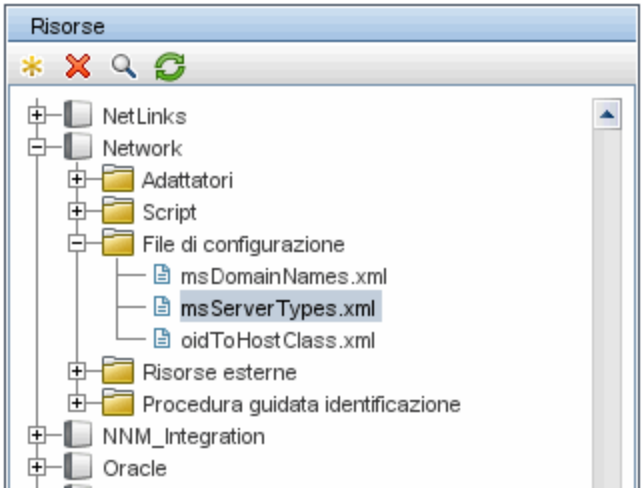
1. Richiamare il metodo **executeCommandAndDecode** una volta.
2. Ottenere il nome del set di caratteri utilizzato più di recente tramite il metodo **getCharsetName**.
3. Far utilizzare a **shellUtils** questo set di caratteri come impostazione predefinita richiamando il metodo **useCharset** sull'oggetto **ShellUtils**.
4. Richiamare il metodo **execCmd** di **ShellUtils** una o più volte. L'output viene restituito con il set di caratteri specificato nel passaggio precedente. 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 Configuration File Pane nella *Guida di Gestione 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 ogni pacchetto di risorse contiene dati per una sola 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`, for example, `langNetwork_spa.properties`. (Il pacchetto di risorse predefinito ha il seguente formato: `<base_name>.properties`, for example, `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 gli argomenti seguenti:

- "Classe Language" alla pagina successiva
- "Metodo executeCommandAndDecode" alla pagina successiva

- "Metodo getCharsetName" alla pagina successiva
- "Metodo useCharset" alla pagina successiva
- "Metodo getLanguageBundle" alla pagina successiva
- "Campo osLanguage" alla pagina successiva

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.

Nome	Descrizione
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" in precedenza.

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	L'oggetto lingua. `ShellUtils.osLanguage` deve essere passato in questo argomento.
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.

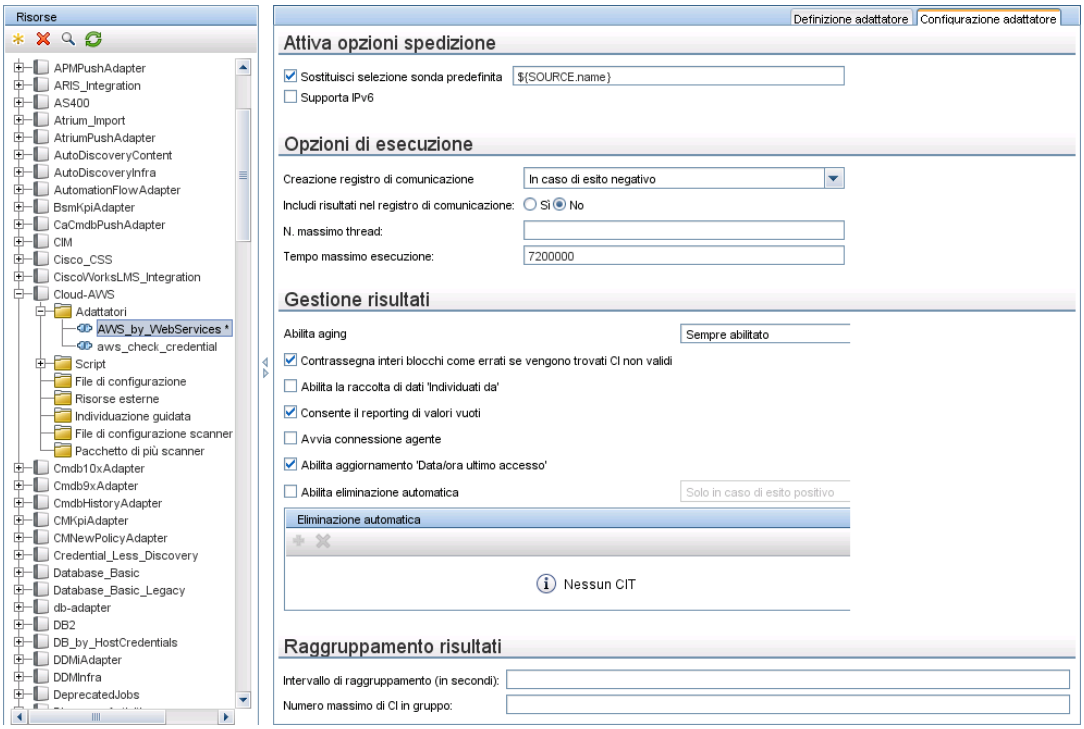
Registrare il codice GFD

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:

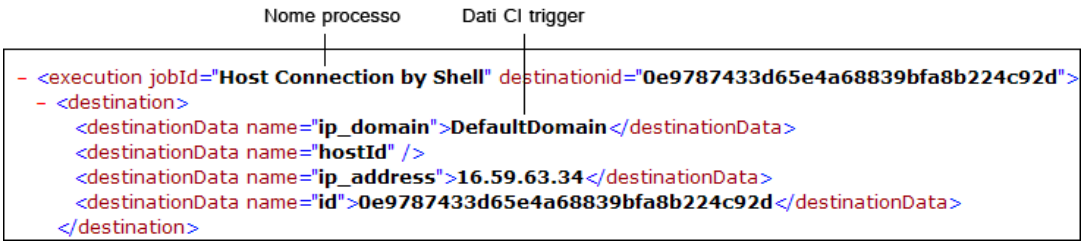
1. Accedere a **Gestione flusso di dati > Universal Discovery**. Fare clic con il pulsante destro del mouse sul processo di cui registrare l'esecuzione e selezionare **Vai ad adattatore** per aprire l'applicazione Gestione adattatori.

2. Individuare il riquadro **Opzioni di esecuzione** nella scheda **Configurazione adattatore**, come indicato di seguito:

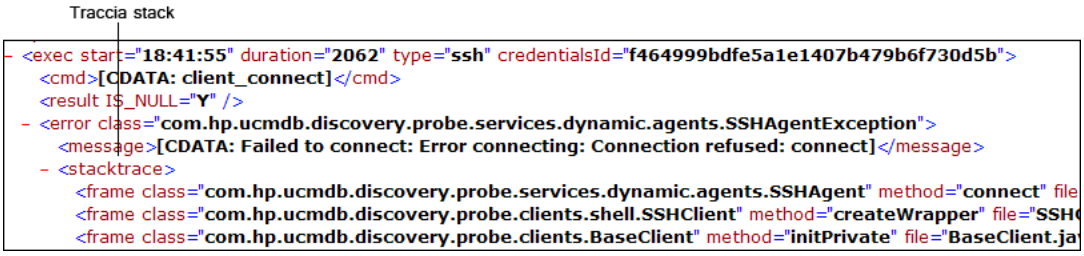


3. Cambiare la casella **Creazione registro di comunicazione** in **Sempre**. Per i dettagli sull'impostazione delle opzioni di registrazione consultare "Execution Options Pane" nella *Guida di Gestione flusso di dati di HP Universal CMDB*.

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



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



Librerie e utilità Jython

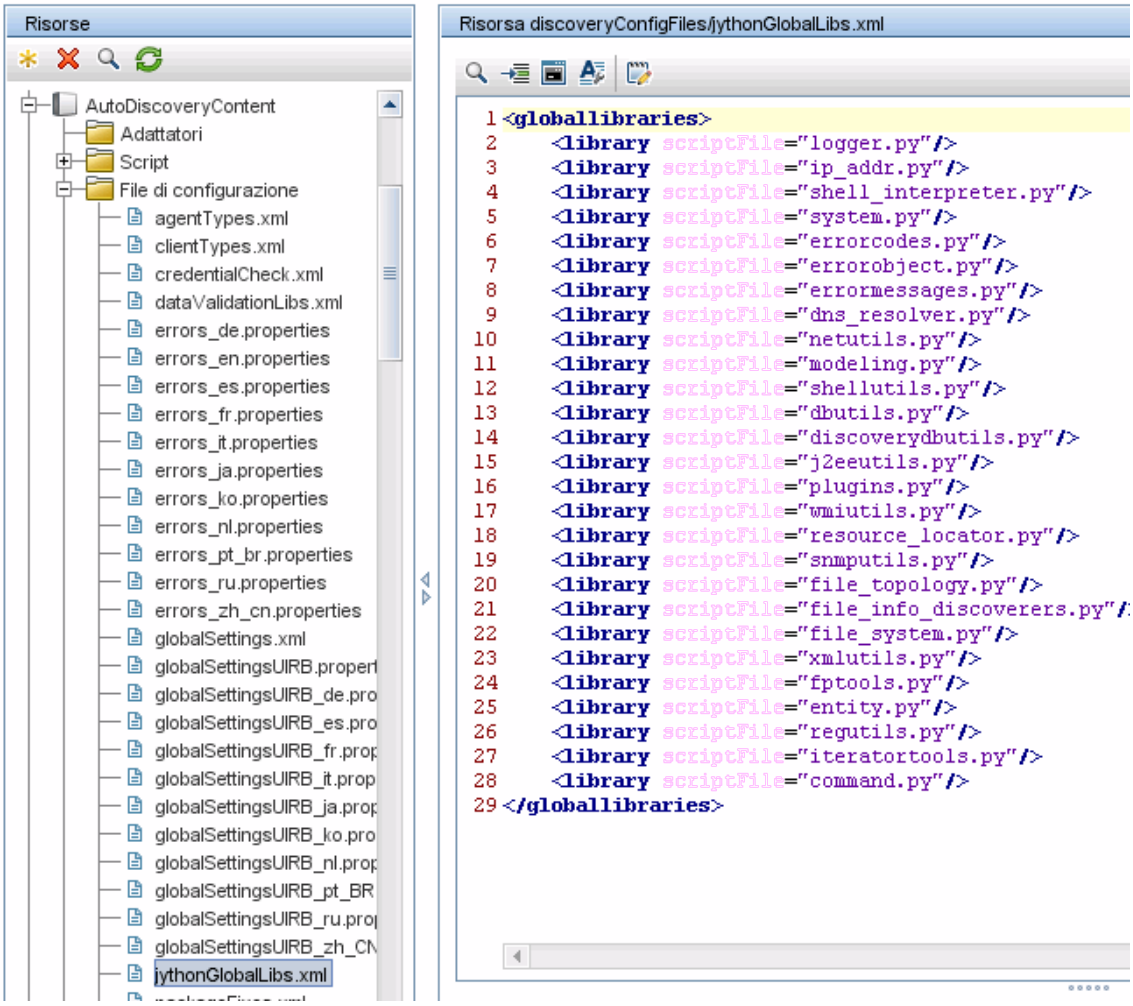
Negli adattatori sono largamente utilizzati diversi script di utilità. Questi script fanno parte del pacchetto `AutoDiscovery` e si trovano in: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts** con gli altri script scaricati nella sonda.

> **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" nel seguito
- "modeling.py" alla pagina successiva
- "netutils.py" alla pagina successiva
- "shellutils.py" a pagina 65

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 dell'errore" a pagina 69.

```
#########################################################################
################         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/pro
beMgr-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 = Framework.createClient(Framework.getTriggerCIData(BaseClient.CREDENT
IALS_ID), Props)
clientShUtils = shellutils.ShellUtils(ttyClient)
if (clientShUtils.isWinOs()):
        logger.debug ('discovering Windows..')
```

Capitolo 3: Messaggi di errore

Questo capitolo comprende:

Panoramica dei messaggi di errore

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. Questi errori vengono visualizzati nella finestra di Universal Discovery ogni volta che il flusso di individuazione regolare non ha esito positivo. 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" nel seguito.

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 Avanzamento e risultati dell'individuazione nella *Guida di Gestione flusso di dati di HP Universal CMDB*.

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

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 modello contenente zero o più argomenti. Si presuppone una corrispondenza tra il numero di argomenti nel modello 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 ai 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 per le impostazioni internazionali

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

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 dell'errore

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.

Avvisi

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.

Capitolo 4: Sviluppo degli adattatori generici del database

Questo capitolo comprende:

Panoramica di Adattatore generico del database

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.

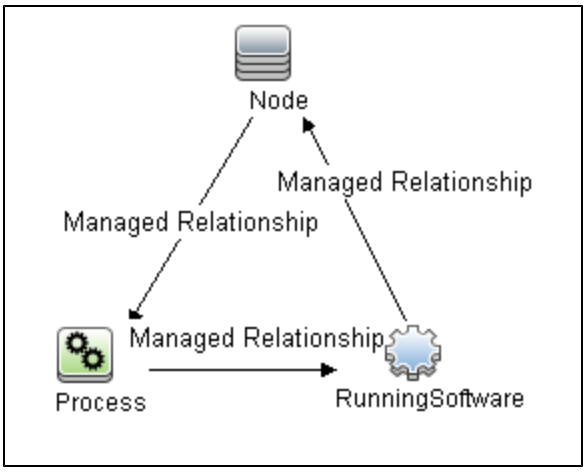
Query TQL per l'adattatore generico del database

Per i processi di popolamento, ciascun layout richiesto di un CI deve essere verificato in una finestra di dialogo Impostazioni layout nello Studio di modellazione. Per i dettagli consultare Query Node/Relationship Properties Dialog Box nella *Guida alla modellazione di HP Universal CMDB*. È importante notare che un CI può richiedere l'identificazione di un attributo e che senza tali attributi il CI non potrà essere aggiunto aUCMDB.

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

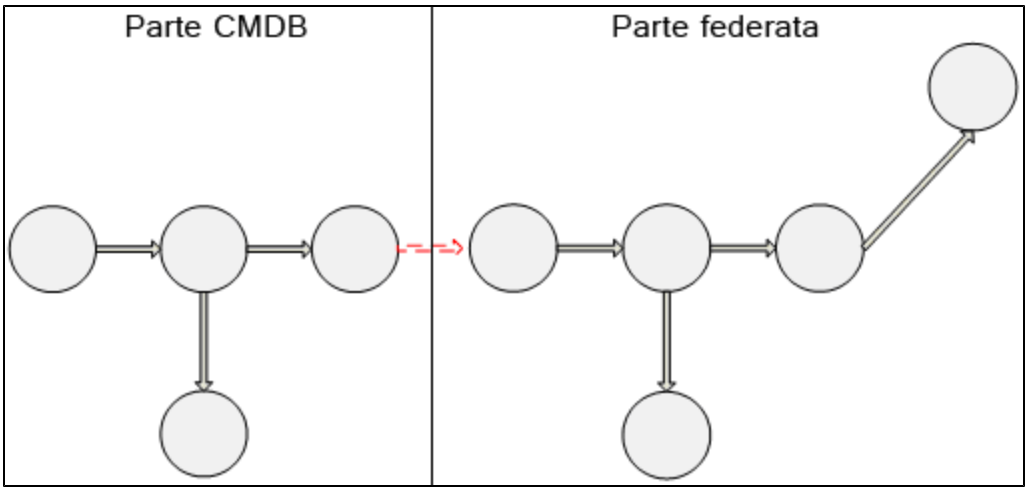
- I sottografici non sono supportati
- Le relazioni compound non sono supportate
- I cicli o parti di essi non sono supportati

  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 di riconciliazione.

Mapping. Ciascun attributo in CMDB è mappato a una colonna nell'origine di dati.

Anche se il mapping viene eseguito direttamente, vengono supportate anche le funzioni di trasformazione sul mapping 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 in CMDB in un formato e nel database federato in un altro formato).

Nota:

- Per collegare CMDB e la sorgente esterna del database, è necessario che esista un'associazione adeguata nel database. Per i dettagli consultare "Prerequisiti" a pagina 76.
- È inoltre supportata la riconciliazione con l'ID di CMDB
- È inoltre supportata la riconciliazione con l'ID globale.

Hibernate come provider JPA

Hibernate è uno strumento di mapping di tipo object-relational (OR) che consente il mapping 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 pagina 147.

In un mapping elementare, ciascuna classe Java viene mappata a una sola tabella. Un mapping più avanzato consente l'ereditarietà (come si può verificare nel database di CMDB).

Altre funzioni supportate comprendono il mapping 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" alla pagina successiva più avanti.

Per i nostri fini non è necessario creare classi Java. Il mapping è definito dai CIT del modello di classe di CMDB alle tabelle del database.

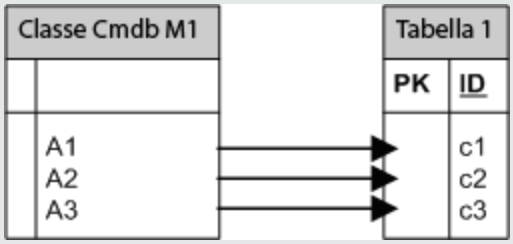
La sezione è suddivisa negli argomenti seguenti:

Esempi di Object-Relational Mapping

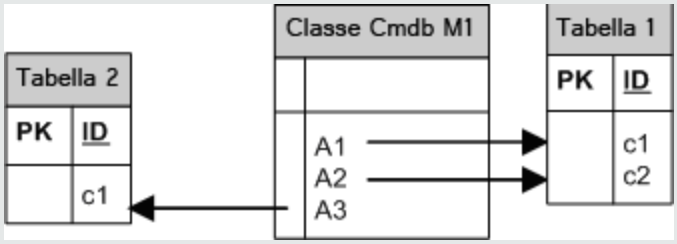
L'esempio seguenti descrive l'object-relational mapping:

Esempio di una classe di 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`.

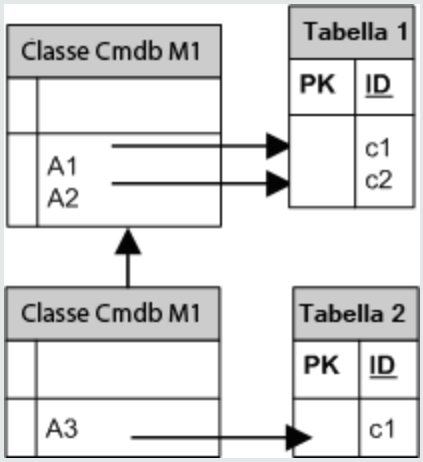


Esempio di una classe di CMDB mappata a due tabelle di database:



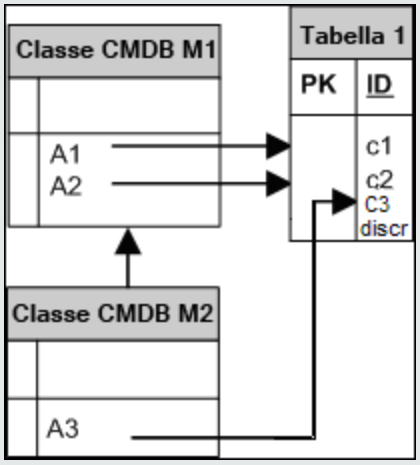
Esempio di ereditarietà:

Questo caso viene utilizzato in 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.



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 mapping (per il caso many-to-many).

Usabilità

Poiché lo schema JPA è molto ampio, viene fornito un file XML semplificato per facilitare la definizione delle associazioni.

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 di CMDB non federata. Inoltre, è possibile un solo tipo di relazione tra la classe federata e la classe non federata.

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 DB generico nell'API UCMDB. In particolare, l'esempio di adattatore DDMi contiene un file **orm.xml** complesso e le implementazioni delle interfacce di alcuni plug-in.

Questo compito include i passaggi seguenti:

- "Prerequisiti" nel seguito
- "Creare un tipo di CI" a pagina 78
- "Creare una relazione" a pagina 78

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 multinodi). 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 di CMDB corrispondenti agli ID di CMDB dei nodi in CMDB. Per i dettagli sulla riconciliazione consultare "Riconciliazione" a pagina 73.

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 mapping. 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 mapping 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 pagina 73.

  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 che rappresenta i dati nel RDBMS (origine di dati esterna).

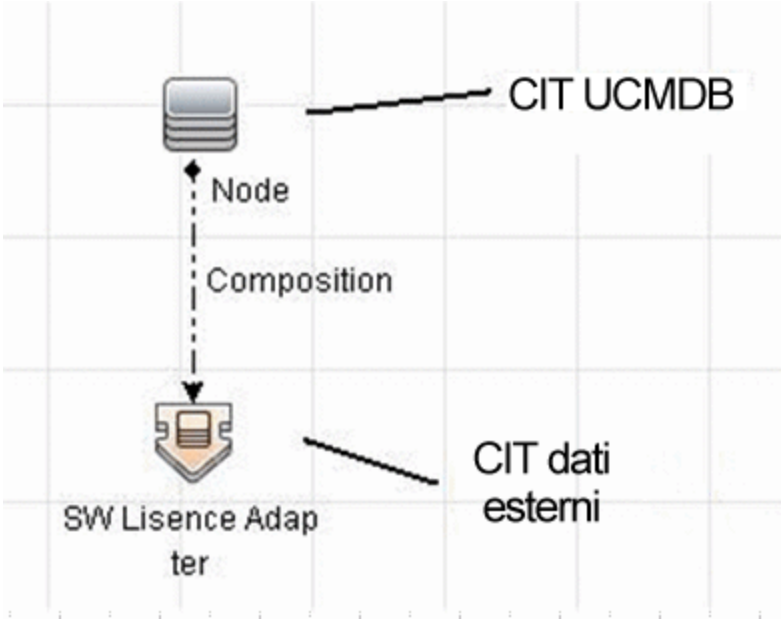
   a. In UCMDB, accedere a Gestione tipi CI e creare un nuovo tipo CI. Per i dettagli consultare Create a CI Type nella *Guida alla modellazione di HP Universal CMDB*.

   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 di CMDB.

3. **Creare una relazione**

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

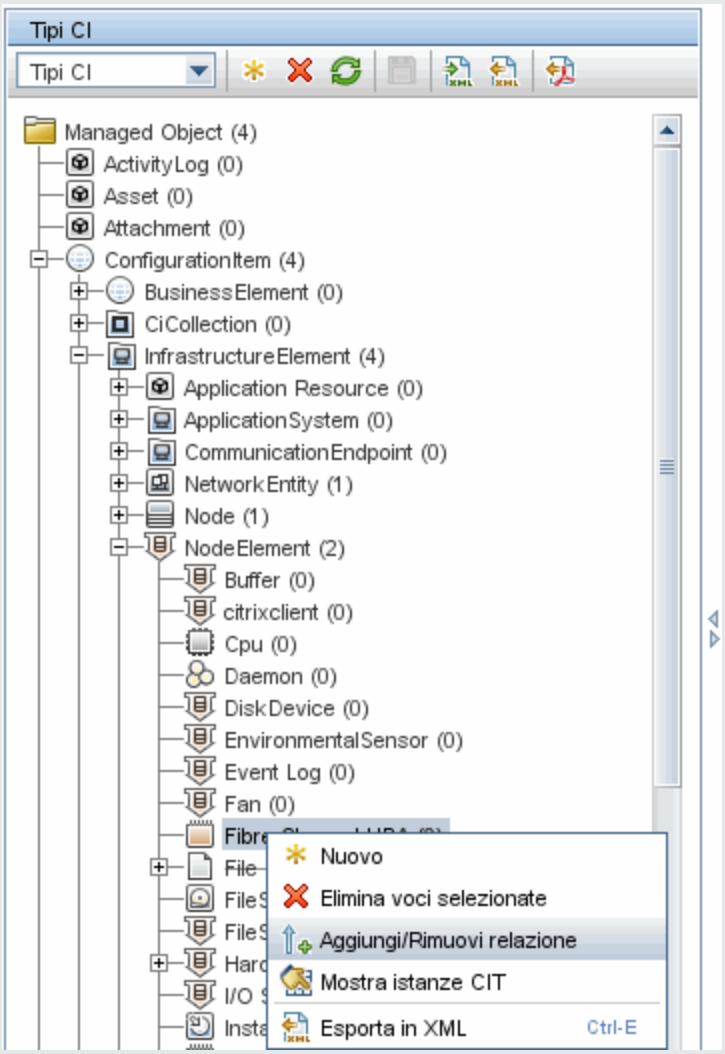
   Aggiungere le relazioni adeguate, valide al nuovo CIT. Per i dettagli consultare Add/Remove Relationship Dialog Box nella *Guida alla modellazione di HP Universal CMDB*.



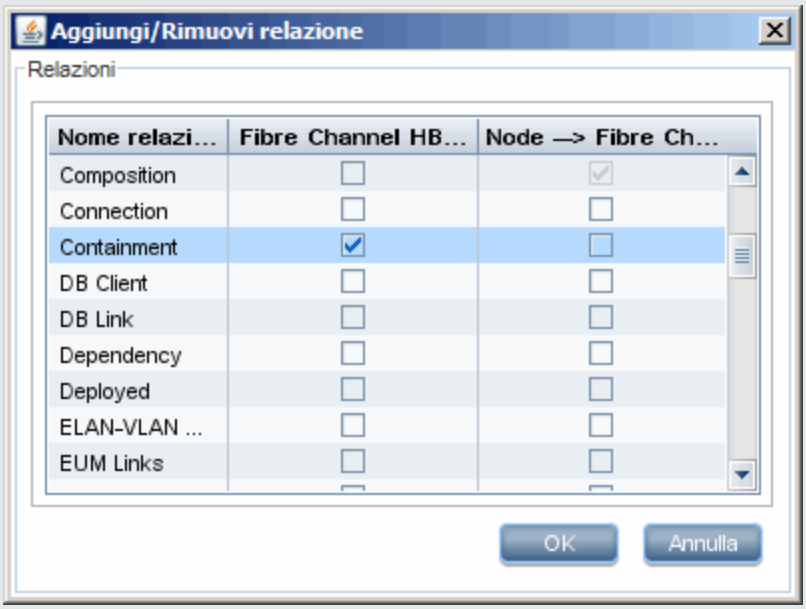
Nota: a questo punto non è ancora possibile visualizzare i dati federati o popolare i dati esterni poiché non è stato ancora definito il metodo per portare dentro i dati.

Esempio di creazione di una relazione Containment:

a. In Gestione tipo CI selezionare i due CIT:



b. Creare una relazione **Containment** tra i due CIT:



Preparare il pacchetto dell'adattatore

In questo passaggio si individua e si configura il pacchetto dell'adattatore DB generico.

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:noNamespaceSchem
aLocation="../../Patterns.xsd" description="Discovery Pattern Descript
ion"
         schemaVersion="9.0" xmlns:xsi="http://www.w3.org/2001/XMLSche
ma-instance" displayName="UCMDB API Population">
```

Se l'adattatore supporta il popolamento, 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 Punto di integrazione in HP Universal CMDB.

Quando si crea un adattatore DB generico non è necessario modificare il tag **changes-source** nel tag **support-replicatioin-data**. Se il plug-in **FcmdbPluginForSyncGetChangesTopology** è stato implementato, verrà restituita la topologia modificata dall'ultima esecuzione. Se il plug-in non viene modificato, verrà restituita la topologia completa ed eseguita l'eliminazione in base ai CI restituiti.

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

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

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

  con:

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

  Per invertire il nuovo motore di mapping, 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.

- La connessione al database può essere descritta con un nome utente (schema), una password, un tipo di database, il nome del computer host del database e il nome del database o SID.

  Per questo tipo d connessione, i parametri seguono i seguenti elementi nella sezione **parameter** del file XML dell'adattatore:

```
<parameters>
    <!--The description attribute may be written in simple text or HTM
L.-->
    <!--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.-->
    <!--Display name and description may be overwritten by I18N values-
->
        <parameter name="host" display-name="Hostname/IP" type="string"
description="The host name or IP address of the remote machine" mandat
ory="false" order-index="10" />
        <parameter name="port" display-name="Port" type="integer" desc
ription="The remote machine's connection port" mandatory="false" order-
index="11" />
        <parameter name="dbtype" display-name="DB Type" type="string"
description="The type of database" valid-values="Oracle;SQLServer;MySQ
L;BO" mandatory="false" order-index="13">Oracle</parameter>
        <parameter name="dbname" display-name="DB Name/SID" type="stri
ng" description="The name of the database or its SID (in case of Oracl
e)"  mandatory="false" order-index="13" />
        <parameter name="credentialsId" display-name="Credentials ID"
type="integer" description="The credentials to be used" mandatory="tru
e" order-index="12" />
</parameters>
```

Nota: Questa è la configurazione predefinita. Tuttavia, il file **db_adapter.xml** già contiene questa definizione.

In alcuni casi la connessione al database non può essere configurata in questo modo. Ad esempio, la connessione a Oracle RAC o utilizzando un driver database diverso da quello fornito con CMDB.

Per questi casi, è possibile descrivere la connessione con un nome utente (schema), password e una stringa URL per la connessione.

Per definire questa connessione, modificare la sezione dei parametri XML dell'adattatore come segue:

```
<parameters>
    <!--The description attribute may be written in simple text or HTML.-->
    <!--The host attribute is treated as a special case by CMDBRTSM-->
    <!--and will automatically select the probe name (if possible)-->
    <!--according to this attribute's value.-->
    <!--Display name and description may be overwritten by I18N values-->
        <parameter name="url" display-name="Connection String"
```

```
type="string" description="The connection string to connect to the databas
e" mandatory="true" order-index="10" />
        <parameter name="credentialsId" display-name="Credentials ID" type=
"integer" description="The credentials to be used" mandatory="true" order-i
ndex="12" />
</parameters>
```

Di seguito l'esempio di un'URL che si connette a un Oracle RAC con il driver Data Direct predefinito:
jdbc:mercury:oracle://labm3amdb17:1521;ServiceName=RACQA;AlternateServers=(labm3amdb18:1521);LoadBalancing=true.

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

   Questa cartella contiene .la configurazione dell'adattatore, ad esempio il nome dell'adattatore, le query e le classi in CMDB, e i campi del RDBMS supportati dall'adattatore.

5. Configurare l'adattatore come necessario. Per i dettagli consultare "Configurare l'adattatore - Metodo minimale" nel seguito.

6. Creare un file `*.zip` con lo stesso nome dato all'attributo **adapter id** come descritto nel passaggio"Modificare il file XML dell'adattatore:" a pagina 80.

   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**.

Configurare l'adattatore - Metodo minimale

Il metodo semplificato (minimale) è un metodo per la creazione del file di mapping **simplifiedConfiguration.xml** utilizzato dall'adattatore. Questo metodo consente un popolamento o federazione di base di un singolo CIT.

Le istruzioni specificate in questa sezione descrivono un metodo per mappare il modello di classe per alcuni tipi CI in CMDB a un RDBMS.

Tutti i file di configurazione indicati in questa sezione si trovano nel pacchetto **db-adapter.zip** nella cartella **C:\hp\UCMDB\UCMDBServer\content\adapters** estratta in "Preparare il pacchetto dell'adattatore" a pagina 80.

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.

È possibile utilizzare il metodo minimale quando è necessario:

- Federare/popolare un solo nodo come attributo del nodo.
- Dimostrare le funzioni dell'adattatore generico del database.

Questo metodo:

- Supporta solo la federazione o il popolamento a un nodo
- supporta soltanto le relazioni virtuali many-to-one

Configurazione del file adapter.conf

Per cambiare le impostazioni nel file `adapter.conf` in modo che l'adattatore utilizzi il metodo di configurazione semplificato:

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**.

Esempio: Popolare un nodo e un indirizzo IP con il metodo semplificato

Questo esempio mostra come popolare un **Nodo** correlato a un collegamento containment all'**Indirizzo IP** in UCMDB. RDBMS ha una tabella denominata **simpleNode** contenente i dati sul nome, sul nodo e sull'indirizzo IP del computer.

Il contenuto della tabella **simpleNode** è indicato di seguito:

	host_id	host_name	note	ip_address
	1	Comp1	Test Computer 1	12.33.211.52
	2	Comp2	Test Computer 2	12.33.211.53
	3	Comp3	Test Computer 3	12.33.211.54
	4	Comp4	Test Computer 4	12.33.211.55

Il popolamento viene eseguito in tre fasi, come segue:

1. "Creare simplifiedConfiguration.xml" nel seguito
2. "Creare la TQL" a pagina 86
3. "Creare un punto di integrazione" a pagina 87

Creare simplifiedConfiguration.xml

Creare **simplifiedConfiguration.xml** come segue:

1. Creare un'entità **cmdb-class** come segue:

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

   Il tipo CI è **node** e il nome della tabella RDBMS è **simpleNode**.

2. Impostare la chiave primaria della tabella come segue:

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

   La chiave primaria equivale all'entità nel file **orm.xml**.

3. Impostare la regola **reconciliation-by-two-nodes** come segue:

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

   Questo tag definisce la relazione tra il **Nodo** e i tipi CI **IpAddress**. Il tipo di relazione è collegamento Containment. La riconciliazione viene eseguita da due tipi CI connessi. Il mapping degli attributi del nodo connesso (in questo caso IpAddress) è definito nell'attributo **connected-node**.

4. Aggiungere la condizione **or** tra gli attributi di riconciliazione come segue:

   ```
   <or is-ordered="true">
   ```

   Questo tag definisce una relazione OR tra gli attributi di riconciliazione, ciò significa che il primo attributo di riconciliazione **true** imposta l'intera regola di riconciliazione in **true**.

5. Aggiungere i seguenti attributi:

   ```
   <attribute cmdb-attribute-name="name" column-name="host_name" ignore-case="t
   rue"/>
   ```

   Questo tag imposta un mapping tra **node.name** in UCMDB e la colonna **host_name** nella tabella **simpleNode**.

   Procedere allo stesso modo con l'attributo **data_note**:

   ```
   <attribute cmdb-attribute-name="data_note" column-name="note" ignore-case="t
       rue"/>
   ```

   Aggiungere l'attributo nodo connesso:

   ```
   <connected-node-attribute cmdb-attribute-name="name" column-name="ip_addres
   s"/>
   ```

   Questo tag imposta un mapping tra **ip_address.name** e la colonna **ip_address** nella tabella **simpleNode**.

6. Chiudere il tag aperto secondo l'ordine:

   ```
   </or>
   ```

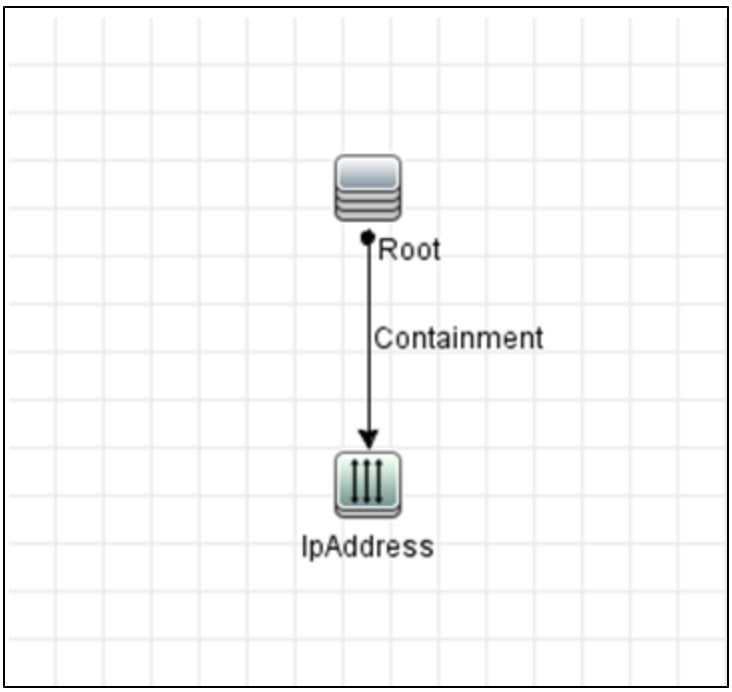
```
    </reconciliation-by-two-nodes>
    </cmdb-class>
```

Il contenuto del file simplifiedConfigurazion.xml ora appare come segue:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-db-adapter-config xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfig
uration.xsd">
   <cmdb-class cmdb-class-name="node" default-table-name="simpleNode">
       <primary-key column-name="host_id"/>
   <reconciliation-by-two-nodes connected-node-cmdb-class-name="ip_address" cmd
b-link-type="containment">
       <or is-ordered="true">
       <attribute cmdb-attribute-name="name" column-name="host_name" ignore-
case="true"/>
       <attribute cmdb-attribute-name="data_note" column-name="note" ignore-
case="true"/>
          <connected-node-attribute cmdb-attribute-name="name" column-name="ip_a
ddress"/>
       </or>
       </reconciliation-by-two-nodes>
   </cmdb-class>
</generic-db-adapter-config>
```

Creare la TQL

La TQL è un **nodo** connesso a **ip_address** tramite un collegamento containment. Il nodo deve essere contrassegnato come **radice**, come indicato di seguito.



Per creare la TQL:

1. Selezionare **Modellazione > Studio di modellazione**.
2. Fare clic sul pulsante **Nuovo** e creare una nuova query.
3. Andare sulla scheda Tipi CI e trascinare il Tipo CI nodo e il Tipo CI IpAddress nella schermata TQL.
4. Collegare **Node** e **IpAddress** con una relazione Containment.
5. Fare clic con il pulsante destro del mouse sull'elemento **Node** e scegliere Proprietà nodo query.
6. Cambiare **Nome elemento** in **Radice**.
7. Andare sulla scheda **Layout elemento**. Selezionare **Attributi specifici** come condizione degli attributi. Scegliere **Nome** e **Nota** dalla finestra Attributi disponibili e spostarli nella finestra Attributi specifici.
8. Fare clic con il pulsante destro del mouse sull'elemento **IpAddress** e scegliere Proprietà nodo query.
9. Andare sulla scheda **Layout elemento**. Selezionare **Attributi specifici** come condizione degli attributi. Scegliere **Nome** dalla finestra Attributi disponibili e spostarlo nella finestra Attributi specifici.
10. Salvare la TQL.

Creare un punto di integrazione

Creare il Punto di integrazione come segue:

1. Selezionare **Gestione flusso di dati > Studio di integrazione**, quindi fare clic sul pulsante **Nuovo punto di integrazione**.
2. Inserire i dettagli del punto di integrazione e fare clic su **OK**.
3. Nella scheda Popolamento, selezionare il pulsante **Nuovo processo di integrazione** e aggiungere la TQL creata in precedenza.
4. Salvare il punto di integrazione e fare clic sul pulsante **Esegui sincronizzazione di tutti i dati**.

Configurare l'adattatore - Metodo avanzato

Questi file di configurazione si trovano nel pacchetto **db-adapter.zip** nella cartella **C:\hp\UCMDB\UCMDBServer\content\adapters** estratta durante la preparazione del pacchetto dell'adattatore. Per i dettagli consultare "Preparare il pacchetto dell'adattatore" a pagina 80.

Questo compito include i passaggi seguenti:

- "Configurare il file orm.xml" nel seguito
- "Configurare il file reconciliation_rules.txt " a pagina 91

Configurare il file orm.xml

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

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

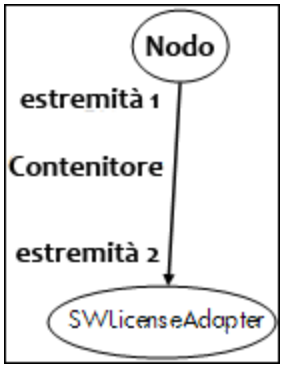
   Questo file, per impostazione predefinita, contiene un modello da utilizzare per mappare tutti i CIT e le relazioni necessari.

   > **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**:

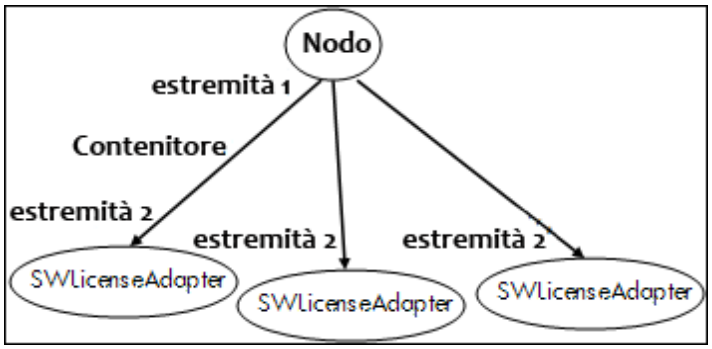
   - One to one:

   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>
   ```

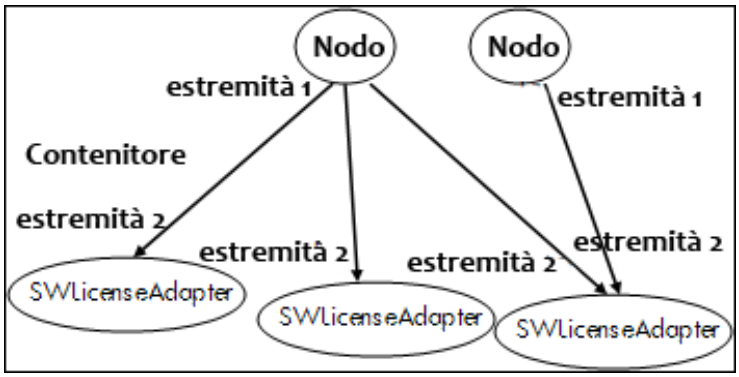
   - Many to one:



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 pagina 113.

Esempio di mapping 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 pagina 78:

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

- Il nome della tabella corrispondente del RDBMS:

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

- L'identità univoca nel RDBMS:

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

Esempio di mapping 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_rules.txt

In questo passaggio definire le regole in base alle quali l'adattatore riconcilia CMDB e il RDBMS (solo se viene utilizzato il motore di mapping 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 per i dati del database vi è distinzione tra 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 pagina 124.

Implementare un plug-in

Questo compito descrive come implementare e distribuire un adattatore DB generico 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 pagina 80.

1. Opzione 1 - Scrivere un plug-in basato su Java

   a. 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\db-adapter-framework** .
      - 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\APIs\DBAdapterFramework_JavaAPI\index.html**
      - **C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\Federation_JavaAPI\index.html**

   b. 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	FcmdbPluginForSyncGetFullTopology	getFullTopology
Sincronizzazione cambiamenti	FcmdbPluginForSyncGetChangesTopology	getChangesTopology
Sincronizzazione layout	FcmdbPluginForSyncGetLayout	getLayout
Recupero query supportate	FcmdbPluginForSyncGetSupportedQueries	getSupportedQueries
Modifica definizione query TQL e risultati	FcmdbPluginGetTopologyCmdbFormat	getTopologyCmdbFormat
Modifica richiesta layout per CI	FcmdbPluginGetCIsLayout	getCisLayout
Modifica richiesta layout per collegamenti	FcmdbPluginGetRelationsLayout	getRelationsLayout
Restituisci ID	FcmdbPluginPushBackIds	getPushBackIdsSQL

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.

Se **FcmdbPluginForSyncGetChangesTopology** viene implementato, esistono due modi diversi per segnalare i cambiamenti:

- **Segnalare l'intera topologia radice in qualsiasi momento.** In base a questa topologia, la funzione di eliminazione automatica rileva quale CI deve essere rimosso. In questo caso, tale funzione deve essere abilitata nel modo seguente:

```
<autoDeleteCITs isEnabled="true">
                <CIT>link</CIT>
                <CIT>object</CIT>
            </autoDeleteCITs>
```

- **Segnalare ogni istanza CI rimossa/aggiornata.** In questo caso, tale funzione deve essere abilitata nel modo seguente:

```
<autoDeleteCITs isEnabled="false">
                <CIT>link</CIT>
```

```
            <CIT>object</CIT>
        </autoDeleteCITs>
```

c. Accertarsi che il file JAR dell'SDK di federazione e i file JAR dell'adattatore DB generico si trovino nel percorso delle classi 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**.

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

2. Opzione 2 – Scrivere un plug-in basato su Groovy

a. Creare un file di codice Groovy (MyPlugin.groovy) nel menu Gestione adattatori, sotto i file di configurazione del pacchetto dell'adattatore.

b. Implementare le interfacce appropriate nella classe Groovy. Le interfacce sono definite nel file db-interfaces.jar (vedere la tabella precedente).

3. 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 re
sult
[getRelationsLayout]
# internal not mandatory plugin to change action on pushBackIds
[pushBackIds]
```

Legenda:

- Riga di commento.

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

Sotto ogni [<tipo di adattatore>] può esservi una riga vuota, a indicare che non vi sono classi plug-in associate, oppure vi può essere il nome completo della classe plug-in.

4. Inserire nell'adattatore il nuovo file jar e il file **plugins.xml** aggiornato. I restanti file del pacchetto devono essere gli stessi presenti in ogni adattatore basato sull'adattatore DB generico.

Distribuire l'adattatore

1. In UCMDB, accedere a Gestione pacchetti. Per i dettagli consultare "Package Manager Page" 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 adattatori" nella *Guida di Gestione flusso di dati di HP Universal CMDB*.

Creazione di un punto di integrazione

In questo passaggio si verifica 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 di dati > Studio di integrazione**).

2. Creare un punto di integrazione. Per i dettagli consultare New Integration Point/Edit Integration Point Dialog Box nella *Guida di Gestione 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 Federation Tab nella *Guida di Gestione 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 Create a Pattern View nella *Guida alla modellazione di HP Universal CMDB*.

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.

Visualizzazione dei report

In questo passaggio vengono visualizzati i report Topologia. Per i dettagli consultare Topology Reports Overview 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 pagina 145.

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 Assistenza HP Software.

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

- Consentire il mapping grafico tra classi di attributi di CMDB e colonne di tabelle del database.
- Abilitare la modifica manuale del file di mapping (`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 mapping 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 mapping.

La versione 1.1 del plug-in è compatibile con UCMDB versione 9.01 o successiva e con Eclipse IDE for Java EE Developers, versione 1.2.2.20100217-2310 o successiva.

Questo compito include i passaggi seguenti:

- "Prerequisiti" alla pagina successiva
- "Installazione" alla pagina successiva
- "Preparare l'ambiente di lavoro" alla pagina successiva
- "Creare un adattatore" a pagina 99
- "Configurare il plug-in di CMDB" a pagina 99
- "Importare il modello di classe di UCMDB" a pagina 99
- "Costruire il file ORM File - Mappare le classi di UCMDB alle tabelle del database" a pagina 100
- "Mappare gli ID" a pagina 100
- "Mappare gli attributi" a pagina 101
- "Mappare un collegamento valido" a pagina 101
- "Creare il file ORM File - Utilizzare le tabelle secondarie" a pagina 102
- "Definire una tabella secondaria" a pagina 102
- "Mappare un attributo a una tabella secondaria" a pagina 102
- "Utilizzare un file ORM esistente come base" a pagina 102
- "Importazione di un file ORM esistente da un adattatore" a pagina 103
- "Verificare la correttezza del file orm.xml - Verifica correttezza incorporata" a pagina 103
- "Creare un nuovo punto di integrazione" a pagina 103

- "Distribuire il file ORM in CMDB" a pagina 104
- "Eseguire una query TQL di esempio" a pagina 104

1. ## Prerequisiti

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

2. ## Installazione

   a. Scaricare ed estrarre **Eclipse IDE for Java EE Developers** da **http://www.eclipse.org/downloads**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**. 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>\bi
   n"
   ```

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.zip** da **C:\hp\UCMDB\ UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** in **C:\Documents and Settings\All Users.**

   > **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 > Area di lavoro > Altro**:

   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 pagina 28.

5. ## Configurare il plug-in di 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 di CMDB, ad esempio **localhost** oppure **labm1.itdep1**. Non è necessario includere il numero di porta o il prefisso `http://` all'indirizzo.

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

e. Accertarsi che la cartella **C:\hp** nel 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 deve 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 Types 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 database 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 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 precedentemente al punto "Costruire il file ORM File - Mappare le classi di UCMDB alle tabelle del database" alla pagina precedente per il mapping 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 pagina 124.

a. Mappare gli attributi dell'ID della classe del collegamento come descritto in "Mappare gli ID" alla pagina precedente. 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 precedentemente in "Mappare gli attributi" alla pagina precedente.

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 in tutto o in parte i mapping delle entità dal file esistente.

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

d. Incollare tutte le mappature dell'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 precedentemente nel passaggio "Costruire il file ORM File - Mappare le classi di UCMDB alle tabelle del database" a pagina 100. Tutte le entità incollate ora sono visualizzate nella vista JPA Structure. D'ora in poi i mapping potranno essere modificati sia graficamente che manualmente tramite 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.xml** 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 in CMDB" alla pagina successiva.

16. **Verificare la correttezza del file orm.xml - 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 mapping (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 in CMDB per questo adattatore, è possibile crearlo in Studio di integrazione. Per i dettagli consultare Integration Studio nella *Guida di Gestione 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 pagina 124.

18. **Distribuire il file ORM in CMDB**

    Salvare il file **orm.xml** e distribuirlo sul server UCMDB facendo 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 mapping al momento della distribuzione.

19. **Eseguire una query TQL di esempio**

    a. Definire una query)non vista) in Studio di modellazione. Per i dettagli consultare Modeling Studio nella *Guida alla modellazione di HP Universal CMDB*.

    b. Creare un punto di integrazione utilizzando l'adattatore creato nel passaggio "Creare un nuovo punto di integrazione" alla pagina precedente. Per i dettagli consultare New Integration Point/Edit Integration Point Dialog Box nella *Guida di Gestione 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 il nome di questa query di esempio nella finestra di dialogo Impostazioni. Per i dettagli vedere il passaggio precedente "Configurare il plug-in di CMDB" a pagina 99.

    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.

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 descritti i seguenti file di configurazione:

- "File transformations.txt" a pagina 126
- "File discriminator.properties" a pagina 127
- "File replication_config.txt" a pagina 128
- "File fixed_values.txt" a pagina 128
- "File Persistence.xml" a pagina 129

Configurazione generale

- **adapter.conf.** File di configurazione dell'adattatore. Per i dettagli consultare "File adapter.conf" alla pagina successiva.

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 pagina 107.

Configurazione avanzata

- **orm.xml**. File di mapping di tipo object-relational per mappare i CIT di CMDB e le tabelle del database. Per i dettagli consultare "File orm.xml" a pagina 109.
- **reconciliation_rules.txt**. Contiene le regole di riconciliazione. Per i dettagli consultare "File reconciliation_rules.txt (per la contabilità inversa)" a pagina 124.
- **transformations.txt.** File delle trasformazioni per indicare i convertitori da applicare per convertire il valore di CMDB nel valore del database e viceversa. Per i dettagli consultare "File transformations.txt" a pagina 126.
- **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 pagina 127.
- **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 pagina 128.
- **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 pagina 128.

Configurazione di Hibernate

- **persistence.xml.** Utilizzato per sostituire le configurazioni di Hibernate preimpostate. Per i dettagli consultare "File Persistence.xml" a pagina 129.

Abilitazione del supporto delle tabelle temporanee per l'adattatore

L'abilitazione delle tabelle temporanee consente all'adattatore di lavorare in modo più efficiente con il database remoto, riducendo il carico sul database e sulla rete e migliorando le prestazioni.

Per abilitare il supporto delle tabelle temporanee nell'adattatore generico del database, devono essere rispettate le seguenti condizioni:

- Le credenziali specificate per connettersi al database includono l'autorizzazione di creare, modificare ed eliminare le tabelle temporanee.
- Configurare le impostazioni seguenti nel file di configurazione adapter.conf:

  temp.tables.enabled=true

  performance.enable.single.sql=true.

Nota: Le tabelle temporanee sono supportate solamente per Microsoft SQL e Oracle.

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.

- **performance.memory.id.filtering=true**. Quando l'adattatore generico del database esegue i TQL, in alcuni casi un numero notevole di ID può essere recuperato e inviato al database tramite SQL. Per evitare questo lavoro in eccesso e migliorare le prestazioni, l'adattatore generico del database tenta di leggere l'intera vista/tabella e i filtra i risultati nella memoria.
- **id.reconciliation.cmdb.id.type=string/bytes**. Quando si esegue il mapping dell'adattatore DB generico mediante Riconciliazione ID, **cmdb_id** può essere mappato a un tipo di colonna **string** o **bytes/raw** cambiando la proprietà **META-INF/ adapter.conf**.
- **performance.enable.single.sql=true**. Questo parametro è facoltativo. Se non compare nel file,

il suo valore predefinito è **true**. Quando è **true**, l'adattatore generico del database tenta di generare un'istruzione SQL singola per ogni query eseguita (sia per il popolamento che per una query federata). L'utilizzo di un'istruzione SQL singola migliora le prestazioni e il consumo di memoria dell'adattatore generico del database. Quando **false**, l'adattatore generico del database genera istruzioni SQL multiple che possono richiedere più tempo e consumare più memoria rispetto a una singola. Anche quando l'attributo è impostato su **true**, l'adattatore non genera un'istruzione SQL singola nei seguenti scenari:

- Il database a cui l'adattatore si connette non è un server Oracle o SQL.
- La TQL da eseguire contiene una condizione di cardinalità diversa da 0..* e 1..* (ad esempio, se è presenta una condizione di cardinalità pari a 2..* o 0..2).

- **in.expression.size.limit=950** (predefinito). Questo parametro divide l'espressione 'IN' della SQL eseguita, quando si raggiunge il limite delle dimensioni relativo all'elenco degli argomenti.
- **stringlist.delimiter.of.<Nome CIT>.<Nome attributo>=<delimitatore>**. Per poter mappare un attributo elenco stringhe a una colonna di database nell'adattatore database generico, l'attributo deve essere mappato a una colonna stringa contenente un elenco di valori concatenati. Ad esempio, per mappare l'attributo **policy_category** al tipo CI **policy**, dove la colonna stringa contiene un elenco di valori: valore1##valore2##valore3 (che definisce un elenco di 3 valori valore1, valore2, valore3), utilizzare l'impostazione: **stringlist.delimiter.of.policy.policy_category=##.**
- **temp.tables.enabled=true**. Consente l'utilizzo di tabelle temporanee per migliorare le prestazioni. Disponibile solo quando è abilitato **performance.enable.single.sql** (supportato solo in Microsoft SQL e Oracle). Possono essere necessarie determinate autorizzazioni nel server del database.
- **temp.tables.min.value=50**. Definisce il numero di valori di condizione (o ID) necessari per utilizzare le tabelle temporanee.

File simplifiedConfiguration.xml

Questo file viene utilizzato per il mapping semplice delle classi di UCMDB alle tabelle del database. Per accedere al modello per la modifica del file selezionare **Gestione adattatori > db-adapter > File di configurazione**.

In questa sezione vengono trattati i seguenti argomenti:

- "Modello del file simplifiedConfiguration.xml" nel seguito
- "Limitazioni" a pagina 109

Modello del file simplifiedConfiguration.xml

- La proprietà **CMDB-class-name** è 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-instanc
e" 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 in CMDB e nel database non sono gli stessi.

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.impl. GenericEnumTransformer
(generic-enum-transformer-example.xml)" to-CMDB-onverter="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 il mapping dei CIT di CMDB alle tabelle del database.

Un modello da utilizzare per la creazione di un nuovo file si trova nella directory **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INF**.

Per modificare il file XML per un adattatore distribuito passare a **Gestione adattatori > db-adapter > File di configurazione**.

In questa sezione vengono trattati i seguenti argomenti:

Modello di file orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xs
i="http://www.w3.org/2001/XMLSchema-instance" version="1.0" xsi:schemaLocati
on=
"http://java.sun.com/xml/ns/persistence/orm http://java.sun.com/xml/ns/persi
stence/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 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_n
ame]" />
                <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_n
ame]" />
            </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_cl
ass_name]">
        <table name="[default_table_name]" />
        <secondary-table name="" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <generated-value strategy="TABLE" />
            </id>
            <id name="id2">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <generated-value strategy="TABLE" />
```

```
            </id>
            <id name="id3">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <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_n
ame]" />
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
            </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_adap
ter.node_composition_[CMDB_class_name]">
        <table name="[default_table_name]" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <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>
```

node attribute. Di seguito un esempio di come aggiungere un attributo del nodo.

```
<entity class="generic_db_adapter.host_node">
  <discriminator-value>host_node</discriminator-value>
    <attributes/>
</entity>
<entity class="generic_db_adapter.nt">
  <discriminator-value>nt</discriminator-value>
    <attributes>
      <basic name="nt_servicepack">
        <column updatable="false" insertable="false" name="specific_type_value"/>
      </basic>
    </attributes>
</entity>
```

File ORM multipli

Sono supportati file di mapping multipli. Ciascun nome di file di mapping deve terminare con **orm.xml**. Tutti i file di mapping 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 primarie 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`.
- Ai CIT di CMDB che sono anche parole riservate in Java deve essere anteposto il prefisso **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 f
rom
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 è consultabile al sito in inglese 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.

Nome e percorso elemento	Descrizione	Attributi
entity-mappings	L'elemento radice per il documento di mapping 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 mapping dell'entità. (Facoltativo)	

Nome e percorso elemento	Descrizione	Attributi
package (entity-mappings)	Nome del pacchetto Java che conterrà le classi di mapping. Dovrebbe contenere sempre il testo `generic_db_adapter`.	1. **Nome:** nome **Descrizione:** il 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. **Obbligatorio?:** Facoltativo **Tipo:** String 2. **Nome:** class **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 l'interfaccia sarà `generic_db_adapter.gdba_interface`. **Obbligatorio?:** Obbligatorio **Tipo:** String

Nome e percorso elemento	Descrizione	Attributi
table (entity-mappings>entity)	Questo elemento definisce la tabella primaria dell'entità del database. Può essere visualizzato solo una volta. Obbligatorietà.	**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. Può essere qualsiasi istruzione SELECT valida. Se è una dichiarazione SELECT, deve essere incapsulata tra parentesi. **Obbligatorio?:** 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. Può essere qualsiasi istruzione SELECT valida. Se è una dichiarazione SELECT, deve essere incapsulata tra parentesi. **Obbligatorio?:** Obbligatorio **Tipo:** String
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. **Obbligatorio?:** Facoltativo **Tipo:** String

Nome e percorso elemento	Descrizione	Attributi
inheritance (entity-mappings > entity)	Se l'entità corrente è l'entità padre di una famiglia di entità del database, utilizzare questo elemento per contrassegnarla come tale. Facoltativo.	**Nome:** strategy **Descrizione:** Definisce il modo di implementazione dell'ereditarietà nel database. **Obbligatorio?:** Obbligatorio **Tipo:** Uno dei valori seguenti: • SINGLE_TABLE: Questa entità e tutte le entità figlio esistono nella stessa tabella. • JOINED: Le entità figlio 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. **Obbligatorio?:** Obbligatorio **Tipo:** String
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 radice per tutti i mapping degli attributi di un'entità.	

Nome e percorso elemento	Descrizione	Attributi
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. **Obbligatorio?:** Obbligatorio **Tipo:** String
basic (entity-mappings > entity attributes)	Questo elemento definisce un mapping 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. **Obbligatorio?:** Obbligatorio **Tipo:** String
column (entity-mappings > entity > attributes >id -OR- (entity-mappings > entity > attributes > basic)	Definisce il nome della colonna nella tabella per il mapping di base o un campo id.	1. **Nome:** name **Descrizione:** Nome del campo. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **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. **Obbligatorio?:** Facoltativo **Tipo:** String

Nome e percorso elemento	Descrizione	Attributi
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 il mapping delle entità collegamento e non per altri tipi di CI. Questo è l'unico modo per definire un mapping tra una tabella e un collegamento di UCMDB.	1. **Nome:** name **Descrizione:** Quale delle due estremità rappresenta il campo. **Obbligatorio?:** Obbligatorio **Tipo:** O `end1` oppure `end2` 2. **Nome:** target-entity **Descrizione:** Nome dell'entità alla quale si riferisce il campo. **Obbligatorio?:** Obbligatorio **Tipo:** Uno dei nomi dell'entità definito nel documento di mapping dell'entità.
join-column (entity-mappings > entity attributes > one-to-one)	Definisce il modo di giunzione dell'entità di destinazione definita nell'elemento padre one-to-one e l'entità corrente.	1. **Nome:** name **Descrizione:** Nome del campo nella tabella corrente che sarà utilizzato per eseguire giunzioni di tipo one-to-one. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **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. **Obbligatorio?:** Facoltativo **Tipo:** String

Esempio di creazione del file orm.xml

L'esempio qui rappresentato mostra come creare il **file orm.xml**. In questo esempio, le tabelle SQL in a database remoto vengono mappate sui tipi CI in UCMDB.

Alla luce delle tabelle con il seguente formato nel database remoto, popolare la tabella **Hosts** con i nodi, la tabella **IP_Addresses** con gli indirizzi IP e creare i collegamenti tra quest'ultimi e i nodi come segue:

Tabella host

host_name	host_id
Test1	1
Test2	2
Test3	3

Tabella IP_Addresses

ip_address	ip_id
10.1.1.1	1
10.2.2.2	2
10.3.3.2	3
10.4.4.4	4

Tabella Host_IP_Link (collegamenti tra nodi e indirizzi IP)

host_id	ip_id
1	1
2	2
2	3
3	4

La chiave primaria per la tabella **Hosts** è il campo **host_id** e quella della tabella **IP_Addresses Table** è il campo **ip_id**. Nella tabella **Host_IP_Link**, **host_id** e **ip_id** sono chiavi esterne delle tabelle **Hosts** e **IP_Addresses**.

In base alle tabelle sopra citate, creare il file **orm.xml** secondo i seguenti passaggi: Le entità usate in questo esempio sono **node**, **ip_address** e **node_containment_ip_address**

1. Creare l'entità **node** tramite il mapping di **host_id** dalla tabella **Hosts** come segue:

```
<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_1_0.xsd">
    <description>test_integration</description>
    <package>generic_db_adapter</package>
```

```
    <entity class="generic_db_adapter.node">
       <table name="Hosts"/>
       <attributes>
        <id name="id1">
                <column updatable="false" insertable="false" name=
                       "host_id"/>
                <generated-value strategy="TABLE"/>
        </id>
        <basic name="name">
                <column updatable="false" insertable="false" name=
                       "host_name"/>
        </basic>
       </attributes>
    </entity>
```

La **classe** dell'entità deve essere un tipo CI già esistente in UCMDB. Il **nome** della tabella è la tabella del database contenente sia un ID sia le informazioni sull'host. L'attributo ID è necessario per identificare host specifici e viene utilizzato in seguito per il mapping. In questo esempio, popolare l'attributo **name** di questa entità con la colonna **nome host** nella tabella host.

2. Per l'entità successiva, mappare gli indirizzi IP dalla tabella Interfacce.

```
<entity name="ip_address" class="generic_db_adapter.ip_address">
    <table name="IP_Addresses"/>
    <attributes>
       <id name="id1">
        <column insertable="false" updatable="false" name="ip_id"/>
        <generated-value strategy="TABLE"/>
       </id>
       <basic name="name">
        <column updatable="false" insertable="false" name="ip_address"/>
       </basic>
    </attributes>
```

```
</entity>
```

3. Quindi, creare un collegamento tra il nodo e l'indirizzo IP tramite la tabella di mapping, con riferimento al campo **ip_id** (sebbene si possa far riferimento a entrambi i campi **host_id** e **ip_id**, se si desidera).

```
<entity name="node_containment_ip_address"
        class="generic_db_adapter.node_containment_ip_address">
    <table name="Host_IP_Link"/>
    <attributes>
       <id name="id1">
        <column updatable="false" insertable="false" name="ip_id"/>
        <generated-value strategy="TABLE"/>
       </id>
       <many-to-one target-entity="node" name="end1">
        <join-column name="host_id"/>
       </many-to-one>
       <one-to-one target-entity="ip_address" name="end2">
        <join-column name="ip_id"/>
       </one-to-one>
    </attributes>
</entity>
```

Il nome dell'entità del contenitore segue il seguente formato: [end1 CIT]_[link CIT]_[end2 CIT]. Per tale esempio, poiché il tipo CI di collegamento è **containment**, il nome entità del contenitore sarà: **node_containment_ip_address** e la classe entità **generic_db_adapter.node_containment_ip_address**. Per questo blocco di codici l'ID è necessario e, mentre in questo esempio c'è un ID singolo dell'interfaccia, entrambe le colonne possono fare riferimento sia a id1 sia a id2. In questo caso il codice sarà:

```
<id name=”id1”>
    <column updatable=”false” insertable=”false” name=”ip_id”/>
    <generated-value strategy=”TABLE”/>
</id>
```

```
<id name=”id2”>
    <column updatable=”false” insertable=”false” name=”host_id”/>
    <generated-value strategy=”TABLE”/>
</id>
```

Le due estremità di questo collegamento sono ‘many-to-one’ e ‘one-to-one’, ovvero ogni indirizzo IP sarà collegato a 1 nodo ma un nodo può essere collegato a più indirizzi IP. Le colonne incluse sono contenute nella tabella collegamenti e fanno riferimento alle tabelle host e interfacce.

Configurazione di un orm.xml specifico per ogni versione prodotto remoto

È possibile configurare un file **orm.xml** specifico in modo che l'adattatore utilizzi un **orm.xml** specifico per una versione prodotto remoto specificata. Ad esempio, se l'archivio dati remoto ha due versioni del prodotto x e y, per ogni versione può esservi un mapping diverso delle entità.

Per configurare un file orm.xml specifico in base alla versione prodotto remoto:

1. Aggiungere al file **adapter.xml** un parametro chiamato **version** e specificare i possibili valori della versione come **valid-values**.
2. Nel pacchetto dell'adattatore, nella cartella META-INF, creare una cartella chiamata **VersionOrm**.
3. Nella cartella **VersionOrm**, creare un file **orm.xml** per ogni versione specifica. Il nome del file deve contenere il prefisso della versione. Ad esempio, se la versione è chiamata **x**, il nome del file deve essere **x_orm.xml**.

Nota: Il file **orm.xml** nella cartella META-INF viene caricato per ogni versione prodotto remoto, che l'utente crei o meno un file **orm.xml** specifico per una versione prodotto remoto. Può avere entità mappate nello stesso modo per tutte le versioni.

File reconciliation_types.txt

A partire da UCMDB 10.00, il file **reconciliation_types.txt** non è più rilevante. Per la riconciliazione si può utilizzare qualsiasi CIT. Il motore di federazione esegue il mapping automaticamente.

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 (il CIT di CMDB collegato al CIT del database federato nella query TQL).

Questa espressione comprende la logica che decide se due multinodi sono uguali (un multinodo in CMDB e l'altro nell'origine 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 indica che i multinodi non sono uguali, la seconda sottoespressione OR non viene confrontata), 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 query TQL di riconciliazione contiene due nodi e non soltanto un multinodo. In questo caso, la query 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 E/Oname 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 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 CMDB o l'origine dati non hanno `name of node`, viene verificato `name of ip_address`.

- **Regular match.** Se c'è uguaglianza in una delle sottoespressioni `OR`, 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`), il mapping di un nodo soprainsieme 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 query TQL di riconciliazione non deve avere più di due nodi. Ad esempio, la query TQL `node > ticket` ha un nodo in CMDB e un ticket nell'origine dati.

Per riconciliare i risultati, è necessario recuperare `name` da node e/o `ip_address`.

Se `name` in CMDB è nel formato `*.m.com`, è possibile utilizzare un convertitore da 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):

Nodo DB	
PK	node_id
	nome

DB IP_Address	
PK	ip_id
	nome

Ticket DB	
PK	ticket_id
	node_id

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.

Modello del file transformations.txt

```
entity[[CMDB_class_name]] attribute[[CMDB_attribute_name]] to_DB_class[com.mercu
ry.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.FcmdbDalTransformerTo.ExternalDB**. 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 CMDB, ad esempio per aggiungere il suffisso **.com** a ciascun nome di nodo.

Per i dettagli consultare "Convertitori preimpostati" a pagina 130.

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 oppure una condizione che corrisponde a valori possibili della colonna del discriminatore.

Se viene utilizzata una condizione, ricorrere alla sintassi: `like(condition)`, dove `condition` è una stringa che può contenere i seguenti caratteri jolly:

- **%** (percentuale) - consente di confrontare qualsiasi stringa di qualunque lunghezza (includendo una stringa di lunghezza zero)
- **_** (sottolineatura) - consente di confrontare un singolo carattere

Ad esempio, `like(%unix%)` corrisponderà a unix, linux, unix-aix, e così via. Le condizioni Simili a si possono applicare solo alle colonne di stringhe.

Inoltre, è possibile avere un valore del discriminatore singolo mappato su ogni valore non appartenente a nessun altro discriminatore tramite l'affermazione `'all-other'`.

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 mapping del discriminatore:

Ad esempio, l'adattatore supporta i tipi di CI `node`, `nt` e `unix`, e il database contiene una tabella singola nominata `t_nodes` contenente la colonna **tipo**. Se il tipo è 10001, la riga rappresenta un nodo; se il tipo è 10004, rappresenta un computer UNIX e così via. Il file **discriminator.properties** può avere il seguente aspetto:

```
node=10001, 10005
nt=10002,10003
```

```
unix=2%
mainframe=all-other
```

Il file **orm.xml** include il codice seguente:

```
    <entity class="generic_db_adapter.node" >
        <table name="t_nodes" />
        ...
        <inheritance strategy="SINGLE_TABLE" />
        <discriminator-value>node</discriminator-value>
        <discriminator-column name="type" />
        ...
    </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 quindi calcolato come segue:

- Se **tipo** contiene `10002` o `10003` per una determinata entità, la voce viene mappata al CIT **nt**.
- Se **tipo** contiene `10001` o `10005` per una determinata entità, la voce viene mappata al CIT **node**.
- Se **tipo** inizia con `2` per una determinata entità, la voce viene mappata al CIT **unix**.
- Ogni altro valore nella colonna **tipo** viene mappato al CIT **mainframe**.

> **Nota:** Il CIT **node** è anche il 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 pagina 135.

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]
```

Il file supporta inoltre un elenco di costanti. Per definire un elenco di costanti, utilizzare la seguente sintassi:

```
entity[<entityName>] attribute[<attributeName>] value[{<Val1>, <Val2>, <Val3>, .
.. }]
```

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 SQL 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** (per i dettagli, visitare il sito http://www.hibernate.org in inglese).

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">
        <properties>
            <!--  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.me
rcury.
               jdbc.MercOracleDriver" />
            <!--The connection url"/-->
            <property name="hibernate.connection.url" value="jdbc:mercury:or
acle:
               //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>
```

Connessione al database mediante l'autenticazione NT

È possibile connettersi a un server MS SQL che richiede l'autenticazione NT. Per fare ciò è necessario un driver in grado di analizzare il dominio (driver jTDS JDBC).

L'autenticazione viene eseguita secondo i parametri dati (dominio, nome utente, password) e non con le credenziali NT del processo in esecuzione corrente.

1. In **persistence.xml** modificare le proprietà seguenti:

```
<!--The driver class name"/-->
<property name="hibernate.connection.driver_class"
value="net.sourceforge.jtds.jdbc.Driver"/>
<property name="hibernate.connection.url" value="jdbc:jtds:sqlserver://[host name]:
[port];DatabaseName=[database name];domain=[the domain]"/>
<!--DB login credentials"/-->
<property name="hibernate.connection.username" value="[username]"/>
<property name="hibernate.connection.password" value="[password]"/>
```

2. Posizionare il file del driver JDBC in: **<cartella di installazione sonda>\lib**.

3. Riavviare la sonda.

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 StringDelimitedListTransformer" a pagina 134
- "Convertitore personalizzato" a pagina 134

Convertitore enum-transformer

Questo convertitore utilizza un file XML che viene assegnato come parametro di input.

Il file XML mappa i valori hardcoded 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.

Il convertitore esegue una confronto tra due stringhe distinguendo o meno tra maiuscole/minuscole. Per impostazione predefinita, distingue tra maiuscole/minuscole. Per definire l'utilizzo senza distinzione tra maiuscole/minuscole: `case-sensitive="false"` nell'elemento enum-transformer.

Utilizzare un file di mapping 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**.

File di input XSD:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="q
ualified" 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:attribute name="case-sensitive" use="optional">
        <xs:simpleType>
```

```
                <xs:restriction base="xs:boolean">
                </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="require
d"/>
        <xs:attribute name="is-cmdb-value-null" type="xs:boolean" use="optio
nal"/>
        <xs:attribute name="is-db-value-null" type="xs:boolean" use="optiona
l"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Esempio di conversione del valore 'sys' nel valore 'System':

In questo esempio il valore `sys` in CMDB viene trasformato nel valore `System` nel database federato e il valore `System` nel database federato viene trasformato nel valore `sys` in 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-act
ion="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>
```

Esempio di conversione di un valore esterno o di CMDB in un valore null:

In questo esempio, un valore di **NNN** nel database remoto viene trasformato in un valore null nel database di CMDB.

```
<value cmdb-value="null" is-cmdb-value-null="true" external-db-value="NNN"/>
```

In questo esempio, il valore **OOO** in CMDB viene trasformato in un valore null nel database remoto.

```
<value cmdb-value="OOO" external-db-value="null" is-db-value-null="true"/>
```

Convertitore SuffixTransformer

Questo convertitore viene utilizzato per aggiungere o rimuovere suffissi dal valore di origine del database di CMDB o 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 di 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 di CMDB al valore del database federato.

Convertitore PrefixTransformer

Questo convertitore viene utilizzato per aggiungere o rimuovere un prefisso dal valore del database di CMDB o 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 di 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 di CMDB al valore del database federato.

Convertitore BytesToStringTransformer

Questo convertitore viene utilizzato per convertire matrici di byte in CMDB nella rispettiva rappresentazione stringa nell'origine del database federato.

Il convertitore è:
com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer.

Convertitore StringDelimitedListTransformer

Questo convertitore viene utilizzato per convertire un elenco di stringhe singole in un elenco di stringhe intere in CMDB.

Il convertitore è: **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.StringDelimitedListTransformer.**

Convertitore personalizzato

È possibile scrivere un convertitore (trasformatore) personalizzato da zero. In questo modo è possibile creare qualsiasi convertitore necessario per le proprie esigenze.

Vi sono due modi per scrivere un convertitore personalizzato:

1. Scrivere un convertitore Java compilato
   a. Creare un progetto Java in un IDE Java (ad esempio Eclipse, Intellij o Netbeans).
   b. Aggiungere federation-api.jar e db-interfaces.jar al percorso delle classi.
   c. Creare una classe Java che implementa le interfacce seguenti (da **db-interfaces.jar**):
      - FcmdbDalTransformerFromExternalDB
      - FcmdbDalTransformerValuesToExternalDB
      - FcmdbDalTransformerInit
   d. Compilare il progetto e creare un file jar.
   e. Inserire il file jar nel pacchetto dell'adattatore (in adapterCode\<ID adattatore>)
   f. Distribuire il pacchetto.
   g. Aggiungere il nome della nuova classe convertitore al file **transformations.txt**.
2. Scrivere un convertitore groovy (basato su script)

   Un esempio si trova nel pacchetto GDBA originale, **GroovyExampleTransformer.groovy**.

   a. Creare un file groovy nel pacchetto dell'adattatore (in adapterCode\<ID adattatore>). È possibile farlo direttamente utilizzando il menu Gestione adattatori.
   b. Creare una classe Groovy che implementa le interfacce seguenti (da **db-interfaces.jar**):
      - FcmdbDalTransformerFromExternalDB
      - FcmdbDalTransformerValuesToExternalDB
      - FcmdbDalTransformerInit
   c. Aggiungere il nome della nuova classe groovy convertitore al file **transformations.txt**.

   Nota: Groovy è un linguaggio di scripting che estende Java. Il codice Java regolare è codice valido anche in Groovy.

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 la sincronizzazione del layout.
- 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.
- Plug-in interno facoltativo per cambiare l'azione Restituisci ID.

Per i dettagli sull'implementazione e la distribuzione di plug-in, consultare "Implementare un plug-in" a pagina 92.

Esempi di configurazione

In questa sezione vengono presentati esempi di configurazione.

In questa sezione vengono trattati i seguenti argomenti:

Caso di utilizzo

Una query TQL è:

node > (composition) > card

dove:

- **node** è l'entità di 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-insta
nce" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xs
d">
    <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_Na
me" />
            </or>
        </reconciliation-by-single-node>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMD
B-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="hwCardVend
or" />
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"
/>
    </class>
</generic-DB-adapter-config>
```

Definizione avanzata

File orm.xml

Prestare attenzione all'aggiunta del mapping della relazione. Per i dettagli consultare la sezione sulla definizione in "File orm.xml" a pagina 109.

Esempio di file orm.xml:

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

```
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xs
i="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_rules.txt

Per i dettagli consultare "File reconciliation_rules.txt (per la contabilità inversa)" a pagina 124.

```
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` O di `ip_address`:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
nce" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xs
d">
    <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_addr
ess" CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-name="Device_Na
me" />
```

```
                <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-CMD
B-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="hwCardVend
or" />
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"
/>
    </class>
</generic-DB-adapter-config>
```

La riconciliazione avviene per `name` di `node` E di `ip_address`:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
nce" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xs
d">
    <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_addr
ess" CMDB-link-type="containment">
            <and>
                <attribute CMDB-attribute-name="name" column-name="Device_Na
me" />
                <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-CMD
B-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="hwCardVend
or" />
        <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_rules.txt

Per i dettagli consultare "File reconciliation_rules.txt (per la contabilità inversa)" a pagina 124.

```
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] en
d2_type[ip_address] link_type[containment]

multinode[node] expression[ip_address.name] end1_type[node] end2_type[ip_add
ress] 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-CMD
B-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="hwCardVend
or" />
        <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 pagina 109.

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="fals
e"/>
                <generated-value strategy="TABLE" />
            </id>
            <id name="id2">
                <column name="hwBusesSupported_Seq" insertable="false" updat
able="false"/>
                <generated-value strategy="TABLE" />
            </id>
            <id name="id3">
                <column name="hwCards_Seq" insertable="false" updatable="fal
se"/>
                <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="fals
e"/>
                <generated-value strategy="TABLE" />
            </id>
            <id name="id2">
                <column name="hwBusesSupported_Seq" insertable="false" updat
able="false"/>
                <generated-value strategy="TABLE" />
            </id>
            <id name="id3">
                <column name="hwCards_Seq" insertable="false" updatable="fal
se"/>
                <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-n
ame="hwBusesSupported_Seq" insertable="false" updatable="false"/>
                <join-column name="hwCards_Seq" referenced-column-name="hwCa
rds_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 mapping 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:noNa
mespaceSchemaLocation="../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.impl.GenericEnumTransformer(generic-enum-transformer-
example.
                 xml)" to-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAda
pter.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.
GenericEnumTransformer(generic-enum-transformer-example.xml)] from_DB_class
```

```
[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(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" nel seguito
- "Percorsi dei registri" nel seguito

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 dei registri

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 mapping. Fornisce i dettagli sulla query TQL di riconciliazione che utilizza il motore di mapping 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 mapping 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 pagina 73.

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

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 modello. 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 delle classi CMDB devono seguire la convenzione 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 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

- L'`id` CMDB del CI esterno è composto dalla relativa 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.

Capitolo 5: Sviluppo degli adattatori Java

Questo capitolo comprende:

Panoramica del framework di federazione

Nota:

- Il termine **relazione** equivale al termine **collegamento**.
- Il termine **CI** equivale al termine **oggetto**.
- Un grafico è una raccolta di nodi e collegamenti.

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 in CMDB.
- **Popolamento.** Popola i dati (dati topologici e proprietà CI) in CMDB da un'origine dati esterna.
- **Invio dati.** Invia i dati (dati topologici e proprietà CI) da 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" nel seguito
- "Invio dati" alla pagina successiva
- "Popolamento" a pagina 152

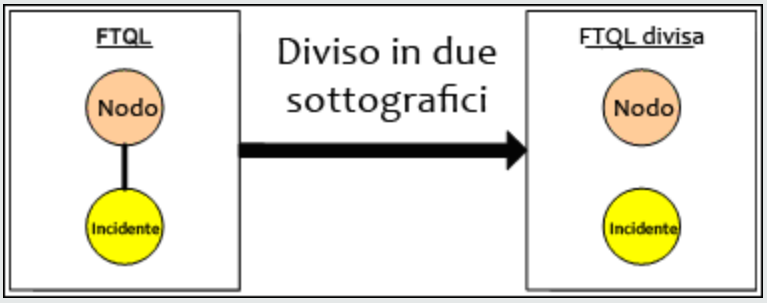
Federazione in tempo reale

Le query TQL federate permettono il recupero dei dati da un repository di dati esterno senza replicarne i relativi dati.

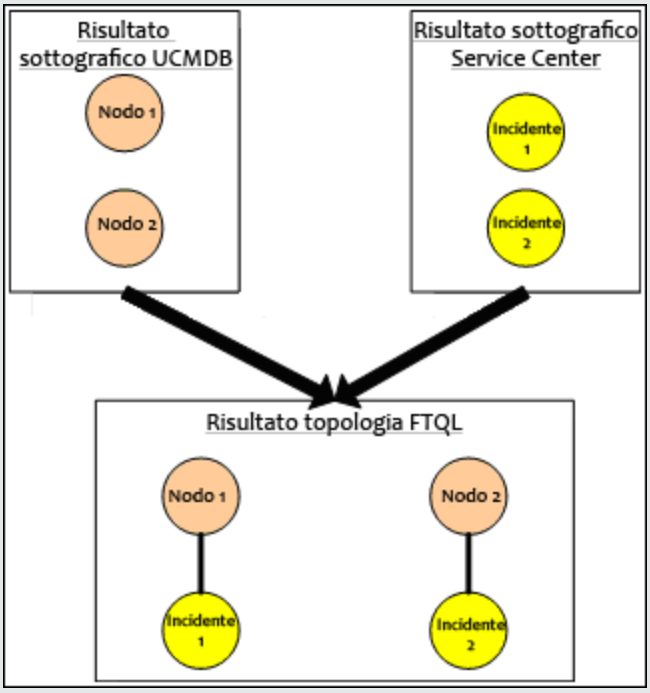
Una query TQL federata utilizza adattatori che rappresentano repository di dati esterni, al fine di creare relazioni esterne adeguate tra CI di repository di dati esterni diversi e CI di UCMDB.

Esempio di flusso di federazione in tempo reale:

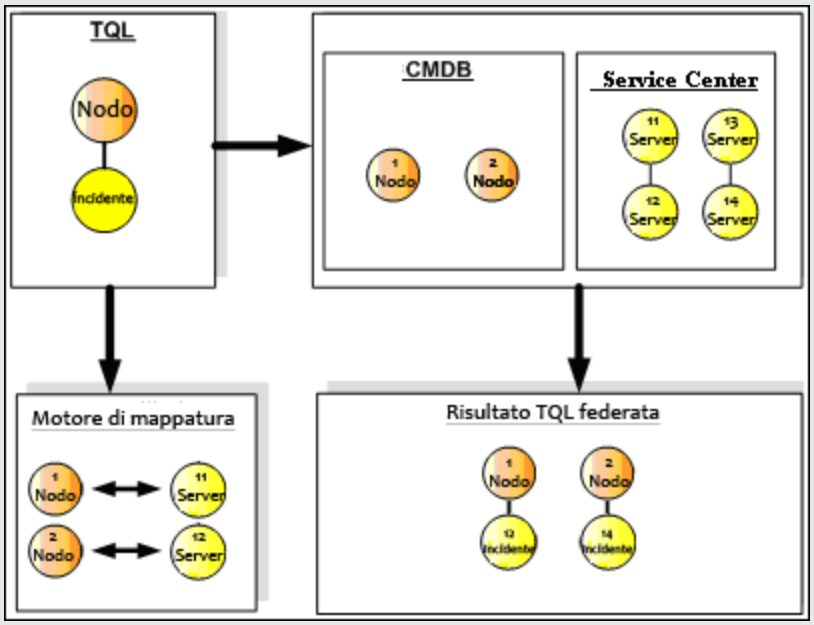
1. Il framework di federazione suddivide una query TQL federata in vari sottografici; tutti i nodi di un sottografico fanno riferimento allo stesso repository di dati. Ciascun sottografico è connesso ad altri sottografici da una relazione virtuale (ma non contiene di per sé relazioni virtuali).



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 correlati creando relazioni virtuali tra i nodi appropriati.



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.

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 CMDB con dati provenienti da 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 Data Flow Probe 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 su CMDB.

Il flusso di popolamento si basa sui nomi query, ovvero i dati vengono sincronizzati tra il repository di dati di origine e la Data Flow Probe, 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 nessun'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.

Flusso di popolamento basato su istanze

Se l'adattatore è definito per supportare un flusso basato su istanze (per mezzo del tag **<instance-based-data>**, come descritto in "Proprietà e tag di configurazione XML" a pagina 178), il motore di popolamento individua automaticamente i CI rimossi nell'istanza e li rimuove da UCMDB (a condizione che l'eliminazione sia consentita per il processo di popolamento specifico). Ogni istanza deve avere un CI radice, contrassegnato nella definizione TQL con il nome **Root**. Ogni volta che viene inviato un CI radice, la sua intera istanza (tutti i CI connessi ad esso) viene confrontata all'ultima volta che è stata inviata a UCMDB, ed eventuali CI che erano connessi alla radice ma ora non lo sono più vengono eliminati da UCMDB. Ogni modifica a qualsiasi CI o attributo nell'intera istanza deve attivare un reinvio dell'intera istanza a UCMDB per far si che l'adattatore supporti correttamente il flusso basato su istanze.

Interazione dell'adattatore e del mapping 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" nel seguito
- "Metodi assist dell'adattatore" nel seguito

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 Data Flow Probe, 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 classi

esterne 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.

Framework di federazione per le query TQL federate

In questa sezione vengono trattati i seguenti argomenti:

- "Definizioni e termini" nel seguito
- "Motore di mapping" nel seguito
- "Adattatore federato" alla pagina successiva

Consultare "Interazioni tra framework di federazione, server, adattatore e motore di mapping" alla pagina successiva per i diagrammi che illustrano l'interazione tra il framework di federazione, UCMDB, l'adattatore e il motore di mapping.

Definizioni e termini

Dati di riconciliazione. La regola per creare la corrispondenza tra i CI del tipo specificato che vengono ricevuti da CMDB e il 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 node 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 mapping per eseguire la connessione tra CI esterni e CI di CMDB.

Tipo CI di riconciliazione. Il tipo di CI che rappresentano oggetti di riconciliazione. Questi CI devono essere memorizzati sia in CMDB, sia nei repository di dati esterni.

Motore di mapping. 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 mapping

Il framework di federazione utilizza il motore di mapping per calcolare la query TQL federata. Il motore di mapping esegue la connessione tra CI ricevuti da diversi repository di dati e connessi

tramite relazioni virtuali. Il motore di mapping 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 in 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 di 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.

Interazioni tra framework di federazione, server, adattatore e motore di mapping

I seguenti diagrammi illustrano le interazioni tra il framework di federazione, UCMDB, l'adattatore e il motore di mapping. 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.

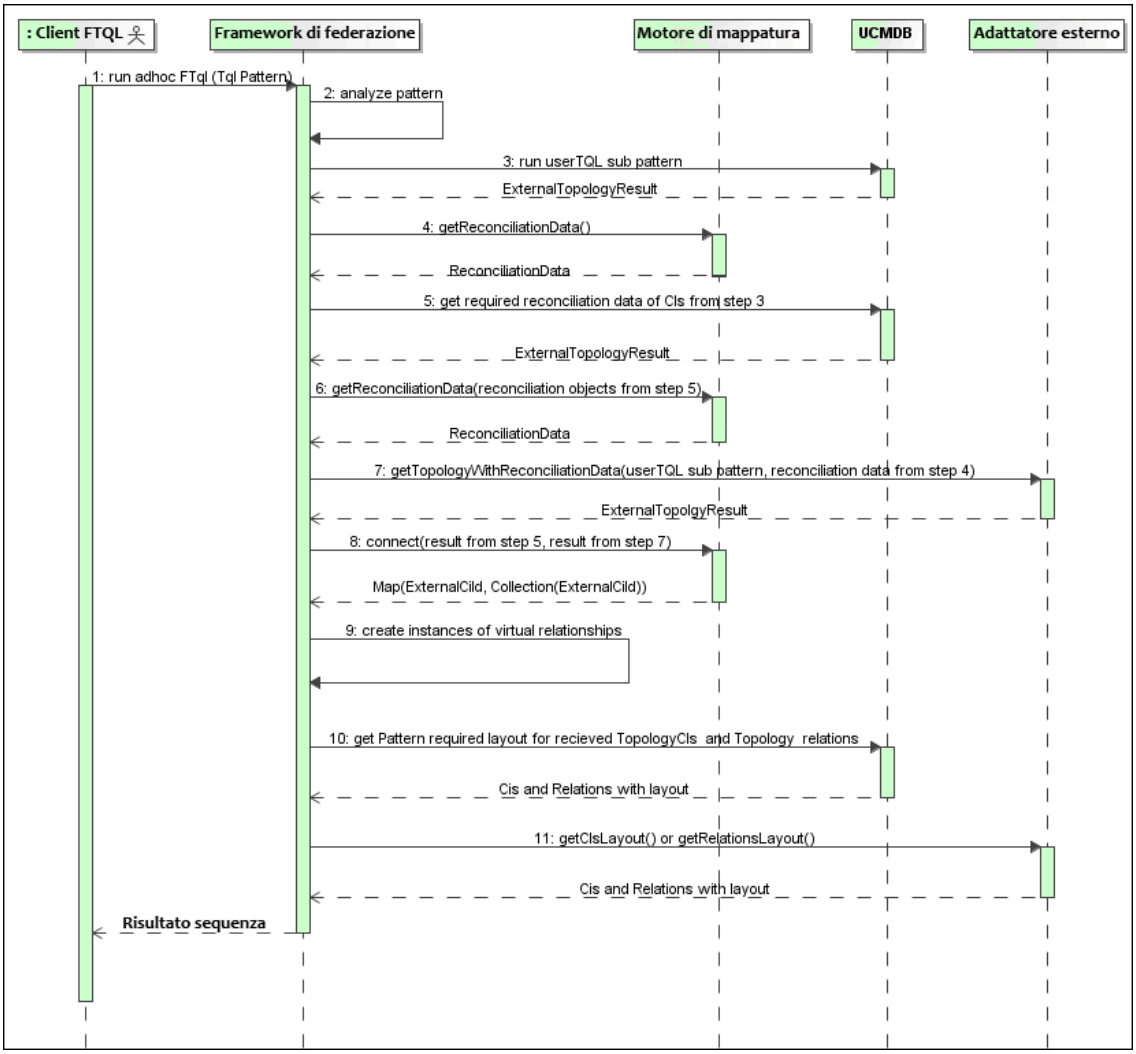
In questa sezione vengono trattati i seguenti argomenti:

- "Il calcolo inizia sul lato server" nel seguito
- "Il calcolo inizia sul lato dell'adattatore esterno" a pagina 159
- "Esempio di flusso del framework di federazione per le query TQL federate" a pagina 160

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 mapping o adattatore.

Il calcolo inizia sul lato server

Il seguente diagramma sequenza illustra l'interazione tra il framework di federazione, UCMDB, l'adattatore e il motore di mapping. 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.

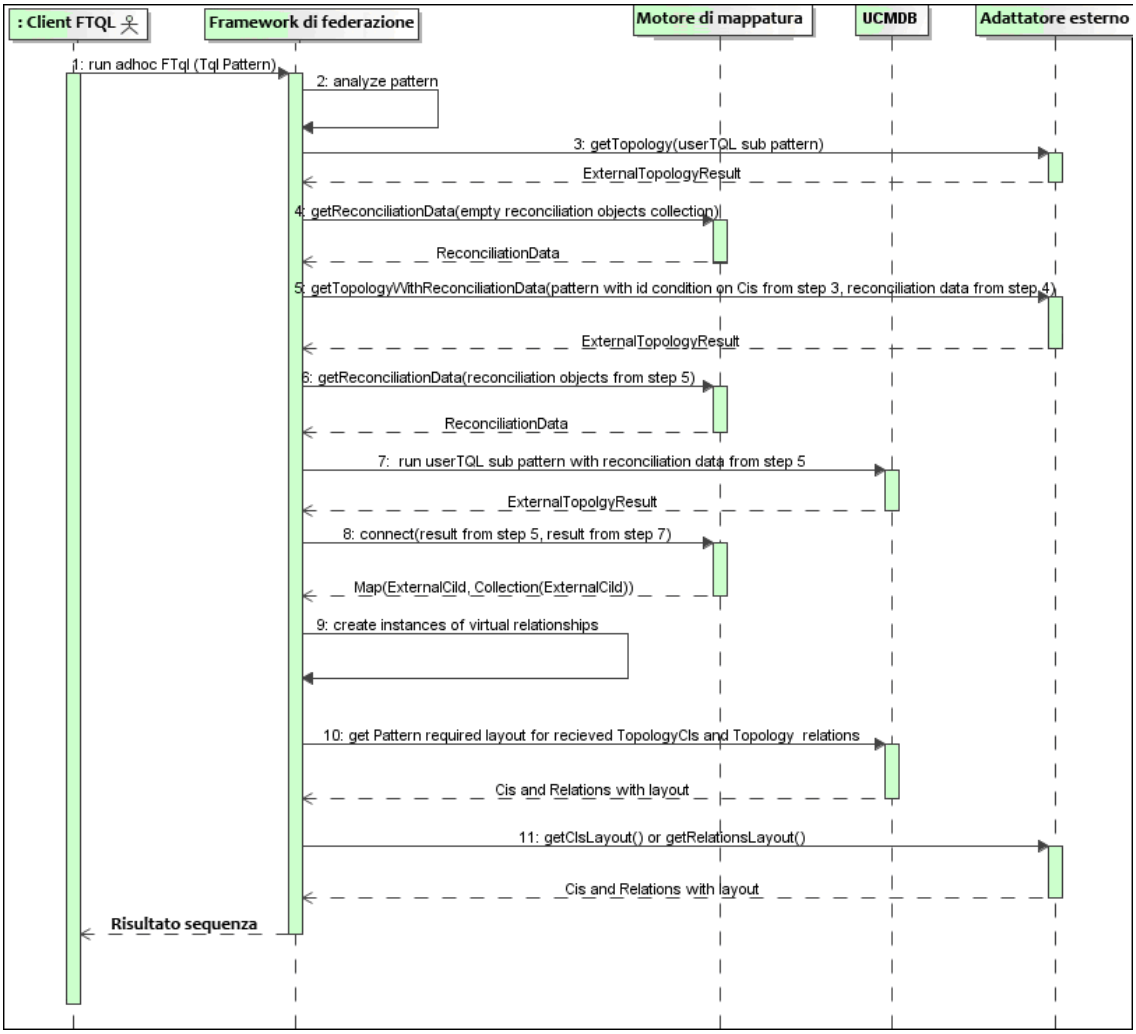


I numeri in questa immagine sono spiegati di seguito:

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.

Numero	Spiegazione
4	Dopo aver ricevuto i risultati della topologia, il framework di federazione chiama il motore di mapping 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" alla pagina precedente da UCMDB.
6	Il framework di federazione chiama il motore di mapping per il recupero dei dati di riconciliazione. In questo stato (in contrasto con il passaggio "3" alla pagina precedente), il motore di mapping riceve gli oggetti di riconciliazione dal passaggio "5" in precedenza come parametri. Il motore di mapping traduce l'oggetto di riconciliazione ricevuto nella condizione dei dati di riconciliazione.
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" in precedenza come un parametro.
8	Il framework di federazione chiama il motore di mapping per eseguire la connessione tra i risultati ricevuti. Il parametro `firstResult` è il risultato della topologia esterna ricevuto da UCMDB nel passaggio "5" in precedenza e il parametro `secondResult` è il risultato della topologia esterna ricevuto dall'adattatore esterno nel passaggio "7" in precedenza. Il motore di mapping restituisce una mappa dove viene mappato l'ID CI eterno dal primo repository di dati (UCMDB in questo caso) rispetto agli ID CI esterni del secondo (esterno) repository di dati.
9	Per ciascun mapping, 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 sul lato dell'adattatore esterno



I numeri in questa immagine sono spiegati di seguito:

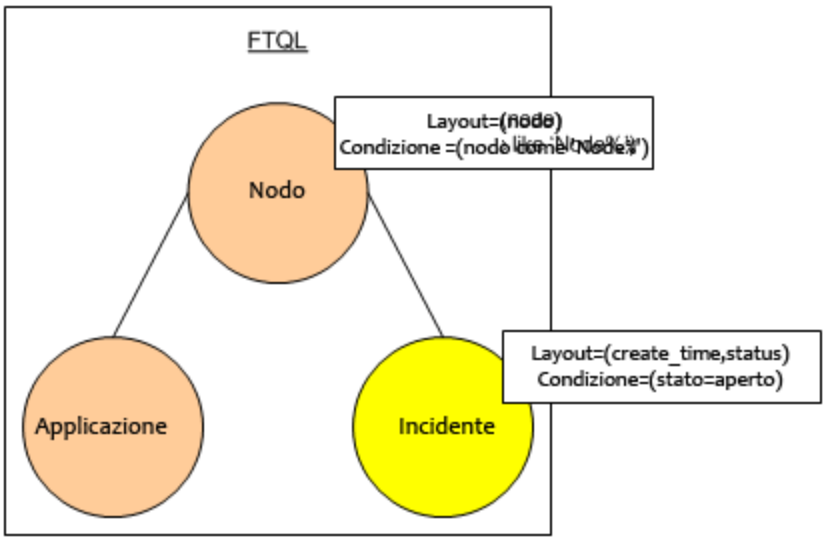
Numero	Spiegazione
1	Il framework di federazione riceve una chiamata per un calcolo della TQL federata.
2	Il framework di federazione analizza l'adattatore, trova la relazione virtuale e suddivide la TQL originale in due sottoadattatori, uno per UCMDB uno per il repository di dati esterno.
3	Il framework 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.

Numero	Spiegazione
4	Dopo aver ricevuto i risultati della topologia, il framework di federazione chiama il motore di mapping 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.
6	Il framework di federazione chiama il motore di mapping per il recupero dei dati di riconciliazione. In questo stato (in contrasto con il passaggio 3), il motore di mapping riceve gli oggetti di riconciliazione dal passaggio 5 come parametri. Il motore di mapping 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 mapping 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 mapping 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 ciascun mapping, 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

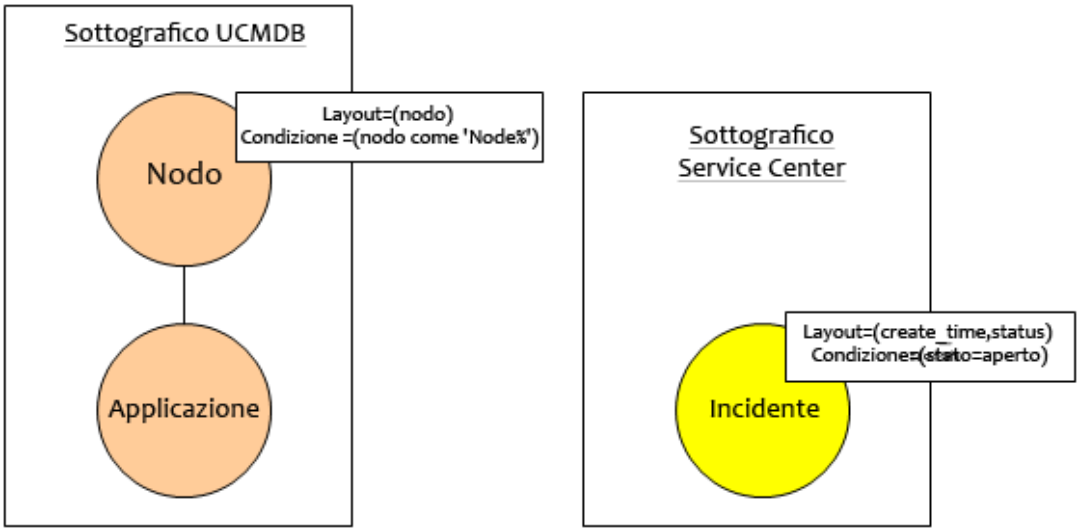
Questo esempio spiega come visualizzare tutti gli incidenti aperti su nodi specifici. ServiceCenter è il repository di dati esterno. Le istanze del nodo sono memorizzate in UCMDB e le istanze degli

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.

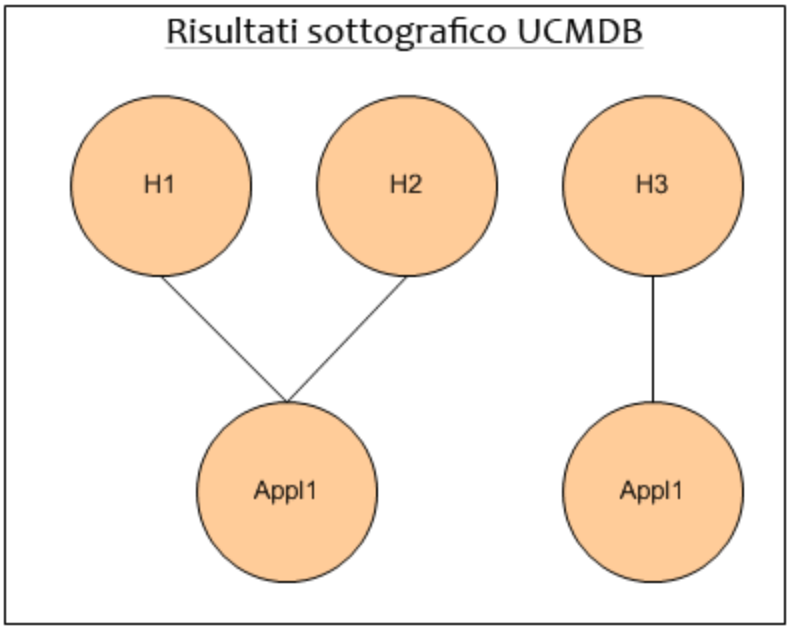


Nota: per la federazione dell'attributo, viene chiamato il metodo **getTopology** dell'adattatore. I dati di riconciliazione sono adattati nella TQL utente (in questo caso, l'elemento CI).

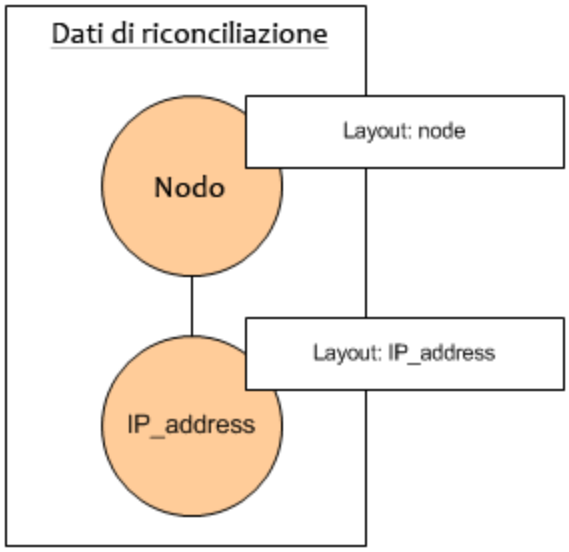
1. Dopo aver analizzato l'adattatore, il framework di federazione riconosce la relazione virtuale tra `Node` e `Incident` e suddivide la query TQL federata in due sottografici:



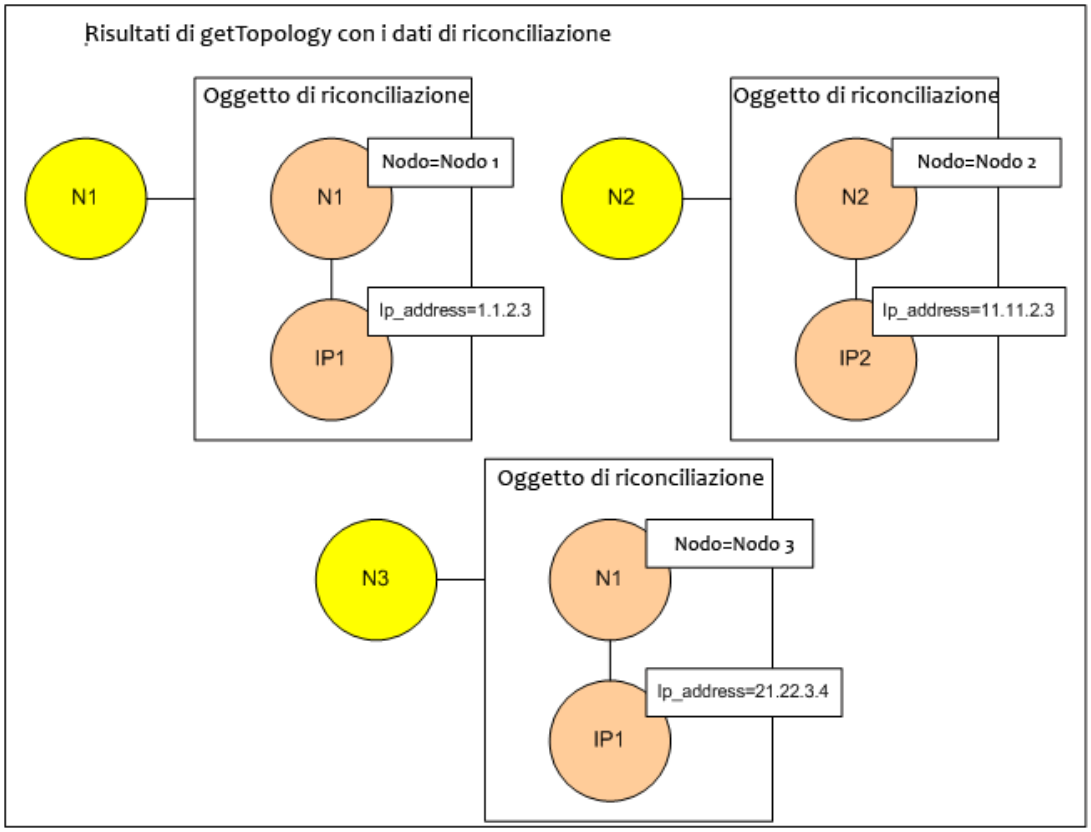
2. Il framework di federazione esegue il sottografico UCMDB per richiedere la topologia e riceve i risultati seguenti:



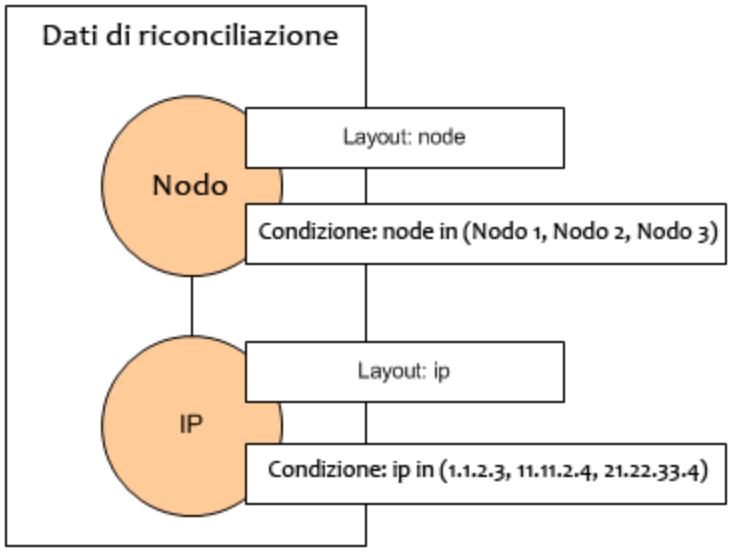
3. Il framework di federazione richiede, dal motore di mapping appropriato, i dati di riconciliazione per il primo repository di dati (UCMDB) che contiene le informazioni necessarie per la connessione tra i dati ricevuti da due repository di dati. I dati di riconciliazione in questo caso sono:



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:

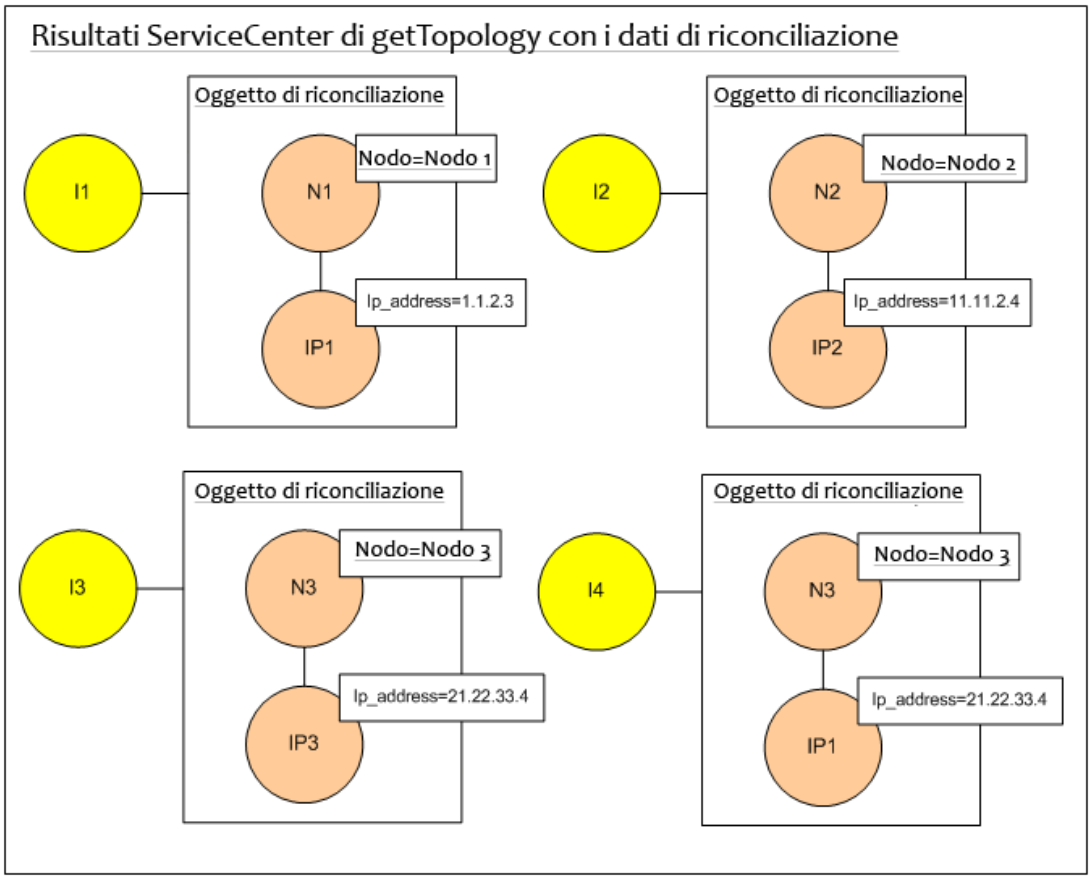


5. I dati di riconciliazione per ServiceCenter devono contenere una condizione per `node` e `ip` derivata dagli oggetti di riconciliazione ricevuti da UCMDB:

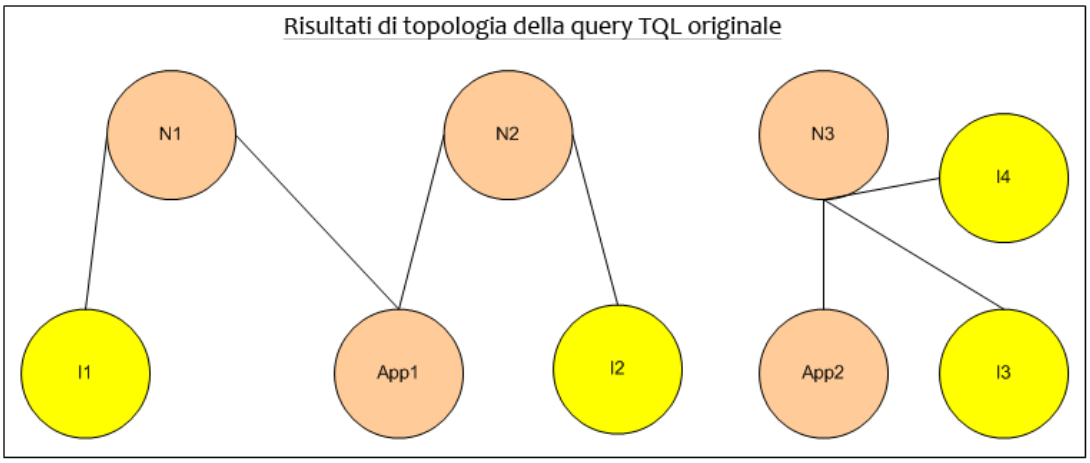


6. Il framework di federazione esegue il sottografico di ServiceCenter con i dati di riconciliazione per richiedere la topologia e gli oggetti di riconciliazione appropriati, ricevendo i risultati

seguenti:



7. Il risultato dopo la connessione nel motore di mapping 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:

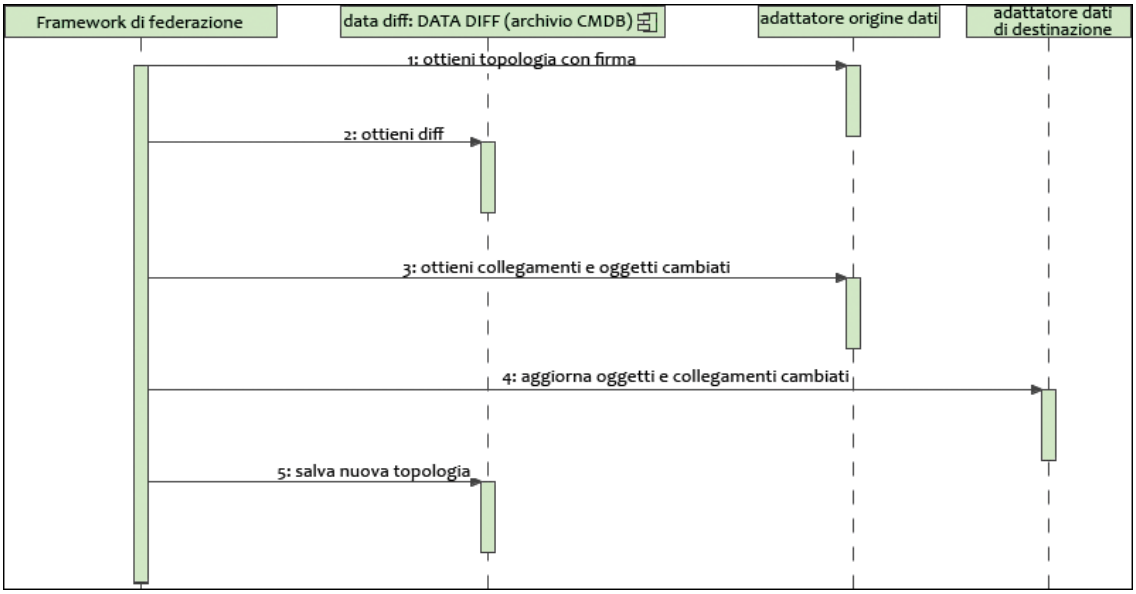
- "Definizioni e termini" nel seguito
- "Diagramma di flusso" nel seguito

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

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 SingleNode** quando l'adattatore non supporta alcuna relazione esterna, ossia l'adattatore non è progettato per ricevere richieste con più di un CI esterno. I dati di riconciliazione necessari per completare l'operazione possono essere descritti come query complessa (vedere SingleNodeFederationTopologyReconciliationAdapter sotto).

  Tutte le interfacce SingleNode sono create per semplificare il flusso di lavoro; per i casi in cui si richiede l'utilizzo di una query più estesa, usare l'interfaccia **FederationTopologyAdapter**.

- Un'**interfaccia FederationTopologyAdapter** è utilizzata per definire gli adattatori che supportano query federate complesse. La richiesta di riconciliazione in questi adattatori è parte del parametro **QueryDefinition**.

  Il motore di federazione utilizza dati di riconciliazione per poter connettere i dati federati ai CI locali corretti. I dati di riconciliazione possono essere recuperati in più di una richiesta (calcolata in modo ricorsivo in base ai risultati). In tal caso, l'adattatore riceve una richiesta solamente con i dati di riconciliazione.

Interfacce SingleNode

Le seguenti interfacce hanno diversi tipi di dati di riconciliazione:

- **SingleNodeFederationIdReconciliationAdapter.** Utilizzare se l'adattatore supporta una **TQL a nodo singolo** e la riconciliazione tra repository di dati viene calcolata dall' ID.

- **SingleNodeFederationPropertyReconciliationAdapter.** Utilizzare se l'adattatore supporta una **TQL a nodo singolo** e la riconciliazione tra repository di dati viene eseguita dalle proprietà

di un CI.

- **SingleNodeFederationTopologyReconciliationAdapter.** Utilizzare se l'adattatore supporta una **TQL a nodo singolo** e la riconciliazione tra repository di dati viene eseguita dalla topologia. L'adattatore deve supportare il caso nel quale l'elemento query è vuoto ed è richiesta solamente la topologia di riconciliazione.

Interfacce adattatore dati

- **FederationTopologyAdapter.** Utilizzare questo adattatore per supportare query TQL federate complesse. Consente la massima diversità. L'adattatore deve supportare il caso nel quale la definizione della query descrive solamente i dati di riconciliazione.
- **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.

Nota: Quando si sviluppa un adattatore che può restituire grandi insiemi di dati, è importante consentire le operazioni sui blocchi implementando l'interfaccia ChunkGetter. Vedere il documento Java dell'adattatore specifico per maggiori informazioni.

Interfacce di report delle risorse

Le interfacce seguenti consentono all'adattatore di segnalare le risorse che possono essere configurate per personalizzare il comportamento dell'adattatore. Queste risorse possono quindi essere modificate direttamente in Studio di integrazione. Queste interfacce devono essere utilizzate in aggiunta alle interfacce adattatore regolari riportate sopra.

- **PopulationQueriesResourcesLocator.** Definisce quali risorse possono essere modificate per ogni query di popolamento specifica.
- **PushQueriesResourceLocator.** Definisce quali risorse possono essere modificate per ogni query di invio dati.
- **GeneralResourcesLocator.** Definisce quali risorse generali possono essere modificate in questo adattatore.

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.

Risorse adattatore debug

Questa attività descrive come utilizzare la JMX Console per creare, visualizzare ed eliminare le risorse di stato dell'adattatore (qualsiasi risorsa creata utilizzando i metodi di manipolazione delle risorse nell'interfaccia DataAdapterEnvironment, salvata nel database di UCMDB o nel database della sonda) per scopi di debug e sviluppo.

1. Avviare il browser Web e specificare l'indirizzo del server come segue:
   - Per il server UCMDB. `http://localhost:8080/jmx-console`
   - Per la sonda: `http://localhost:1977`

   Potrebbe essere necessario effettuare l'accesso con nome utente e password.
2. Per aprire la pagina JMX MBEAN View, eseguire una delle operazioni seguenti:
   - Sul server UCMDB: fare clic su **UCMDB:service=FCMDB Adapter State Resource Services**
   - Sulla sonda: fare clic su **type=AdapterStateResources**
3. Immettere i valori nelle operazioni che si desidera utilizzare, quindi fare clic su **OK**.

Aggiungere un adattatore per una nuova origine dati esterna

Questo compito spiega come definire un adattatore per supportare una nuova origine dati esterna.

Questa attività include le seguenti fasi:

- "Prerequisiti" alla pagina successiva
- "Definire relazioni valide per le relazioni virtuali" alla pagina successiva
- "Definire una configurazione adattatore" a pagina 170
- "Definire le classi supportate" a pagina 174
- "Implementare l'adattatore" a pagina 175
- "Definire le regole di riconciliazione o implementare il motore di mapping" a pagina 175

- "Aggiungere i Jar richiesti per l'implementazione sul percorso classe" a pagina 176
- "Distribuire l'adattatore" a pagina 176
- "Aggiornare l'adattatore" a pagina 176

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 di UCMDB consultare Working with the CI Selector 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 in 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 "Package Manager" in *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. Selezionare **Gestione adattatori**.
   b. Fare clic sul pulsante **Crea nuova risorsa**  e selezionare **Nuovo adattatore**.
   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.cor
e.
 AdapterService">
```

```
<params
className="com.hp.ucmdb.discovery.probe.services.dynamic.cor
e.
 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.BaseDestination
Data">

<!-- 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 pagina 178.

4. **Definire le classi supportate**

Definire le classi supportate, implementando il metodo *getSupportedClasses()* nel codice dell'adattatore o utilizzando il file XML dei pattern.

```
<supported-classes>
    <supported-class name="HistoryChange" is-derived="false" is-reconcili
ation-supported="false" federation-not-supported="false" is-id-reconcilia
tion-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 ereditari

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.

   Se l'adattatore implementa **getTopologyWithReconciliationData** e le capacità dell'adattatore includono la possibilità di essere utilizzato come punto iniziale, l'adattatore deve supportare anche la richiesta della topologia con i dati di riconciliazione senza alcuna condizione (solo tipo). In questo caso l'adattatore deve restituire i dati di riconciliazione completi dei risultati trovati.

   Il supporto alla riconciliazione dell'adattatore può essere definito in base al **global_id**, in tal caso **global_id** deve essere definito come parte degli attributi di riconciliazione nelle classi supportate dall'adattatore. Se il supporto alla riconciliazione dell'adattatore viene definito in base al **global_id**, quindi **getTopologyWithReconciliationData()** deve restituire **global_id** come parte delle proprietà oggetto della riconciliazione. UCMDB utilizza **global_id** per la riconciliazione dei risultati della federazione per un CIT invece che la regola di identificazione.

   Parte dell'API federazione è l'interfaccia DataAdapterEnvironment. Questa interfaccia rappresenta l'ambiente dell'adattatore dati. Contiene l'API dell'ambiente necessaria per il funzionamento dell'adattatore. Per ulteriori informazioni sull'interfaccia DataAdapterEnvironment, vedere "Interfaccia DataAdapterEnvironment" a pagina 180.

6. **Definire le regole di riconciliazione o implementare il motore di mapping**

   Se l'adattatore supporta query TQL federate, sono disponibili due opzioni per definire il motore di mapping:

   - Utilizzare il motore di mapping predefinito, che usa le regole interne di riconciliazione di CMDB per il mapping. Per utilizzarlo, lasciare vuoto il tag XML **<default-mapping-engine>**.
   - Scrivere il motore di mapping implementando l'interfaccia motore di mapping 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

   Distribuire il pacchetto dell'adattatore. Per i dettagli generali sulla distribuzione di un pacchetto consultare "Package Manager" 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 adattatori. I cambiamenti ai file di configurazione nel modulo Gestione adattatori 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 "How to Deploy a Package" in *Guida all'amministrazione di HP Universal CMDB*.

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" nel seguito
- "Caricare il progetto" nel seguito

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. Filtrare alcuni dati sul servizio remoto, facendo sì che l'adattatore selezioni e filtri i rimanenti.

   Nell'esempio MyAdapter, viene utilizzata la logica nell'opzione *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\APIs\DBAdapterFramework_JavaAPI\index.html

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.

	<start-point-adapter>	Specifica che questo adattatore è il punto iniziale per il calcolo delle query TQL.
	<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.
	<push-back-ids>	Restituire l'ID globale del CI alla colonna global_id della tabella (deve essere definito in orm.xml). Il comportamento può essere sostituito implementando il plug-in **FcmdbPluginPushBackIds**.
	<changes-source>	Questo adattatore può essere utilizzato per i flussi dei cambiamenti del popolamento.
	<instance-based-data>	Questo tag specifica che l'adattatore supporta un flusso di popolazione basato su istanze.
	<target>	Questo adattatore può essere utilizzato per i flussi di invio dati.
	<default-mapping-engine>	Consente la definizione di un motore di mapping per l'adattatore (per impostazione predefinita, l'adattatore utilizza il motore di mapping predefinito). Per qualsiasi altro motore di mapping, immettere il nome della classe di implementazione del motore di mapping
	<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.
	<adapter-settings>	L'elenco delle impostazioni dell'adattatore.
	<list.attributes.for.set>	Stabilisce quali attributi sostituiscono il valore precedente (se esistente).

Interfaccia DataAdapterEnvironment

OutputStream openResourceForWriting(String resourceName) throws FileNotFoundException;

Questo metodo apre una risorsa con un nome dato per la scrittura. Viene utilizzato per salvare dati persistenti per l'integrazione. Deve essere utilizzato questo metodo invece di tentare di caricare file utilizzando metodi Java. L'utente deve accertarsi che il flusso sia chiuso quando ha finito l'operazione di scrittura su di esso. close()/flush() salverà la risorsa. Questo metodo crea una risorsa runtime (non può sovrascrivere i file forniti nel pacchetto dell'adattatore).

Parametro

- **resourceName:** nome della risorsa da recuperare. Questo nome deve essere unico in tutte le integrazioni dello stesso adattatore.

Valore restituito

Restituisce un flusso per la scrittura.

Eccezioni

- Questo metodo genera *FileNotFoundException* se il tipo della risorsa è un file e il file non esiste, se la risorsa è una directory invece di un file regolare o se, per altri motivi, la risorsa non può essere aperta per la lettura.
- Questo metodo genera *SecurityException* se esiste un'utilità di Gestione protezione e il suo metodo *checkRead* nega l'accesso al file.

InputStream openResourceForReading(String resourceName) throws FileNotFoundException;

Questo metodo apre una risorsa con un nome dato per la lettura. Viene usato per leggere dati persistenti per l'integrazione. Deve essere utilizzato questo metodo invece di tentare di caricare un file utilizzando metodi Java. L'utente deve assicurare che il flusso venga chiuso a lettura terminata. Tenta innanzitutto di caricare i file forniti nel pacchetto dell'adattatore. Se non trovati, tenta di caricare una risorsa creata in runtime da *DataAdapterEnvironment.openResourceForWriting (String)*. Le risorse runtime possono essere visualizzate utilizzando JMX (rispettivamente della sonda e del server).

Parametro

- **resourceName:** nome della risorsa da recuperare. Questo nome deve essere univoco in tutte le integrazioni dello stesso adattatore.

Valore restituito

Restituisce un flusso in lettura.

Eccezioni

- Questo metodo genera *FileNotFoundException* se il tipo della risorsa è un **file** e il file non esiste, se la risorsa è una directory invece di un file regolare o se, per altri motivi, la risorsa non può essere aperta per la lettura.
- Questo metodo genera *SecurityException* se esiste un'utilità di Gestione protezione e il suo metodo *checkRead* nega l'accesso in lettura al file.

Properties openResourceAsProperties(String propertiesFile) throws IOException;

Questo metodo apre una risorsa con un nome dato e la carica come struttura *Proprietà*. Viene usato per leggere dati persistenti per l'integrazione. Deve essere utilizzato questo metodo invece di tentare di caricare i file **proprietà** utilizzando metodi Java. Tenta innanzitutto di caricare i file forniti nel pacchetto dell'adattatore. Se non trovati, tenta di caricare una risorsa creata in runtime da *DataAdapterEnvironment.openResourceForWriting(String)*. Le risorse runtime possono essere visualizzate utilizzando JMX (rispettivamente della sonda e del server).

Parametro

- **propertiesFile:** nome della risorsa da recuperare. Questo nome deve essere univoco in tutte le integrazioni dello stesso adattatore.

Valore restituito

Restituisce il contenuto del file rappresentato in Proprietà.

Eccezioni

- Questo metodo genera *FileNotFoundException* se il tipo della risorsa è un **file** e il file non esiste, se la risorsa è una directory invece di un file regolare o se, per altri motivi, la risorsa non può essere aperta per la lettura.
- Questo metodo genera *SecurityException* se esiste un'utilità di Gestione protezione e il suo metodo *checkRead* nega l'accesso in lettura al file.
- Questo metodo genera *IOException* se non è stato possibile convertire il file delle proprietà nell'oggetto *Proprietà*.

String openResourceAsString(String resourceName) throws IOException;

Questo metodo apre una risorsa con un nome dato e la carica come stringa. Viene usato per leggere dati persistenti per l'integrazione. Deve essere utilizzato questo metodo invece di tentare di caricare file utilizzando metodi Java.

Tenta innanzitutto di caricare i file forniti nel pacchetto dell'adattatore. Se non trovati, tenta di caricare una risorsa creata in runtime da *DataAdapterEnvironment.openResourceForWriting (String)*. Le risorse runtime possono essere visualizzate utilizzando JMX (rispettivamente della sonda e del server).

Parametro

- **resourceName:** nome della risorsa da recuperare. Questo nome deve essere univoco in tutte le integrazioni dello stesso adattatore.

Valore restituito

Restituisce il contenuto del file rappresentato in formato stringa.

Eccezioni

- Questo metodo genera *FileNotFoundException* se il tipo della risorsa è un **file** e il file non esiste, se la risorsa è una directory invece di un file regolare o se, per altri motivi, la risorsa non può essere aperta per la lettura.
- Questo metodo genera *SecurityException* se esiste un'utilità di Gestione protezione e il suo metodo *checkRead* nega l'accesso in lettura al file.
- Questo metodo genera *IOException* se si verifica un errore I/O.

public void saveResourceFromString(String relativeFileName, String value) throws IOException;

Questo metodo riceve una stringa e la salva come risorsa. Viene utilizzato per salvare dati persistenti per l'integrazione. Deve essere utilizzato questo metodo invece di tentare di salvare file utilizzando metodi Java. Questo metodo converte la stringa in un flusso e la salva nella risorsa. Crea una risorsa runtime, ma non può sovrascrivere i file forniti nel pacchetto dell'adattatore. Le risorse runtime possono essere visualizzate utilizzando JMX (rispettivamente della sonda e del server).

Parametro

- **relativeFileName:** nome della risorsa da recuperare. Questo nome deve essere univoco in tutte le integrazioni dello stesso adattatore.
- **value:** stringa da salvare come risorsa

Eccezioni

Questo metodo genera IOException se si verifica un errore I/O.

boolean resourceExists(String resourceName);

Questo metodo controlla se il nome della risorsa data esiste. Cerca i file forniti nel pacchetto dell'adattatore e le risorse create in runtime da *DataAdapterEnvironment.openResourceForWriting (String)*.

Parametro

- **resourceName:** nome della risorsa da recuperare. Questo nome deve essere univoco in tutte le integrazioni dello stesso adattatore.

Valore restituito

Restituisce **True** se *resourceName* esiste.

boolean deleteResource(String resourceName);

Questo metodo elimina la risorsa specificata dai dati persistenti. Elimina una risorsa di runtime e non può eliminare i file forniti nel pacchetto dell'adattatore. Le risorse runtime possono essere visualizzate utilizzando JMX (rispettivamente per la sonda e per il server).

Parametro

- **resourceName:** nome della risorsa da eliminare. Questo nome deve essere univoco in tutte le integrazioni dello stesso adattatore.

Valore restituito

Restituisce **True** se la risorsa viene correttamente eliminata.

Collection<String> listResourcesInPath(String path);

Questo metodo recupera un elenco di risorse nel percorso specificato. Cerca i file forniti nel pacchetto dell'adattatore e le risorse create in runtime da *DataAdapterEnvironment.openResourceForWriting(String)*. Le risorse runtime possono essere visualizzate utilizzando JMX (rispettivamente per la sonda e per il server).

Parametro

- **path:** percorso della risorsa. Ad esempio, "META-INF/myfiles/"

Valore restituito

Restituisce un elenco di risorse nel percorso.

DataAdapterLogger getLogger();

Recupera il registratore che verrà usato dall'adattatore. Il registratore viene usato per registrare eventi nell'adattatore.

Valore restituito

Restituisce il registratore utilizzato da DataAdapter.

DestinationConfig getDestinationConfig();

Questo metodo recupera la configurazione di destinazione dell'integrazione. Questa configurazione contiene tutte le impostazioni di connessione ed esecuzione per l'integrazione.

Valore restituito

Restituisce il DestinationConfig dell'adattatore.

int getChunkSize();

Questo metodo recupera le dimensioni blocco popolamento richieste per questa integrazione.

Valore restituito

Restituisce le dimensioni blocco popolamento.

int getPushChunkSize();

Questo metodo restituisce le dimensioni blocco invio richieste per questa integrazione.

Valore restituito

Restituisce le dimensioni blocco invio.

ClassModel getLocalClassModel();

Questo metodo recupera un modello classe per leggere informazioni sul modello di classe di UCMDB locale. Questo metodo contiene un ClassModel aggiornato. Una volta che l'oggetto ClassModel viene restituito, non verrà aggiornato per alcun cambiamento del modello di classe. Per poter recuperare un modello di classe aggiornato, utilizzare nuovamente questo metodo per recuperarlo.

Valore restituito

Restituisce il modello di classe di UCMDB.

CustomerInformation getLocalCustomerInformation();

Questo metodo recupera informazioni sul cliente che esegue l'adattatore.

Valore restituito

Restituisce informazioni sul cliente che esegue l'adattatore.

Object getSettingValue(String name);

Questo metodo recupera una specifica impostazione dell'adattatore.

Parametro

name: nome dell'impostazione.

Valore restituito

Restituisce il valore dell'impostazione dell'oggetto.

Map<String, Object> getAllSettings();

Questo metodo recupera tutte le impostazioni dell'adattatore.

Valore restituito

Restituisce le impostazioni dell'adattatore.

boolean isMTEnabled();

Questo metodo controlla se l'ambiente del server supporta la multi-titolarità (MT).

Valore restituito

Restituisce **true** se l'ambiente del server supporta la MT, altrimenti restituisce **false**.

String getUcmdbServerHostName();

Questo metodo restituisce il nome host del server UCMDB locale.

Valore restituito

Restituisce il nome host del server UCMDB locale.

Capitolo 6: Sviluppo degli adattatori Push

Questo capitolo comprende:

Sviluppo e distribuzione degli adattatori Push

Gli adattatori Push generici forniscono una piattaforma comune che consente lo sviluppo rapido di integrazioni che inviano i dati di UCMDB a repository di dati esterni (database e applicazioni di terze parti). Gli adattatori Push generici vengono classificati secondo il protocollo utilizzato per inviare i dati. Per i dettagli sull'invio tramite XML mediante l'adattatore Push XML generico, consultare "Query SQL dell'adattatore Push XML generico" a pagina 200. Per i dettagli sull'invio tramite il servizio Web mediante l'adattatore Push servizio Web generico, consultare "Adattatore Push servizio Web generico" a pagina 200.

Lo sviluppo di un'integrazione personalizzata basata su un adattatore Push generico richiede:

- Creazione di un nuovo pacchetto dell'adattatore dai file modello appropriati dell'adattatore Push generico. Per i dettagli consultare "Creare un pacchetto adattatore" alla pagina successiva.
- Mapping tra i tipi collegamento CI di UCMDB e i dati esterni. I mapping sono conservati come XML e vengono personalizzati per ogni archivio dati esterno. Per i dettagli consultare "Creazione di mapping" a pagina 191.
- Uno script Jython per inviare gli elementi di dati al repository di dati esterno. Per i dettagli consultare "Scrivere gli script Jython" a pagina 194.
- Ulteriori passaggi specifici per l'adattatore. Ad esempio, la scelta del percorso del file da scrivere per l'adattatore Push XML o la creazione di un ricevitore dati per l'adattatore Push servizio Web.

Creare un pacchetto adattatore

Per creare un nuovo adattatore push specifico per MDR, è necessario fare una copia dell'adattatore generico e quindi modificarlo per personalizzarlo come adattatore per un target push specifico.

Pacchetti di adattatori generici possono essere trovati in uno dei due percorsi seguenti:

- Adattatore push XML generico: **hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip**
- Adattatore servizio Web generico: **hp\UCMDB\UCMDBServer\content\adapters\web-service-push-adapter.zip**

Per creare un nuovo adattatore push dall'adattatore push generico:

1. Estrarre il contenuto del file zip del pacchetto selezionato in una cartella di lavoro.
2. Rivedere le directory seguenti come preparazione della fase di ridenominazione e sostituzione.
   - **adapterCode:** contiene la directory distribuita nella directory **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase**. I file jar distribuiti in questa directory non riavviano automaticamente la sonda e non appaiono automaticamente nel CLASSPATH della sonda.
   - **discoveryConfigFiles:** contiene le definizioni dei mapping dell'adattatore e punta allo script Jython corretto (**push.properties**).
   - **discoveryPatterns:** contiene la definizione XML dell'adattatore distribuita nel server UCMDB.
   - **discoveryScripts:** Contiene gli script Jython dell'adattatore attraverso i quali viene stabilita la connessione all'archivio dati di terze parti e vengono inviati i dati
   - **discoveryResources:** contiene il file **UCMDBDataReceiver.jar** contenente le classi di integrazione Java per il servizio Web.

   Nota: Quando viene distribuito questo pacchetto, la sonda viene riavviata per includere questo file **.jar** nel CLASSPATH della sonda. Non è richiesta alcuna azione oltre alla distribuzione del pacchetto.
3. Apportare le modifiche seguenti all'interno della struttura della directory dell'adattatore decompressa:
   a. **discoveryConfigFiles\<Your_Push_Adapter_Name>:** rinominare la directory "PushAdapter" o "XMLtoWebService" con il nome del nuovo adattatore push (ad esempio, “myPushAdapter”).
   b. **discoveryConfigFiles\<Your_Push_Adapter_Name>\push.properties:** nel file

push.properties eseguire le operazioni seguenti:

- Aggiornare il nome di **jythonScript.name** al nome dello script Jython che verrà utilizzato dal nuovo adattatore push (ad esempio, **pushToMyService.py**).
- Aggiornare il nome del file di mapping che verrà utilizzato dal nuovo adattatore push (ad esempio, myPushAdapter_mappings). Non aggiungere l'estensione **.xml**, che verrà inserita automaticamente.

c. **discoveryPatterns\<nome adattatore push>.xml:** rinominare questo file con il nome del nuovo file XML di definizione dell'adattatore (ad esempio, **my_push_adapter.xml**).

d. **discoveryPatterns\<your_push_adapter>.xml:** Aggiornare il file come segue:

- Per l'elemento XML **<pattern>:** impostare gli attributi id e description di conseguenza. Ad esempio:

```
<pattern id="PushAdapter"
xsi:noNamespaceSchemaLocation="../../Patterns.xsd"
description="Discovery Pattern Description" schemaVersion="9.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

viene cambiato in:

```
<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">
```

- Per l'elemento XML **<parameters>:** Aggiornare gli elementi figlio a seconda delle esigenze dell'adattatore. Per impostazione predefinita, per definire un adattatore push vengono utilizzati gli elementi figlio seguenti. Questi valori vengono assegnati quando il punto di integrazione viene definito in Studio di integrazione dopo la configurazione dell'adattatore. Aggiornare l'elenco dei parametri in modo che rifletta gli attributi di connessione richiesti. Non rimuovere l'attributo **probeName**.
  - **host:** nome del server che ospita il servizio Web
  - **port:** porta del servizio ricevitore dati UCMDB in ascolto
  - **Adattatore Push servizio Web: uri** - parte rimanente dell'URL per formare l'indirizzo dell'endpoint servizio del ricevitore dati.
  - **probeName:** definisce in quale Data Flow Probe viene eseguito il processo di invio
- Per l'elemento XML **<integration>:** aggiornare l'elemento figlio **<category>** impostandolo su un valore diverso da Generic. Per impostazione predefinita, gli adattatori che appartengono alla categoria Generic non vengono mostrati in Studio di integrazione. Se si sta eseguendo l'integrazione con un archivio dati di terze parti,

impostare questo valore su "Third Party". Se si sta eseguendo l'integrazione con un prodotto HP BTO , impostare questo valore su "HP BTO Products".

e. **adapterCode\PushAdapter:**rinominare questa cartella con l'ID adattatore utilizzato nel passaggio precedente (ad esempio, **adapterCode\MyPushAdapter**).

f. **discoveryScripts\<your_Jython_push_script>.py:** creare un file con lo stesso nome di quello definito nella proprietà **push.properties jythonScript.name**. Nel file **discoveryScript** vi è uno script che inserisce i CI e i collegamenti a un database Oracle esterno. Sostituire **discoveryScripts\pushScript.py** con lo script scritto (per i dettagli consultare"Scrivere gli script Jython" a pagina 194). Se si rinomina lo script, la proprietà **jythonScript.name** in **adapterCode\<adapterID>\push.properties** deve essere aggiornata di conseguenza.

- Adattatore Push XML: **pushScript.py**
- Adattatore Push servizio Web: **XMLtoWebService.py**

g. **tql\<your_integration_TQLs>:** inserire la definizione XML delle TQL dell'integrazione nella directory specificata, come in un normale pacchetto. Tutte le TQL in questa cartella verranno distribuite durante la distribuzione del pacchetto dell'adattatore.

h. **discoveryConfigFiles\<Your_Push_Adapter_Name>\mappings:** creare un file di mapping XML per le TQL che si desidera utilizzare nell'integrazione. Si noti che l'adattatore push applica le trasformazioni nel file di mapping ai risultati delle TQL di integrazione e quindi invia tali dati in tre parametri (**addResult**, **updateResult** e **deleteResult**) di un compito ad hoc alla Data Flow Probe.

i. **adapterCode\<ID adattatore>\mappings:** sostituire il file **mappings.xml** con i file di mapping preparati (per i dettagli consultare "Creazione di mapping" a pagina 191).

Adattatore Push XML: Questo esempio di mapping corrisponde all'esempio delle tabelle create in ORACLE nel file sql_queries.

Per utilizzare un file di mapping per ciascun metodo TQL, assegnare il nome della 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 mapping specifico per il nome della TQL corrente. Il nome del file di mapping predefinito può essere modificato cambiando la proprietà **mappingFile.default** in **adapterCode\<ID adattatore>\push.properties**.

4. Dopo aver apportato tutte le modifiche indicate, creare un file **.zip** selezionando le cartelle e i file specificati nel precedente passaggio 3 (ad esempio, **my_Push_Adapter.zip**).

5. Distribuire il file **.zip** appena creato al server UCMDB tramite Gestione pacchetti (selezionare **Amministrazione >Gestione pacchetti**).

6. Creare un punto di integrazione in **Gestione flusso di dati > Studio di integrazione** e definire le TQL di integrazione utilizzate dal punto di integrazione. Impostare una pianificazione per l'invio dati automatico.

Risoluzione dei problemi

La procedura per la creazione di un nuovo adattatore push richiede l'esecuzione completa e corretta di operazioni di ridenominazione e sostituzione. Qualsiasi avrà probabilmente ripercussioni sull'adattatore. Il pacchetto deve essere decompresso e ricompresso correttamente per agire come pacchetto UCMDB. Fare riferimento ai pacchetti preconfigurati come esempi. Gli errori più comuni comprendono:

- Includere un'altro directory sopra le directory del pacchetto nel file ZIP.
  Soluzione: comprimere il pacchetto nella stessa directory delle directory del pacchetto, come **discoveryResources**, **adapterCode** e così via. Non includere un altro livello di directory sopra questo livello nel file zip.

- Omettere una ridenominazione critica di una directory, un file o una stringa in un file.
  Soluzione: seguire le istruzioni in questa sezione molto attentamente.

- Inserire un refuso nel nome di una directory, di un file o di una stringa in un file.
  Soluzione: non cambiare le convenzioni di denominazione dopo aver già cominciato la procedura di ridenominazione. Se ci si rende conto di dover cambiare un nome, ricominciare da capo invece di tentare di correggere quanto già fatto, in quanto il rischio di errori è elevato. Utilizzare inoltre procedure di ricerca e sostituzione automatiche invece di sostituire manualmente le stringhe, per ridurre il rischio di errori.

- Distribuire adattatori con nomi file uguali a quelli di altri adattatori, specialmente nelle directory **discoveryResources:** e **adapterCode:**.
  Soluzione: vi è la possibilità che la versione di UCMDB utilizzata abbia un problema noto che impedisce il mapping di file che hanno lo stesso nome di altri adattatori nello stesso ambiente UCMDB. Se si tenta di distribuire un pacchetto con nomi duplicati, la distribuzione non andrà a buon fine. Il problema può verificarsi anche se questi file sono in directory diverse. Può inoltre verificarsi indipendentemente dal fatto che i duplicati siano nello stesso pacchetto o in altri pacchetti distribuiti in precedenza.

A questo punto è possibile creare un nuovo processo adattatore push in Studio di integrazione utilizzando il nuovo adattatore appena distribuito.

Best practice delle TQL per gli adattatori Push

1. Creare una struttura di cartelle nella strutture TQL e Vista e mantenervi tutte le nuove query TQL e viste. Utilizzare una convenzione di denominazione.

2. A meno che la TQL non sia piccola, copiare prima la query TQL più simile.

3. Apportare una modifica alla volta. Salvare, eseguire i test e visualizzare l'anteprima dopo ogni modifica. Ripetere finché i risultati risponderanno ai requisiti.

Creazione di mapping

I dati raw dei risultati TQL hanno il formato dello schema del modello classi UCMDB. È probabile che il consumatore utilizzi un modello dati diverso. L'adattatore Push fornisce un meccanismo di mapping per trasformare i dati in un formato più adatto al consumo. I mapping consentono di eseguire trasformazioni sia dirette che complesse, dalla semplice conversione di tipo denominazione a funzioni di aggregazione e riferimento padre/figlio.

Le specifiche relative al mapping sono disponibili nella sezione "Riferimento per i file di mapping" a pagina 220. Utilizzare questo riferimento per creare un file di mapping.

Nota: Il file delle proprietà dell'adattatore si riferisce al nome del file di mapping. Nei file di configurazione dell'adattatore, l'adattatore implementa una struttura di cartelle utilizzando il nome dell'adattatore. Rinominare questa cartella quando si implementa un adattatore per mantenere l'univocità richiesta da Gestione pacchetti.

Creazione di un file di mapping

Per creare un file di mapping:

1. Cominciare con un file di mapping predefinito.
2. Distribuire l'adattatore ed eseguirlo una volta.
3. Osservare i risultati.
4. Individuare le modifiche da apportare e prenderne nota.
5. Apportare le modifiche individuate nel passaggio precedente. L'elenco seguente può servire come guida per l'ordine delle modifiche.
    a. Iniziare dalla sezione superiore, non trasformativa. Accertarsi che l'adattatore venga eseguito dopo ogni modifica.
    b. Cambiare la sezione dei CI di origine con i nomi UCMDB nel risultato della TQL.
    c. Prima mappare le chiavi.
    d. Quindi aggiungere tutti i mapping diretti.
    e. Aggiungere i mapping complessi.
    f. Aggiungere i mapping dei collegamenti.

Ripetere i passaggi 2-5 finché i dati mappati saranno adatti al consumo. Selezionare il pacchetto dell'adattatore Generic appropriato dal quale creare il nuovo adattatore Push.

I file di mapping funzionano allo stesso modo per tutti i tipi di adattatori Push. L'adattatore Push XML Generic scrive i risultati mappati in un file. L'adattatore Push servizio Web generico invia i

risultati XML a un ricevitore dati. Per ulteriori dettagli consultare "Adattatore Push servizio Web generico" a pagina 200.

Preparare i file di mapping

Nota: È possibile recuperare tutti i CI e le relazioni così come sono in CMDB prive di mappatura, se non si crea il file **mappings.xml**. In questo modo verranno restituiti tutti i CI e tutte le relazioni con tutti i loro attributi.

Esistono due modi diversi per preparare i file di mapping:

- È possibile preparare un file di mapping 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 mapping è denominato **<nome query>.xml**.

Per i dettagli consultare "Schema del file di mapping" a pagina 222.

Questo compito include i passaggi seguenti:

- "Creare un file.XML di mapping" nel seguito
- "Mappare i CI" alla pagina successiva
- "Eseguire il mapping dei collegamenti" a pagina 194

1. ## Creare un file.XML di mapping

   La struttura del file di mapping viene creata nel modo seguente (usare un file esistente come modello):

```
<?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**

   Esistono due modi per mappare i tipi CI CMDB:

   - Mappare i tipi CI in modo che i CI di quel tipo e tutti quelli ereditati vengano mappati nello stesso modo:

```
<source_ci_type_tree name="node" mode="update_else_insert">
    <apioutputseq>1</apioutputseq>
    <target_ci_type  name="host">
        <targetprimarykey>
            <pkey>name</pkey>
        </targetprimarykey
        <target_attribute name=" name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
        <!-- more target attributes --->
    </target_ci_type>
</source_ci_type_tree>
```

   - Mappare un tipo CI in modo che solo i CI di quel tipo vengano elaborati. I CI dei tipi ereditati non vengono elaborati a meno non sia mappato anche il loro tipo (in uno dei due modi):

```
<source_ci_type name="node" mode="update_else_insert">
    <apioutputseq>1</apioutputseq>
    <target_ci_type  name="host">
        <targetprimarykey>
            <pkey>name</pkey>
        </targetprimarykey
        <target_attribute name=" name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
        <!-- more target attributes --->
    </target_ci_type>
</source_ci_type>
```

   Un tipo CI mappato indirettamente (uno dei suoi predecessori è mappato con **source_ci_type_tree**) può anche sovrascrivere la mappa padre poiché appare nel proprio **source_ci_type_tree** o **source_ci_type**.

Quando possibile, si consiglia di utilizzare **source_ci_type_tree**. In caso contrario, i CI risultanti da un tipo CI che non appaiono nei file di mapping non saranno trasferiti allo script Jython.

3. **Eseguire il mapping dei collegamenti**

   Il mapping dei collegamenti può essere eseguito in due modi:

   - Mappare un collegamento in modo che i collegamenti di quel tipo e tutti quelli ereditati vengano mappati nello stesso modo:

```
<source_link_type_tree name="dependency" target_link_type="dependency"
mode="update_else_insert" source_ci_type_end1="webservice" source_ci_t
ype_end2="sap_gateway">
    <target_ci_type_end1 name="webservice" >
    <target_ci_type_end2 name="sap_gateway" >
        <target_attribute name="name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
</source_link_type_tree>
```

   - Mappare un collegamento in modo che vengano elaborati solo i collegamenti di quel tipo. I collegamenti dei tipi ereditati non vengono elaborati a meno non sia mappato anche il loro tipo (in uno dei due modi):

```
<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" >
        <target_attribute name="name" datatype="STRING">
            <map type="direct" source_attribute="name" >
    </target_attribute>
</link>
```

Scrivere gli script Jython

Lo script di mapping è un normale script Jython e deve seguire le regole degli script Jython. Per i dettagli consultare "Sviluppo degli adattatori Jython" a pagina 38.

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. S
ee 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 sulla modifica del file **discoveryPatterns\push_adapter.xml** in "Creare un pacchetto adattatore" a pagina 187), 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 mapping" nel seguito
- "Gestione della verifica connessione nello script" a pagina 198

Utilizzo dei risultati di mapping

Gli adattatori Push generici creano stringhe XML che descrivono i dati da aggiungere, aggiornare o eliminare dal sistema di destinazione. Lo script Jython deve analizzare queste stringhe XML e, successivamente, esegue l'operazione di aggiunta, aggiornamento o eliminazione sulla destinazione.

Nella stringa XML dell'operazione di aggiunta ricevuta dallo script Jython, l'attributo `mamId` per gli oggetti e i collegamenti è sempre l'identificatore UCMDB del collegamento o dell'oggetto originale prima della modifica di tipo, attributo o altre informazioni nello schema del sistema remoto.

Nella stringa XML delle operazioni di aggiornamento o rimozione, l'attributo `mamId` di ciascun oggetto o collegamento contiene la rappresentazione stringa dello stesso `ExternalId` restituito dallo script Jython dalla sincronizzazione precedente.

Nell'XML, l'attributo `id` di un CI contiene `cmdbId` come id esterno, oppure il valore `ExternalId` di quel CI se il CI aveva un valore `ExternalId` al momento dell'invio allo script. I campi `end1Id` e `end2Id` del collegamento conservano per ogni estremità del collegamento il `cmdbId` come un id esterno o `ExternalId` di quell'estremità del collegamento se il CI all'estremità del collegamento ha un `ExternalId` quando inviato allo script.

Durante l'elaborazione dei CI nello script Jython, il valore di ritorno dello script è un mapping tra l'id CMDB del CI e l'id assegnato (l'id assegnato a ogni CI nello script). Se un CI viene inviato per la

prima volta, l'id nell'XML di quel CI è l'id CMDB. Se un CI non viene inviato per la prima volta, l'id CI è lo stesso id dato a quel CI nello script quando è stato inviato per la prima volta.

L'id viene recuperato dallo script XML CI come segue:

1. Dall'elemento CI in XML, recuperare l'id dall'attributo id. Ad esempio: `id = objectElement.getAttributeValue('id')`.

2. Dopo aver recuperato l'id dall'XML, ripristinare l'id dall'attributo (string). Ad esempio: `objectId = CmdbObjectID.Factory.restoreObjectID(id)`.

3. Verificare se `objectId`, ricevuto nel passaggio precedente, sia un id CMDB. Eseguire questa operazione verificando se `objectId` abbia ricevuto il nuovo id dallo script. Se così, l'id restituito non è l'id CMDB. Ad esempio:
   `newId = objectId.getPropertyValue(<il nome dell'attributo id dato dallo script>)`.

   Se `newId` è null, l'id restituito nell'XML è un id CMDB.

4. Se l'id è un id CMDB (ossia, `newId` è null), eseguire quanto segue (se l'id non è un id CMDB, andare al passaggio 5):

   a. Creare una proprietà per quel CI che possiede il nuovo id. Ad esempio: `propArray = [TypesFactory.createProperty('<il nome dell'attributo id dato dallo script>', '<new id>')]`.

   b. Creare un `externalId` a quel CI. Ad esempio:
   ```
   cmdbId = extI.getPropertyValue('internal_id')
   className = extI.getType()
   externalId = ExternalIdFactory.createExternalCiId(className, propArray)
   ```

   c. Mappare l'id CMDB al nuovo `externalId` creato (e nel passaggio successivo restituire il mapping all'adattatore). Ad esempio: `objectMappings.put(cmdbId, externalId)`

   d. Quando tutti i CI e i collegamenti sono mappati:
   ```
   updateResult = DataPushResultsFactory.createDataPushResults
   (objectMappings, linkMappings);
   return updateResult
   ```

5. Se l'id è il nuovo id (ossia, `newId` non è null), `externalId` è il `newId`.

È anche possibile eseguire un report sullo stato di invio per ogni CI e collegamento nel modo seguente:

1. `updateStatus = ReplicationActionDataFactory.createUpdateStatus();`

   dove `updateStatus` è un'istanza della classe `UpdateStatus` che contiene gli stati di CI e collegamenti.

2. Aggiungere uno stato a `updateStatus` chiamando il metodo `reportCIStatus` o `reportRelationStatus`.

Ad esempio:

```
status = ReplicationActionDataFactory.createStatus(Severity.FAILURE, 'Faile
d', ERROR_CODE_CI, errorParams,Action.ADD);

updateStatus.reportCIStatus(externalId, status);
```

Dove `ERROR_CODE_CI` è numero di messaggi di errore come visualizzati nel file **properties.errors** (per i dettagli sul file **properties.errors** consultare "Convenzioni di scrittura errori" a pagina 66) e `errorParams` contiene i parametri da passare al messaggio. Consultare il javadoc **ReplicationActionDataFactory** per maggiori dettagli.

3. Creare un risultato di invio con gli stati nel modo seguente:

```
updateResult = DataPushResultsFactory.createDataPushResults(objectMappings,
linkMappings, updateStatus);

return updateResult
```

Esempio del risultato XML

```
<root>

  <data>

    <objects>

      <Object mode="update_else_insert" name="UCMDB_UNIX" operation="add" ma
mId="0c82f591bc3a584121b0b85efd90b174" id="HiddenRmiDataSource%0Aunix%0A1%0A
internal_id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A">

        <field name="NAME" key="false" datatype="char" length="255">UNIX5</field>

        <field name="DATA_NOTE" key="false" datatype="char" length="255"></field>

      </Object>

    </objects>

    <links>

        <link targetRelationshipClass="TALK" targetParent="unix" targetChild="unix"
operation="add" mode="update_else_insert"
mamId="265e985c6ec51a8543f461b30fa58f81"
id="end1id%5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%3D41372a1
cbcaba27b214b84a2ec9eb535%0A%5D%0Aend2id%
5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%3D0c82f591bc3a584121
b0b85efd90b174%0A%5D%0AHiddenRmi
DataSource%0Atalk%0A1%0Ainternal_id%3DSTRING%3D265e985c6ec51a8543f461b30fa58
f81%0A">

        <field name="DiscoveryID1">41372a1cbcaba27b214b84a2ec9eb535</field>

        <field name="DiscoveryID2">0c82f591bc3a584121b0b85efd90b174</field>
```

```
        <field name="end1Id">HiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%
3D41372a1cbcaba27b214b84a2ec9eb535%0A</field>
        <field name="end2Id">HiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%
3D0c82f591bc3a584121b0b85efd90b174%0A</field>
        <field name="NAME" key="false" datatype="char" length="255">TALK4</field>
        <field name="DATA_NOTE" key="false" datatype="char" length="255"></field>
        </link>
    </links>
  </data>
</root>
```

Nota: Nel caso `datatype="BYTE"`, il valore dei risultati restituiti è **String** generato come: `new String([the byte array attribute])`. Il `byte[] object` può essere ricostruito da: `<the received String>.getBytes()`. Se vi sono differenze nelle impostazioni internazionali predefinite tra il server e la sonda, la ricostruzione viene eseguita secondo le impostazioni del server.

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

Affinché l'adattatore Push supporti la sincronizzazione differenziale, la funzione **DiscoveryMain** deve restituire un oggetto che implementi l'interfaccia **DataPushResults** contenente i mapping tra gli ID che lo script Jython riceve dall'XML e gli ID che lo script Jython crea nel computer remoto. Questi ultimi ID sono di tipo **ExternalId**.

Il comando **ExternalIdUtil.restoreExternal**, che riceve l'ID del CI in CMDB come un parametro, ripristina l'ID esterno dall'ID del CI in CMDB. Questo comando può essere utilizzato, ad esempio, durante la sincronizzazione differenziale e un collegamento viene ricevuto nel punto in cui una delle sue estremità non è nella massa (è stata già sincronizzata).

Se il metodo **DiscoveryMain** nello script Jython sul quale è basato l'adattatore Push restituisce un'istanza **ObjectStateHolderVector** 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 a CMDB durante ciascuna sincronizzazione.

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 pagina 187.

Questo compito consente all'adattatore Push di eseguire la sincronizzazione differenziale.

Lo script Jython restituisce l'oggetto **DataPushResults** contenente due mappe Java, una per il mapping degli ID degli oggetti (le chiavi e i valori sono oggetti di tipo ExternalCiId) e uno per gli ID dei collegamenti (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 ExternalI
dUtil
```

- Utilizzare la classe di valori predefiniti **DataPushResultsFactory** per ottenere l'oggetto **DataPushResults** dalla funzione **DiscoveryMain**.

```
# Create the UpdateResult object

updateResult = DataPushResultsFactory.createDataPushResults(objectMappings, li
nkMappings);
```

- 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):

```
externaCIlId = ExternalIdFactory.createExternalCmdbCiId(ciType, ciIDAsStrin
g)
```

```
externalRelationId = ExternalIdFactory.createExternalCmdbRelationId(linkTyp
e, 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 un mapping 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.

Query SQL dell'adattatore Push XML generico

Nel pacchetto adattatore, il file **sql_queries** , che si trova in **adapterCode > PushAdapter > sqlTablesCreation**, contiene le query necessarie per creare tabelle in un nuovo schema in Oracle per il test dell'adattatore. Le tabelle corrispondono al file adapterCode\<ID adattatore>\mappings\mappings.xml.

Nota: Il file **sql_queries** non è necessario per l'adattatore. Si tratta solo di un esempio.

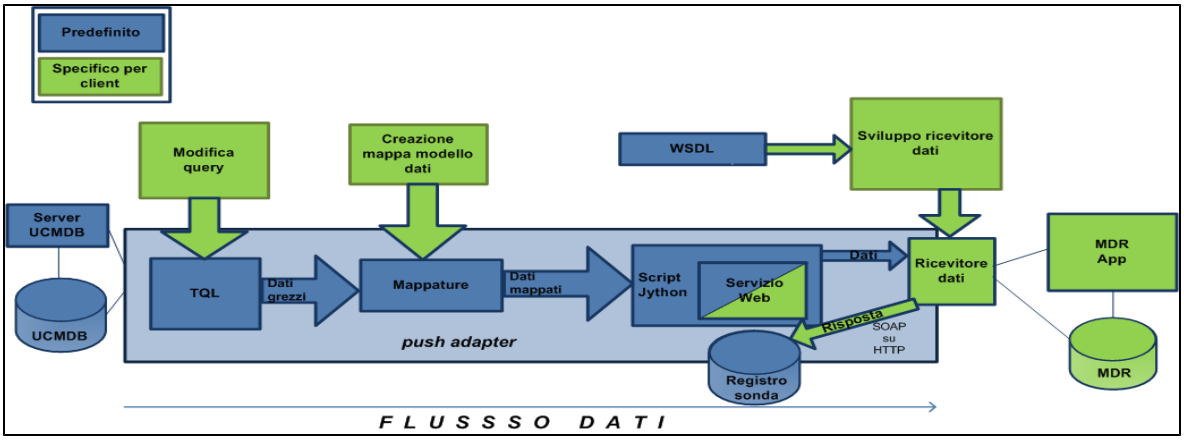
Adattatore Push servizio Web generico

L'adattatore Push servizio Web generico fornisce un invio iniziato da UCMDB di messaggi SOAP contenenti dati di query a un ricevitore dati servizio Web. I risultati mappati sono inviati in messaggi SOAP standard tramite il protocollo HTTP POST al ricevitore dati. Il ricevitore dati deve comprendere i messaggi SOAP prodotti dall'adattatore Push. Per facilitare lo sviluppo di un ricevitore dati corretto, con questo adattatore Push viene fornito un WSDL.

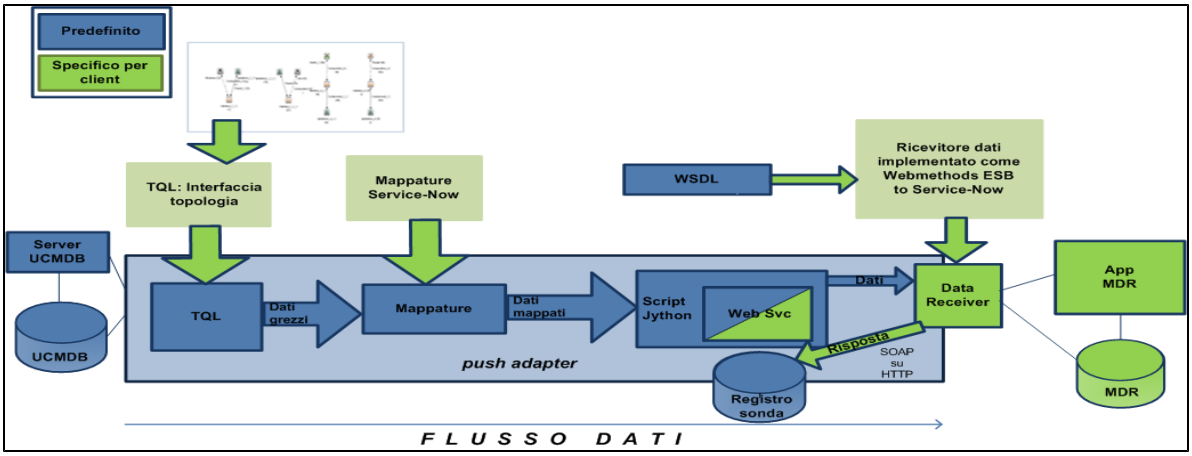
L'elaborazione personalizzata dell'XML di risposta al messaggio SOAP è possibile nello script Jython.

Per comprendere il formato dei dati mappati in entrata, lo sviluppatore del ricevitore dati deve comunicare con lo sviluppatore del file di mapping. Attualmente non è fornito un file **.xsd** con questa versione dell'adattatore Push servizio Web; pertanto i dati devono essere elaborati in modo da tener conto dei dati in entrata, che sono una combinazione della query TQL originale e dei mapping applicati.

Le funzioni dell'adattatore Push servizio Web per l'invio dei dati al client sono mostrate sotto. Gli elementi in verde sono personalizzati o forniti dal cliente per implementare l'adattatore per una destinazione di invio specifica. Gli elementi in blu sono componenti preconfigurati.

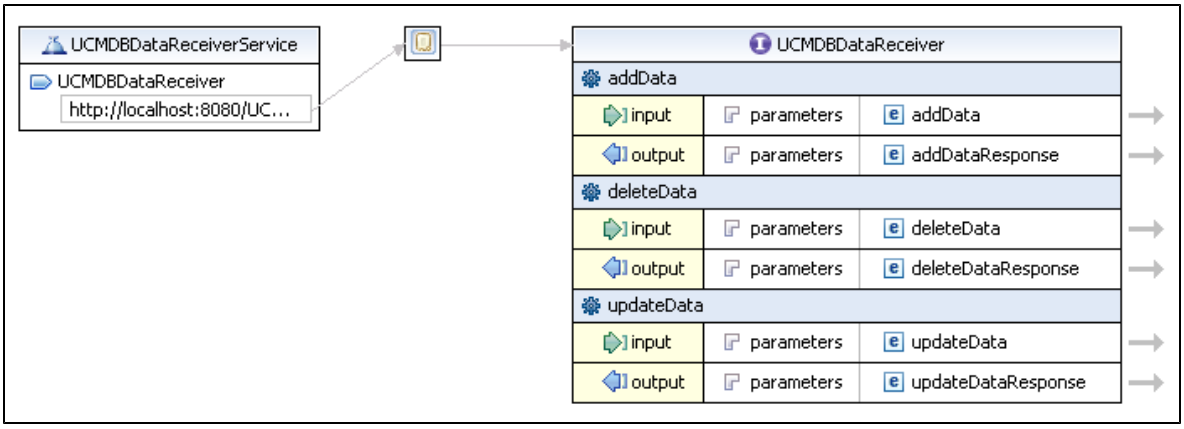


Qui viene mostrato un esempio di implementazione dell'adattatore Push servizio Web generico a un adattatore Push specifico per MDR utilizzando un Enterprise Service Bus (ESB):



WSDL

Allo sviluppatore del cliente viene fornito un WSDL per creare un ricevitore dati in grado di comunicare con l'adattatore Push di UCMDB tramite un servizio Web. Il file **UCMDBDataReceiver.wsdl** descrive i messaggi SOAP utilizzati per inviare dati da UCMDB al ricevitore dati. Il diagramma di progettazione del file WSDL è mostrato qui:



Il ricevitore dati (che in pratica è un server, o “endpoint servizio" nella terminologia SOAP) deve implementare tre metodi: **addData**, **deleteData** e **updateData**, corrispondenti agli insiemi di dati inviati da UCMDB. Le intestazioni HTTP contengono la parola chiave **SoapAction** corretta, che indica il tipo dei dati inviati. Il ricevitore dati è responsabile dell'implementazione della logica di business e dell'elaborazione dei dati.

L'URL predefinito del file WSDL è:

- http://localhost:8080/UCMDBDataReceiver/services/UCMDBDataReceiver?wsdl

Come implementato dal ricevitore dati, l'URL può avere questo aspetto:

- http://testWSPAserver:4444/MyCo.IT.SvcMgt.ws.us:provider/UCMDBDataReceiver?wsdl

L'URL del servizio Web è lo stesso del file WSDL, senza “?wsdl” alla fine.

L'origine del file WSDL è riportata di seguito:

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

<wsdl:definitions targetNamespace="http://ucmdb.hp.com" xmlns:apachesoap="ht
tp://xml.apache.org/xml-soap" xmlns:impl="http://ucmdb.hp.com" xmlns:intf="h
ttp://ucmdb.hp.com" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdl
soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/20
01/XMLSchema">

<!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48
PDT)-->

    <wsdl:types>

        <schema elementFormDefault="qualified" targetNamespace="http://ucmdb
        .hp.com" xmlns="http://www.w3.org/2001/XMLSchema">

            <element name="addData">

                <complexType>

                    <sequence>
```

```
                    <element name="xmlAdded" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="addDataResponse">
                <complexType/>
            </element>
            <element name="deleteData">
                <complexType>
                    <sequence>
                    <element name="xmlDeleted" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="deleteDataResponse">
                <complexType/>
            </element>
            <element name="updateData">
                <complexType>
                    <sequence>
                    <element name="xmlUpdate" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="updateDataResponse">
                <complexType/>
            </element>
        </schema>
    </wsdl:types>

    <wsdl:message name="addDataRequest">
```

```
        <wsdl:part element="impl:addData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="deleteDataResponse">
        <wsdl:part element="impl:deleteDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="updateDataResponse">
        <wsdl:part element="impl:updateDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="deleteDataRequest">
        <wsdl:part element="impl:deleteData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="addDataResponse">
        <wsdl:part element="impl:addDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="updateDataRequest">
        <wsdl:part element="impl:updateData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:portType name="UCMDBDataReceiver">
        <wsdl:operation name="addData">
            <wsdlsoap:operation soapAction="addDataRequest"/>
            <wsdl:input message="impl:addDataRequest" name="addDataRequest">
            </wsdl:input>
            <wsdl:output message="impl:addDataResponse" name="addDataRespons
            e">
            </wsdl:output>
        </wsdl:operation>
```

```
        <wsdl:operation name="deleteData">
            <wsdlsoap:operation soapAction="deleteDataRequest"/>
            <wsdl:input message="impl:deleteDataRequest" name="deleteDataReq
            uest">
            </wsdl:input>
            <wsdl:output message="impl:deleteDataResponse" name="deleteDataR
            esponse">
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="updateData">
            <wsdlsoap:operation soapAction="updateDataRequest"/>
            <wsdl:input message="impl:updateDataRequest" name="updateDataReq
            uest">
            </wsdl:input>
            <wsdl:output message="impl:updateDataResponse" name="updateDataR
            esponse">
            </wsdl:output>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="UCMDBDataReceiverSoapBinding" type="impl:UCMDBDataRe
    ceiver">
        <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap
        .org/soap/http" />
        <wsdl:operation name="addData">
            <wsdl:input name="addDataRequest">
                <wsdlsoap:body use="literal" />
            </wsdl:input>
            <wsdl:output name="addDataResponse">
                <wsdlsoap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="deleteData">
            <wsdl:input name="deleteDataRequest">
```

```
                <wsdlsoap:body use="literal" />
            </wsdl:input>
            <wsdl:output name="deleteDataResponse">
                <wsdlsoap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="updateData">
            <wsdl:input name="updateDataRequest">
                <wsdlsoap:body use="literal" />
            </wsdl:input>
            <wsdl:output name="updateDataResponse">
                <wsdlsoap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="UCMDBDataReceiverService">
        <wsdl:port binding="impl:UCMDBDataReceiverSoapBinding" name="UCMDBDa
        taReceiver">
            <wsdlsoap:address location="http://localhost:8080/UCMDBDataRecei
            ver/services/
            UCMDBDataReceiver"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Gestione delle risposte

Il ricevitore dati deve restituire una stringa nelle strutture **addDataResponse**, **deleteDataResponse** o **updateDataResponse**. L'adattatore passa i dati di risposta non elaborati al file **probeMgr-adaptersDebug.log** della sonda. Il ricevitore può restituire qualsiasi dato stringa e le risposte sono incapsulate in codice XML conforme a SOAP. Nello script Jython è possibile utilizzare la classe **SOAPMessage** e le classi Java correlate per analizzare i messaggi di risposta. Di seguito viene riportato un esempio di messaggio di risposta dal ricevitore dati:

```
<2012-03-16 15:47:38,080> [INFO ] [Thread-110] - XMLtoWebService.py:addData
received response:
```

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<intf:addDataResponse xmlns:intf="http://ucmdb.hp.com">
    <xml>&lt;result&gt;&lt;status&gt;error&lt;/status&gt;
    &lt;message&gt;Error publishing config item changes&lt;/message&gt;
    &lt;/result&gt;</xml>
</intf:addDataResponse>
</soapenv:Body>
```

Il messaggio visualizzato è un messaggio di errore **<Error publishing config item changes>**, ma il contenuto può essere qualsiasi informazione che il ricevitore dati è progettato per fornire come risposta. La risposta è un messaggio di errore semplicemente perché questa è l'intenzione, perché chi l'ha progettata dice che è un messaggio di errore e l'adattatore Push si attende nella risposta un'indicazione di esito positivo o negativo. Il contenuto può consistere in ID di riconciliazione di tutti i CI aggiunti correttamente, oppure in messaggi di errore per CI specifici. La personalizzazione del GWSPA può includere l'analisi del messaggio di risposta e l'avvio di azioni quali il reinvio di determinati CI o l'esecuzione di altre operazioni di registrazione.

Test del WSDL

Per il test dei livelli di un servizio Web durante lo sviluppo è utilizzato il plug-in SOAPUI di Eclipse. È possibile utilizzare SOAPUI per assistere nella personalizzazione di un servizio Web. SOAPUI offre un ambiente di sviluppo integrato (IDE) per il test della creazione, dell'invio e della ricezione dei messaggi SOAP. Dal punto di vista di SOAPUI, il WSDL alle pagine 202-206 genera il messaggio di esempio seguente:

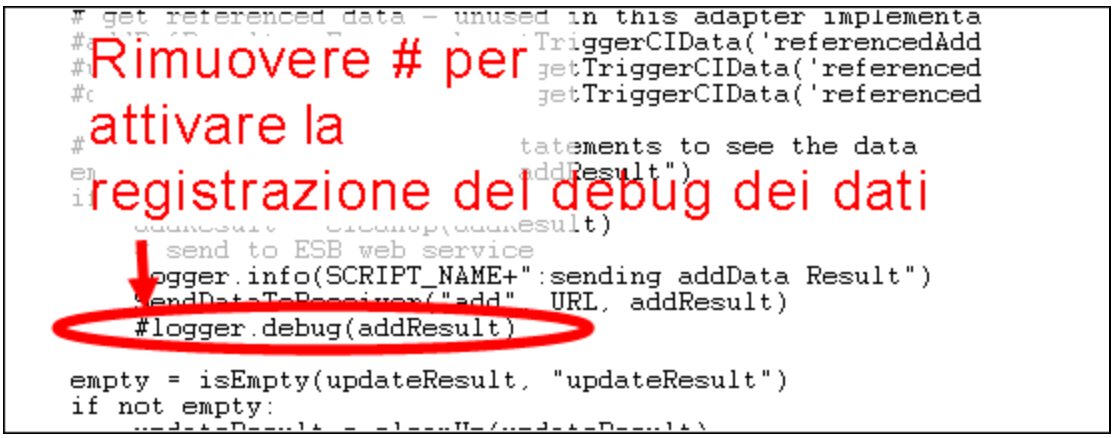
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ucm="http://ucmdb.hp.com">
    <soapenv:Header/>
    <soapenv:Body>
        <ucm:addData>
            <ucm:xmlAdded>?</ucm:xmlAdded>
        </ucm:addData>
    </soapenv:Body>
</soapenv:Envelope>
```

Il "?" nell'elemento **xmlAdded** sopra è la posizione dei dati forniti dall'integrazione dell'adattatore Push servizio Web.

Esame dei risultati

Quando l'adattatore Push funziona normalmente, non in modalità debug, i dati non vengono mai scritti in un file fino alla scrittura del risultato finale (i risultati TQL intermedi e i risultati dei dati mappati non sono normalmente visibili in alcun file registro). Tuttavia, i risultati possono essere

scritti nel file di debug della sonda eliminando il carattere di commento dalle istruzioni **logger.debug** (rimuovere il carattere "#") nella sezione DiscoveryMain, come mostrato di seguito:



Accertarsi che l'istruzione "logger"inizi nella stessa colonna delle istruzioni nelle righe precedenti e successive. In Jython i rientri sono significativi e lo script non funziona se i rientri di tutte le righe non sono corretti.

Nel file registro di debug **probeMgr-adaptersDebug.log** della sonda vi sarà il contenuto dell'output:

```
<2011-12-07 14:02:23,019> [INFO ] [Thread-273] - XMLtoWebService.py started

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - ESB Push parameters:

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - Wshost=harpy.trtc.com

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - WShostport=5555

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - WSuri=ws/DtITServiceManagem
ent.esla.v1.ws.provider:UMDBDataReceiver

<2011-12-07 14:02:23,019> [INFO ] [Thread-273] - URL is http://harpy.trtc.co
m:5555/ws/DtITServiceManagement.esla.v1.ws.
provider:UMDBDataReceiver

<2011-12-07 14:02:23,035> [DEBUG] [Thread-273] - Connected to http://harpy.t
rtc.com:5555/ws/DtITServiceManagement.esla.v1.ws.
provider:UMDBDataReceiver

<2011-12-07 14:02:23,035> [ERROR] [Thread-273] - sending results

<2011-12-07 14:02:23,035> [DEBUG] [Thread-273] - <?xml version="1.0" encodin
g="UTF-8"?>

<root>

    <data>

        <objects>
```

```
<Object mode="" name="u_imp_ip_switch" operation="add" mamId="9e
8c2f6bdfe4b7d0864c79e70833902c">

    <field name="Correlation ID" key="true" datatype="char" len
    gth="">9e8c2f6bdfe4b7d0864c79e70833902c</field>

    <field name="name" key="false" datatype="char" length="">nm
    a_09sw</field>

    <field name="location" key="false" datatype="char" length
    ="" />

    <field name="u_chassis_vendor_type" key="false" datatype="c
    har" length="">ciscoCat2960-24TT</field>

    <field name="serial_number" key="false" datatype="char" len
    gth="" />

    <field name="ram" key="false" datatype="char" length="" />

    <field name="os_version" key="false" datatype="char" length
    ="" />

</Object>
```

Modifica dello script Jython XMLtoWebService.py

Lo script Jython utilizzato dall'adattatore Push servizio Web è molto simile all'adattatore Push XML. Lo script utilizza **UCMDBDataReceiver.jar**, incluso nell'adattatore. Lo script implementa il metodo **SendDataToReceiver()**. **SendDataToReceiver()** utilizza tre parametri:

1. Azione (aggiunta, aggiornamento o eliminazione)
2. L'URL del ricevitore dati
3. I dati

Ad esempio: il blocco di aggiunta sarà: **SendDataToReceiver("add", URL, addResult)**

Tutti i livelli di SOAP e del servizio Web sono incapsulati. L'URL è l'indirizzo endpoint servizio del ricevitore dati UCMDB. È lo stesso URL utilizzato per ottenere il file wsdl tramite il suffisso "?wsdl".

Il sorgente dello script Jython è mostrato sotto. Le righe dove è incapsulata l'integrazione servizio Web sono evidenziate in verde.

```
####################################

# script: XMLtoWebService.py

####################################
```

```
# This jython script accepts TQL data results (adds, updates, and deletes) f
rom the Integration adapter.

# and sends it to a web service. The web service is called UCMDBDataReceiver.

# A web service client of this name must be addressable at the URL provided
by the parameters.

# The SendDataToReceiver.jar exposes the SendDataToReceiver function, as well
as the service locator.

# examples of the service locator are in the testconnection section.

# regular expressions

import re

# logging

import logger

# web service interface

from com.hp.ucmdb import SendDataToReceiver

from com.hp.ucmdb.SendDataToReceiver import locateService

from com.hp.ucmdb.SendDataToReceiver import SendData

################################################

########      VARIABLES             ##########

################################################

SCRIPT_NAME = "XMLtoWebService.py"

logger.info(SCRIPT_NAME+" started")

def cleanUp(str):


    # replace mode=""

    str = re.sub("mode=\"\w+\"\s+", "", str)


    # replace mamId with id

    str = re.sub("\smamId=\"", " id=\"", str)


    # replace empty attributes

    str = re.sub("[\n|\s|\r]*<field name=\"\w+\" datatype=\"\w+\" />", "", s
    tr)
```

```
    # replace targetRelationshipClass with name
    str = re.sub("\stargetRelationshipClass=\"", " name=\"", str)

    # replace Object with object with name
    str = re.sub("<Object mode=\"", "<object mode=\"", str)
    str = re.sub("<Object operation=\"", "<object operation=\"", str)
    str = re.sub("<Object name=\"", "<object name=\"", str)
    str = re.sub("</Object>", "</object>", str)

    # replace field to attribute
    str = re.sub("<field name=\"", "<attribute name=\"", str)
    str = re.sub("</field>", "</attribute>", str)

    #logger.debug("String = %s" % str)
    #logger.debug("cleaned up")

    return str
def isEmpty(xml, type = ""):
    objectsEmpty = 0
    linksEmpty = 0

    m = re.findall("<objects />", xml)
    if m:
        #logger.warn("\t[%s] No objects found" % type)
        objectsEmpty = 1

    m = re.findall("<links />", xml)
    if m:
        #logger.warn("\t[%s] No links found" % type)
        linksEmpty = 1
```

```
    if objectsEmpty and linksEmpty:
        return 1
    return 0
##############################################
########      MAIN                 ##########
##############################################
def DiscoveryMain(Framework):
    #fix this for web service export
    errMsg = "UCMDBDataReceiver Service not found."
    testConnection = Framework.getTriggerCIData("testConnection")
    # Get Web Service Push variables
    WShostName = Framework.getTriggerCIData("Host Name")
    WShostport = Framework.getTriggerCIData("Protocol Port")
    WSuri = Framework.getTriggerCIData("URI")

    logger.info(SCRIPT_NAME+":ESB Push parameters:")
    logger.info("Host Name="+WShostName)
    logger.info("Protocol Port="+WShostport)
    logger.info("URI="+WSuri)
    URL = "http://"+WShostName+":"+WShostport+"/"+WSuri
    logger.info("URL="+URL)
    if testConnection == 'true':
        # locate the service
        test_receiver = SendDataToReceiver()
        locator = test_receiver.locateService(URL)
        #locator = locateService(URL)
        if(locator):
            logger.info(SCRIPT_NAME+":Test connection was successful")
            return
        else:
            raise Exception, errMsg
```

```
        return
# do same thing here if not just a test connection -
receiver = SendDataToReceiver()
locator = receiver.locateService(URL)
if(locator):
    logger.info(SCRIPT_NAME+":Connected to "+URL)
else:
    logger.error(SCRIPT_NAME+":no locator")
    raise Exception, errMsg
    return

# get add/update/delete result objects from the Framework
addResult = Framework.getTriggerCIData('addResult')
updateResult = Framework.getTriggerCIData('updateResult')
deleteResult = Framework.getTriggerCIData('deleteResult')
logger.debug(deleteResult)

# get referenced data - unused in this adapter implementation
#addRefResult = Framework.getTriggerCIData('referencedAddResult')
#updateRefResult = Framework.getTriggerCIData('referencedUpdateResult')
#deleteRefResult = Framework.getTriggerCIData('referencedDeleteResult')
# uncomment out the logger statements to see the data
empty = isEmpty(addResult, "addResult")
if not empty:
    addResult = cleanUp(addResult)
    # send to ESB web service
    logger.info(SCRIPT_NAME+":sending addData Result")
    rcvr = SendDataToReceiver()
    resp = rcvr.SendData("add", URL, addResult)
    logger.info(SCRIPT_NAME+":addData received response:"+resp)
    #logger.debug(addResult)
```

```
    empty = isEmpty(updateResult, "updateResult")
    if not empty:
        updateResult = cleanUp(updateResult)
        # send to ESB web service
        #logger.debug(updateResult)
        logger.info(SCRIPT_NAME+":sending updateData Result")
        rcvr = SendDataToReceiver()
        resp = rcvr.SendData("update", URL, updateResult)
        logger.info(SCRIPT_NAME+":received response:"+resp)


    empty = isEmpty(deleteResult, "deleteResult")
    if not empty:
        deleteResult = cleanUp(deleteResult)
        # send to ESB web service
        #logger.debug(deleteResult)
        logger.info(SCRIPT_NAME+":sending deleteData Result")
        rcvr = SendDataToReceiver()
        resp = rcvr.SendData("delete", URL, deleteResult)
        logger.info(SCRIPT_NAME+":received response:"+resp)
    logger.info(SCRIPT_NAME+" ended")
```

Personalizzazione dell'elaborazione dei messaggi di risposta

Il ricevitore dati deve restituire una stringa contenente qualsiasi risposta o stato desiderati. Per impostazione predefinita, l'adattatore Push servizio Web passa la risposta al registro di livello informativo della sonda. Il messaggio di risposta è codice XML formattato SOAP, contenente la stringa o le stringhe di risposta restituite. Il ricevitore può restituire qualsiasi dato, ad esempio messaggi di esito positivo o negativo, raggruppati o singoli. Se si desidera implementare elaborazioni aggiuntive, la risposta può essere elaborata dallo script Jython dell'adattatore. Non è necessario scrivere codice Java.

Un esempio di messaggio di risposta restituito, inviato con i comandi seguenti:

```
// stub example for building your own UCMDBDataData Receiver
public class UCMDBDataReceiver {
```

```
public String addData (String xmlAdd){
System.out.println(xmlAdd); // do something with the data
// send back a response message based on what you did
String tr = new String("a test response from addData!");
return tr;
}
```

è mostrato qui:

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <addDataResponse xmlns="http://ucmdb.hp.com">
        <addDataReturn>a test response from addData!</addDataReturn>
    </addDataResponse>
</soapenv:Body>
```

Modifica del ricevitore dati

Un client Java può implementare le classi contenute in **UCMDBDataReceiver.jar** e chiamare il servizio Web allo stesso modo di Jython. È inoltre possibile chiamare i metodi non incapsulati. Esiste un Javadoc per le classi di **UCMDBDataReceiver.jar**. Il codice sorgente sottostante mostra come usare questi metodi essenziali per incapsulare i dati in un messaggio SOAP e inviarli al ricevitore su HTTP.

Il processo consiste nel creare un oggetto **UCMDBDataReceiverServiceLocator** e quindi assegnare **UCMDBDataReceiverEndPointAddress** all'URL del ricevitore dati.

Per inviare dati, viene chiamato il metodo **getUCMDBDataReceiver** del localizzatore per creare un oggetto **UCMDBDataReceiver**. L'oggetto **UCMDBDataReceiver** implementa i metodi per inviare effettivamente i dati di aggiunta/modifica/eliminazione. Vi sono tre blocchi di codice identici per elaborare ogni tipo di richiesta.

Di seguito è mostrato il codice sorgente della classe **SendDataToReceiver**. Gli oggetti e i metodi evidenziati sono gli elementi essenziali da utilizzare.

```
/**
* Test SendData for the UCMDB Data Receiver for the UCMDB Web Service Push A
dapter
*/
package com.hp.ucmdb;
import com.hp.ucmdb.SendDataToReceiver;
/**
```

```
* TestSendData can be used to verify the SOAP classes are working.

* TestSendData creates a SendDataToReceiver class and invokes its SendData m
ethod.

* a response String is returned.

* The test URL is typically appended with "?wsdl" to get the WSDL of the ser
vice.

*/

public class TestSendData {

    /**

    * @param args - test SOAP message.

    *  optional arguments [0] a test string [1] a service endpoint URL of a
    Data Receiver.

    * the default URL is sent the incoming argument as a test message.

    * the default URL is "http://localhost:8080/UCMDBDataReceiver/services/U
    CMDBDataReceiver".

    * If any errors are encountered, TestClient will attempt to throw except
    ions.

    */

    public static void main(String[] args) {

        // use test message if supplied, otherwise supply a default test str
        ing

        String teststring = new String("Test SOAP message from UCMDBDataRece
        iver TestSendData.");

        if(args.length > 0) {

            teststring = args[0];

        }

        // use test URL if supplied, otherwise supply the default URL

        String URL = new String("");

        if(args.length > 1) {

            URL = args[1];

        }

        // return response

        String response = new String("");

        // perform the tests
```

```
        try{
            if(URL.equals("")) {
                UCMDBDataReceiverServiceLocator locator = new UCMDBDataRece
                iverServiceLocator();
                UCMDBDataReceiver receiver = locator.getUCMDBDataReceiver()
                ;
                URL = locator.getUCMDBDataReceiverAddress();
                System.out.println("TestClient: tested URL="+locator.getUCM
                DBDataReceiverAddress());
                System.out.println("TestClient: receiver="+receiver.toStrin
                g());
            }
            SendDataToReceiver sdtr = new SendDataToReceiver();
            // this sends a test push and gets a response message
            response = sdtr.SendData("add", URL, args[0]);
            System.out.println("Response received was:"+response);
        } catch(Exception e){
            System.out.println("TestClient: Remote Error:");
            e.printStackTrace();
        }
    }
}
```

Nel file **UCMDBDataReceiver.jar** è incluso anche il codice sorgente per le altre classi:

- TestClient.java
- UCMDBDataReceiver.java
- UCMDBDataReceiverProxy.java
- UCMDBDataReceiverService.java
- UCMDBDataReceiverServiceLocator.java
- UCMDBDataReceiverSoapBindingStub.java

Il codice sorgente è stato generato nell'IDE di Eclipse e quindi modificato. Procedere con cautela quando si modifica il codice di UCMDB, in quanto è stato in gran parte generato-in modo

automatico per assicurare la conformità alle specifiche SOAP e la compatibilità con il ricevitore dati UCMDB.

Javadoc

Con l'adattatore Push servizio Web generico è fornito un **javadoc** completamente commentato. Il **javadoc** è incluso nella cartella di documentazione **javadoc**. Iniziare con **index.html**. La pagina introduttiva consente di accedere alla documentazione per tutte le classi e i metodi contenuti nel SDK.

Tutte le classi

- **SendDataToReceiver:** API per il wrapper servizio Web
- **TestClient:** esegue il test del client per verificare la connettività a un endpoint servizio
- **UCMDBDataReceiver:** wrapper servizio Web

Le restanti sono generato automaticamente dal generatore del servizio Web:

- UCMDBDataReceiverProxy
- UCMDBDataReceiverService
- UCMDBDataReceiverServiceLocator
- UCMDBDataReceiverSoapBindingStub

Panoramica

L'utilizzo di base del SDK, compresi esempi di codice sorgente, è spiegato nella documentazione del pacchetto. Questo **javadoc** riguarda l'adattatore Push servizio Web di UCMDB. L'API può essere chiamata da Jython o da Java.

L'SDK fornisce due esempi di codice sorgente: **TestClient** e **SendDataToReceiver**. **TestClient** implementa un test molto limitato del client locale di risposta. **SendDataToReceiver** è la classe principale utilizzata per inviare dati a un servizio Web.

Utilizzare innanzitutto questo SDK (principalmente il WSDL incluso) per implementare un ricevitore dati UCMDB per comunicare con questo servizio Web. Utilizzare quindi questo SDK per creare un adattatore Push in UCMDB per inviare i dati dei risultati della TQL UCMDB al ricevitore dati. L'utilizzo di base di questa API è descritto sotto, con implementazioni sia Jython che Java.

Implementazione di SendDataToReceiver()

SendDataToReceiver() incapsula tutte le funzioni con un singolo metodo:

- Jython: `SendDataToReceiver("add",yourURL,"Hello!")`
- Java: `SendDataToReceiver("add",yourURL,"Hello!");`

In alternativa, creare un oggetto **SendDataToReceiver** (ad esempio, per manipolare altre impostazioni) e quindi chiamare il metodo **SendData** separatamente, come mostrato qui:

- Jython:

```
rcvr = SendDataToReceiver()
responseMsg = rcvr.SendData("add", yourURL, "Hello!")
```

- Java:

```
SendDataToReceiver rcvr = new SendDataToReceiver();
String responseMsg = rcvr.SendData("add", yourURL, "Hello!");
```

O ancora, se si ha l'esigenza di procedere un passaggio alla volta, è possibile eseguire queste operazioni:

1. Creare un nuovo oggetto x **UCMDBDataReceiverServiceLocator()**, quindi impostare l'indirizzo endpoint dell'oggetto in seguito, come mostrato qui:
   - **Jython:**

     ```
     x = UCMDBDataReceiverServiceLocator()
     x.setUCMDBDataReceiverEndPointAddress(URL)
     ```

   - **Java:**

     ```
     UCMDBDataReceiverServiceLocator x = new UCMDBDataReceiverServiceLocator();
     x. setUCMDBDataReceiverEndPointAddress(URL);
     ```

2. Creare quindi un UCMDBDataReceiver con
   - **Jython:** `y = x.getUCMDBDataReceiver()`
   - **Java:** `UCMDBDataReceiver y = x.getUCMDBDataReceiver();`
3. Inviare infine i dati tramite il servizio Web SOAP, in questo modo:
   - **Jython:**
     - `y.addData(yourData)`
     - or `y.updateData(yourData)`
     - or `y.deleteData(yourData)`
   - **Java:**
     - `y.addData(yourData);`
     - or `y.updateData(yourData);`
     - or `y.deleteData(yourData);`

4. Può essere necessario eseguire un test della connettività e, se l'esito è positivo, riutilizzare lo stesso oggetto localizzatore per restituire **UCDMBDataReceiver** da utilizzare per il trasferimento dati.

Le classi non contengono distruttori e non si occupano di gestire la memoria.

Riferimento per i file di mapping

Utilizzo del mapping

Deve essere creato un mapping per ogni attributo di destinazione nell'output XML trasformato. I mapping specificano dove e come ottenere i dati. Se i dati sono in un altro attributo corrispondente in UCMDB, viene utilizzato un mapping diretto.

Per estrarre dati da più attributi, o attributi dagli attributi dei CI figlio o padre del CI di UCMDB, possono essere necessarie altri mapping complessi. Lo schema di mapping sottostante mostra tutti i mapping possibili.

Il file di mapping è un file XML che definisce quali tipi di CI/relazione in UCMDB vengono mappati ai rispettivi tipi di CI/relazione nell'archivio dati di destinazione. Il formato viene spiegato nel dettaglio più avanti. Il file di mapping controlla quali tipi di CI e relazione vengono inviati e quali attributi vengono inviati esattamente.

Esiste una voce di mapping per ogni attributo che viene inviato all'MDR di destinazione. Ogni voce di mapping può consistere in uno o più attributi nei dati di invio UCMDB raw. Il mapping delle voci permette un controllo completamente granulare della struttura e della denominazione finale dei dati da inviare all'MDR di destinazione.

Mapping diretti

I mapping trasformano un modello di dati in un altro (in questo caso, l'UCMDB nell'MDR di destinazione). Le trasformazioni possono essere semplici; nel caso di una relazione 1:1 tra l'attributo UCMDB e la destinazione, cambia solamente il nome ed eventualmente il tipo.

La maggior parte dei mapping di attributi è diretta. Ad esempio, il nome server “ServerX” può essere rappresentato in UCMDB come un CI di tipo **unix** con nome attributo **primary_server_name**, di tipo **string** con una lunghezza di 50. Il modello dati dell'MDR di destinazione può specificare la stessa entità logica con un CI di tipo **linux**, con nome attributo **hostname**, di tipo **char[]** con una lunghezza massima di 250. I mapping diretti possono realizzare tutti questi tipi di operazioni di traduzione.

Esempio di mapping diretto:

```
 <target_attribute name="dns_domain" datatype="char">

<map type="direct" source_attribute="domain_name" />

</target_attribute>
```

Questo mapping diretto mappa l'attributo UCMDB **dns_domain** all'attributo **domain_name** nel modello dati di destinazione.

Utilizzare il tipo di dato **char** a prescindere dal tipo di dato effettivo, a meno che non sia necessario utilizzare il tipo di dato effettivo.

Mapping complessi

Mapping più complessi consentono ulteriori trasformazioni:

- Mappare valori di attributi di più CI a un unico CI di destinazione.
- Mappare attributi di CI figlio (che hanno una relazione **container_f** o di contenimento) al CI padre nell'archivio dati di destinazione. Ad esempio, impostare un valore chiamato **Numero di CPU** su di un CI host di destinazione. Un altro esempio potrebbe essere l'impostazione del valore **Memoria totale** (aggiungendo i valori di dimensione della memoria di tutti i CI di memoria di un CI host in UCMDB) su di un CI host di destinazione.
- Mappare attributi di CI padre (che hanno una relazione **container_f** o di contenimento) al CI dell'archivio dati di destinazione. Ad esempio, impostare un valore chiamato **Server contenitore** su di un attributo di destinazione chiamato CI **Software installato**, acquisendo il valore dall'host che lo contiene del software CI in UCMDB.

Segue un esempio di mapping complesso, che utilizza due attributi di origine separati da una virgola per creare l'attributo di destinazione **os**:

```
<target_attribute name="os" datatype="char">
    <map type="compoundstring">
        <source_attribute name="discovered_os_name" />
        <constant value="," />
        <source_attribute name="host_osinstalltype" />
    </map>
</target_attribute>
```

Inversione delle direzioni dei collegamenti

È possibile che UCMDB contenga dati che differiscono come struttura a seconda dell'origine. Ad esempio, la relazione tra un CI IpAddress e un CI Interfaccia può essere **padre**, come può verificarsi con l'integrazione di HP Network Node Manager. Oppure può essere un collegamento di **contenimento**, come creato generalmente da Universal Discovery. Inoltre, la direzione di questi collegamenti è opposta l'una all'altra.

Non è attualmente possibile invertire la direzione dei collegamenti nel file di mapping. Se si invertono le variabili **_end1** e **_end2**, viene scambiato l'ordine dei dati nell'XML trasformato, oppure manca il collegamento nei dati di origine.

Una soluzione possibile a questo problema è quella di definire una regola di accrescimento come segue:

1. La parte TQL dell'accrescimento è un sottoinsieme di una TQL utilizzata dall'adattatore push. Questa TQL in particolare seleziona tutti i collegamenti nella direzione opposta a quella desiderata nell'XML trasformato.
2. La parte di accrescimento definisce un nuovo collegamento con la direzione corretta e il tipo desiderato.

3. Viene attivato l'accrescimento, che crea quindi i collegamenti corretti.
4. La TQL del processo di integrazione fa ora riferimento al collegamento accresciuto invece che a quello originale.
5. I mapping <link> nell'adattatore push si riferiscono quindi anch'essi al collegamento accresciuto e producono un insieme di collegamenti coerenti per tipo e direzione.

Schema del file di mapping

Nome e percorso elemento	Descrizione	Attributi
integrazione	Definisce il contenuto di mapping 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.	1. **Nome:** type **Descrizione:** Nome del repository di dati di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** versions **Descrizione:** Versione(i) dei repository di dati di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String 3. **Nome:** vendor **Descrizione:** Fornitore del repository di dati di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String

Nome e percorso elemento	Descrizione	Attributi
target (integration > info)	Definisce le informazioni sul repository di dati di destinazione.	1. **Nome:** type **Descrizione:** Nome del repository di dati di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** versions **Descrizione:** Versione(i) del repository di dati di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String 3. **Nome:** vendor **Descrizione:** Fornitore del repository di dati di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String
targetcis (integration)	Elemento contenitore per tutt i mapping dei CIT.	
source_ci_type_tree (integration > targetcis)	Definisce un CIT di origine e tutti i tipi di CI che erediterà da esso.	1. **Nome:** name **Descrizione:** Nome del CIT di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** mode **Descrizione:** Il tipo di aggiornamento richiesto per il tipo CI corrente. **Obbligatorio?:** Obbligatorio **Tipo:** Una delle stringhe seguenti: a. **insert:** Utilizza questo solo se il CI non è già esistente. b. **update:** Utilizza questo solo se si è certi dell'esistenza del CI. c. **update_else_insert:** Se il CI esiste, aggiornarlo, in caso contrario creare un nuovo CI. d. **ignore:** Non eseguire alcuna operazione con questo tipo CI.

Nome e percorso elemento	Descrizione	Attributi
source_ci_type (integration > targetcis)	Definisce un CIT di origine senza i tipi di CI che erediterà da esso.	1. **Nome:** name **Descrizione:** Nome del CIT di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** mode **Descrizione:** Il tipo di aggiornamento richiesto per il tipo CI corrente. **Obbligatorio?:** Obbligatorio **Tipo:** Una delle stringhe seguenti: a. **insert:** Utilizza questo solo se il CI non è già esistente. b. **update:** Utilizza questo solo se si è certi dell'esistenza del CI. c. **update_else_insert:** Se il CI esiste, aggiornarlo, in caso contrario creare un nuovo CI. d. **ignore:** Non eseguire alcuna operazione con questo tipo CI.
target_ci_type (integration > targetcis > source_ci_type -OR- integration > targetcis > source_ci_type_ tree)	Definisce un CIT di destinazione.	1. **Nome:** name **Descrizione:** Nome del tipo CI di destinazione. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** schema **Descrizione:** Il nome dello schema che verrà utilizzato per archiviare questo tipo CI sulla destinazione. **Obbligatorio?:** Non obbligatorio **Tipo:** String 3. **Nome:** namespace **Descrizione:** Indica lo spazio dei nomi di questo tipo CI sulla destinazione. **Obbligatorio?:** Non obbligatorio **Tipo:** String

Nome e percorso elemento	Descrizione	Attributi
targetprimarykey (integration > targetcis > source_ci_type) -OR- (integration > targetcis > source_ci_type_tree -OR- (integration > targetrelations > link) -OR- (integration > targetrelations > source_link_type_tree)	Identifica gli attributi chiave principali del CIT di destinazione.	
pkey (integration > targetcis > source_ci_type > targetprimarykey -OR- integration > targetcis > source_ci_type_tree > targetprimarykey -OR- (integration > targetrelations > link > targetprimarykey) -OR- integration > targetrelations > source_link_type_tree > targetprimarykey)	Identifica un attributo chiave principale. Richiesto solo se la modalità è **update** o **insert_else_update**.	

Nome e percorso elemento	Descrizione	Attributi
target_attribute (integration > targetcis > source_ci_type -OR- integration > targetcis > source_ci_type_tree -OR- integration > targetrelations > link -OR- integration > targetrelations > source_link_type_tree)	Definisce l'attributo del CIT di destinazione.	1. **Nome:** name **Descrizione:** Nome dell'attributo del CIT di destinazione. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** datatype **Descrizione:** Tipo dati dell'attributo del CIT di destinazione. **Obbligatorio?:** Obbligatorio **Tipo:** String 3. **Nome:** length **Descrizione:** Per i tipi di dati stringa/caratteri, dimensione intera dell'attributo di destinazione. **Obbligatorio?:** Non obbligatorio **Tipo.** Integer 4. **Nome.** option **Descrizione.** La funzione di conversione da applicare al valore. **Obbligatorietà.** False **Tipo.** Una delle stringhe seguenti: a. **uppercase** – converte in maiuscolo b. **lowercase** – converte in minuscolo Se questo attributo è vuoto, non verrà applicata alcuna funzione di conversione.

Nome e percorso elemento	Descrizione	Attributi
map (integration > targetcis > source_ci_type > target_attribute -OR- integration > targetcis > source_ci_type_tree > target_attribute) -OR- (integration > targetrelations > link > target_attribute -OR- integration > targetrelations > source_link_type_tree > target_attribute)	Specifica come ottenere il valore dell'attributo del CIT di origine.	1. **Nome.** type **Descrizione.** Il tipo di mapping tra i valori di origine e quelli di destinazione. **Obbligatorietà.** Obbligatorio **Tipo.** Una delle stringhe seguenti: a. **direct** – specifica un mapping 1-to-1 dal valore dell'attributo di origine al valore dell'attributo di destinazione. b. **compoundstring –** i sottoelementi vengono uniti in una stringa singola e viene impostato il valore dell'attributo di destinazione. c. **childattr** – i sottoelementi sono uno o più attributi del CIT figlio. I CIT figlio sono definiti come quelli con relazione **composition** o **containment**. d. **constant** – stringa statica 2. **Nome.** value **Descrizione.** Constant string for type=**constant** **Obbligatorietà.** Richiesto solo quando tipo=**constant** **Tipo.** String 3. **Nome.** attr **Descrizione.** Nome attributo di origine per tipo=**direct** **Obbligatorietà.** Richiesto solo quando tipo=**direct** **Tipo.** String

Nome e percorso elemento	Descrizione	Attributi
aggregation (integration > targetcis > source_ci_type > target_attribute > map -OR- integration > targetcis > source_ci_type_tree > target_attribute > map -OR- (integration > targetrelations > link > target_attribute > map -OR- integration > targetrelations > source_link_type_tree > 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 **Obbligatorio?:**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 la somma di tutti i valori numerici 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.

Nome e percorso elemento	Descrizione	Attributi
source_child_ci_type (integration > targetcis > source_ci_type > target_attribute > map -OR- integration > targetcis > source_ci_type_tree > target_attribute > map -OR- (integration > targetrelations > link > target_attribute > map -OR- integration > targetrelations > source_link_type_tree > target_attribute > map) Valido solo quando il tipo di mappa è **childattr.**	Specifica da quale CI connesso viene preso l'attributo figlio.	1. **Nome.** name **Descrizione.** Il tipo del CI figlio **Obbligatorietà.** Obbligatorio **Tipo.** String 2. **Name.** source_attribute **Descrizione.** L'attributo del CI figlio mappato. **Obbligatorietà.** Richiesto solo se il tipo di aggregazione **childAttr** (che è sullo stesso percorso) non è =count. **Tipo.** String

Nome e percorso elemento	Descrizione	Attributi
validation (integration > targetcis > source_ci_type > target_attribute > map -OR- integration > targetcis > source_ci_type_tree > target_attribute > map -OR- (integration > targetrelations > link > target_attribute > map -OR- integration > targetrelations > source_link_type_tree > target_attribute > map) Valido solo quando il tipo di mappa è **childatt**	Consente il filtro di esclusione dei CI figlio del CI di origine in base ai valori degli attributi. Utilizzato con il sottoelemento di aggregazione per raggiungere la granularità esatta di quali attributi figlio vengono mappati sul valore attributo del CIT di destinazione. Facoltativo.	1. **Nome.** minlength **Descrizione.** Esclude le stringhe più brevi rispetto al valore fornito. **Obbligatorio?:** Non obbligatorio **Tipo.** Integer 2. **Nome.** maxlength **Descrizione.** Esclude le stringhe più lunghe rispetto al valore fornito. **Obbligatorio?:** Non obbligatorio **Tipo.** Integer 3. **Nome.** minvalue **Descrizione.** Esclude i numeri più piccoli rispetto al valore specificato. **Obbligatorio?:** Non obbligatorio **Tipo.** Numeric 4. **Nome.** maxvalue **Descrizione.** Esclude i numeri più grandi rispetto al valore specificato. **Obbligatorio?:** Non obbligatorio **Tipo.** Numeric
targetrelations (integration)	Elemento contenitore per tutti i mapping delle relazioni. Facoltativo.	

Nome e percorso elemento	Descrizione	Attributi
source_link_type_tree (integration > targetrelations)	Esegue la mappatura di un tipo di relazione di origine senza che i tipi ereditino una relazione di destinazione. Obbligatorio solo se **targetrelation** è presente.	1. **Nome:** name **Descrizione.** Nome relazione di origine. **Obbligatorio?:**Obbligatorio **Tipo.** String 2. **Nome:** target_link_type **Descrizione.** Nome relazione di destinazione **Obbligatorio?:** Obbligatorio **Tipo.** String 3. **Nome:** nameSpace **Descrizione:** Lo spazio dei nomi per il collegamento che verrà creato sulla destinazione. **Obbligatorio?:** Non obbligatorio **Tipo:** String 4. **Nome:** mode **Descrizione:** Il tipo di aggiornamento richiesto per il collegamento corrente. **Obbligatorio?:** 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. 5. **Nome:** source_ci_type_end1 **Descrizione:** Tipo CI End1 della relazione di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String 6. **Nome:** source_ci_type_end2 **Descrizione:** Tipo CI End1 della relazione di origine. **Obbligatorio?:** Obbligatorio

Nome e percorso elemento	Descrizione	Attributi
		Tipo: String

Nome e percorso elemento	Descrizione	Attributi
link (integration > targetrelations)	Esegue la mappatura di una relazione di origine su una relazione di destinazione. Obbligatorio solo se **targetrelation** è presente.	1. **Nome:** source_link_type **Descrizione:** Nome relazione di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** target_link_type **Descrizione:** Nome relazione di destinazione. **Obbligatorio?:** Obbligatorio **Tipo:** String 3. **Nome:** nameSpace **Descrizione:** Lo spazio dei nomi per il collegamento che verrà creato sulla destinazione. **Obbligatorio?:** Non obbligatorio **Tipo:** String 4. **Nome:** mode **Descrizione:** Il tipo di aggiornamento richiesto per il collegamento corrente. **Obbligatorio?:** 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. 5. **Nome:** source_ci_type_end1 **Descrizione:** Tipo CI End1 della relazione di origine **Obbligatorio?:** Obbligatorio **Tipo:** String 6. **Nome:** source_ci_type_end2 **Descrizione:** Tipo CI End1 della relazione di origine **Obbligatorio?:** Obbligatorio

Nome e percorso elemento	Descrizione	Attributi
		Tipo: String
target_ci_type_end1 (integration > targetrelations > link -OR- integration > targetrelations > source_link_type_tree)	Tipo CI End1 della relazione di destinazione.	1. **Nome:** name **Descrizione:** Nome del tipo CI End1 della relazione di destinazione. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** superclass **Descrizione:** Nome della superclasse del tipo CI End1. **Obbligatorio?:** Non obbligatorio **Tipo:** String
target_ci_type_end2 (integration > targetrelations > link -OR- integration > targetrelations > source_link_type_tree)	Tipo CI End2 della relazione di destinazione.	1. **Nome:** nome **Descrizione:** Nome del tipo CI End2 della relazione di destinazione. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** superclass **Descrizione:** Nome della superclasse del tipo CI End2. **Obbligatorio?:** Non obbligatorio **Tipo:** String

Schema dei risultati di mapping

Nome e percorso elemento	Descrizione	Attributi
root	La radice del documento dei risultati.	
data (root)	La radice dei dati stessi.	
objects (root > data)	L'elemento root per gli oggetti da aggiornare.	

Nome e percorso elemento	Descrizione	Attributi
Object (root > data > objects)	Descrive l'operazione di aggiornamento per un singolo oggetto e tutti i suoi attributi.	1. **Nome:** name **Descrizione:** Nome del Tipo CI **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** mode **Descrizione:** Il tipo di aggiornamento richiesto per il tipo CI corrente. **Obbligatorio?:** Obbligatorio **Tipo:** Una delle stringhe seguenti: a. **insert** – utilizza questo solo se il CI non è già esistente. b. **update** – utilizza questo solo se si è certi dell'esistenza del CI. c. **update_else_insert** – se il CI esiste, aggiornarlo, in caso contrario creare un nuovo CI. d. **ignore** – non eseguire alcuna operazione con questo tipo CI. 3. **Nome:** operation **Descrizione:** L'operazione da eseguire con questo CI. **Obbligatorio?:** Obbligatorio **Tipo:** Una delle stringhe seguenti: a. **add** – il CI deve essere aggiunto b. **update** – il CI deve essere aggiornato c. **delete** – il CI deve essere eliminato Se non è impostato alcun valore, viene utilizzato il valore predefinito di **add**. 4. **Nome:** mamId **Descrizione:** L'ID dell'oggetto nel CMDB di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String

Nome e percorso elemento	Descrizione	Attributi
field (root > data > objects > Object -OR- root > data > links > link)	Descrive il valore di un singolo campo 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. Ogni ID estremità appare come un oggetto (sotto <objects>).	1. **Nome:** name **Descrizione:** Nome del campo. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** key **Descrizione:** Specifica se il campo è una chiave per l'oggetto. **Obbligatorio?:** Obbligatorio **Tipo:** Boolean 3. **Nome:** datatype **Descrizione:** Il tipo di campo. **Obbligatorio?:** Obbligatorio **Tipo:** String 4. **Nome:** length **Descrizione:** Per tipi di dati stringa/carattere, questa è la dimensione intera dell'attributo di destinazione. **Obbligatorio?:** Non obbligatorio **Tipo:** Integer

Nome e percorso elemento	Descrizione	Attributi
links (root > data)	L'elemento root per i collegamenti da aggiornare.	1. **Nome:** targetRelationshipClass **Descrizione:** Il nome della relazione (collegamento) nel sistema di destinazione. **Obbligatorio?:** Obbligatorio **Tipo:** String 2. **Nome:** targetParent **Descrizione:** Il tipo della prima estremità del collegamento (padre). **Obbligatorio?:** Obbligatorio **Tipo:** String 3. **Nome:** targetChild **Descrizione:** Il tipo della seconda estremità del collegamento (figlio). **Obbligatorio?:** Obbligatorio **Tipo:** String 4. **Nome:** mode **Descrizione:** Il tipo di aggiornamento richiesto per il tipo CI corrente. **Obbligatorio?:** Obbligatorio **Tipo:** Una delle stringhe seguenti: a. **insert** – utilizza questo solo se il CI non è già esistente. b. **update** – utilizza questo solo se si è certi dell'esistenza del CI. c. **update_else_insert** – se il CI esiste, aggiornarlo, in caso contrario creare un nuovo CI. d. **ignore** – non eseguire alcuna operazione con questo tipo CI. 5. **Nome:** operation **Descrizione:** L'operazione da eseguire con questo CI. **Obbligatorio?:** Obbligatorio **Tipo:** Una delle stringhe seguenti: a. add – il CI deve essere aggiunto b. update – il CI deve essere aggiornato c. delete – il CI deve essere eliminato Se non è impostato alcun valore, viene utilizzato il valore predefinito di **add**. 6. **Nome:** mamId

Nome e percorso elemento	Descrizione	Attributi
		Descrizione: L'ID dell'oggetto nel CMDB di origine. **Obbligatorio?:** Obbligatorio **Tipo:** String

Personalizzazione

Questa sezione spiega alcune delle procedure base per i tipi comuni di personalizzazione per gli adattatori push.

Aggiunta di un attributo

1. Assicurarsi che l'attributo sia incluso nel risultato TQL.
2. Aggiungere il mapping dell'attributo al file di mapping nella sezione di mapping CI corretta.
3. Assicurarsi che il ricevitore dati sia preparato per ricevere l'attributo aggiuntivo nei dati.

Rimozione di un attributo

Per rimuovere un attributo, rimuoverlo dal file di mapping. È necessario rimuoverlo anche dalla query TQL se non è più utilizzato nel risultato o come nodo condizionale.

Aggiunta di un tipo CI

1. Aggiungere il tipo CI alla query TQL.
2. Assicurarsi che il tipo CI e i dati dell'attributo siano visualizzati nel risultato TQL (utilizzare il calcolo e l'anteprima).
3. Aggiungere il mapping del tipo CI nel file di mapping. Copiare il mapping di un altro tipo CI per creare rapidamente un nuovo tipo CI.
4. Modificare il mapping di nome e attributi dell'XML copiato in modo che corrisponda al nuovo tipo CI e ai relativi attributi. Consultare "Riferimento per i file di mapping" a pagina 220 per i tipi di mapping disponibili.

Rimozione di un tipo CI

1. Rimuovere il tipo CI dalla query TQL
2. Rimuovere la sezione di mapping per quel tipo CI nel file di mapping.

Aggiunta di collegamenti

1. Assicurarsi che i due CI finali siano presenti nei dati.
2. Assicurarsi che il collegamento da aggiungere sia valido (controllare in Gestione tipi CI).

3. Aggiungere gli elementi del collegamento nella sezione delle relazioni dell'XML di mapping.

Rimozione di collegamenti

1. Rimuovere la sezione del collegamento che si desidera rimuovere nel file di mapping.
2. Se possibile, rimuovere il collegamento dalla query TQL (a meno che non comprometta l'efficienza o la funzionalità della query).

Capitolo 7: Sviluppo degli adattatori Push generici estesi

Questo capitolo comprende:

Panoramica

Un adattatore Push esteso è una struttura di dati che rappresenta il risultato della query TQL. Ogni adattatore costruito sull'adattatore Push esteso gestirà questa struttura di dati e la invierà alla destinazione richiesta.

La struttura di dati è denominata **ResultTreeNode (RTN)**. L'RTN viene creato secondo il file di mapping dell'adattatore e i risultati della query TQL. Le query utilizzate per l'adattatore Push esteso devono basarsi sulla radice, ossia deve contenere un nodo query con il nome elemento **root** oppure uno o più elementi relazione che iniziano con il prefisso **root**. Questo CI o relazione serve come elemento radice della query. Per i dettagli consultare Data Push nella *Guida di Gestione flusso di dati di HP Universal CMDB*.

Per lo sviluppo degli adattatori Push avanzati sono previsti due passaggi base:

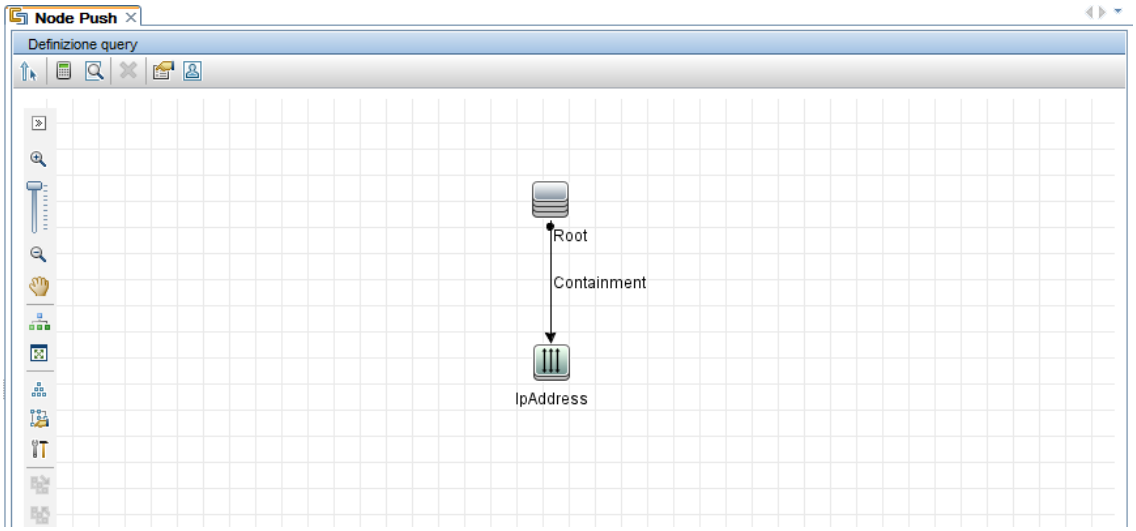
1. Implementare l'interfaccia PushConnector – questa interfaccia riceve i dati da aggiungere, aggiornare ed eliminare sotto forma di elenco di RTN, quindi esegue l'invio di dati nella destinazione.

2. Creare il file di mapping – il file di mapping determina la creazione della struttura RTN mediante la mappatura dei CI e degli attributi del risultato TQL.

Il file di mapping

Nel seguente esempio viene descritto come creare il file di mapping:

In questo esempio si simulerà l'invio di un nodo e di un indirizzo IP. Creeremo una query TQL denominata: **Invio nodo**, come segue:



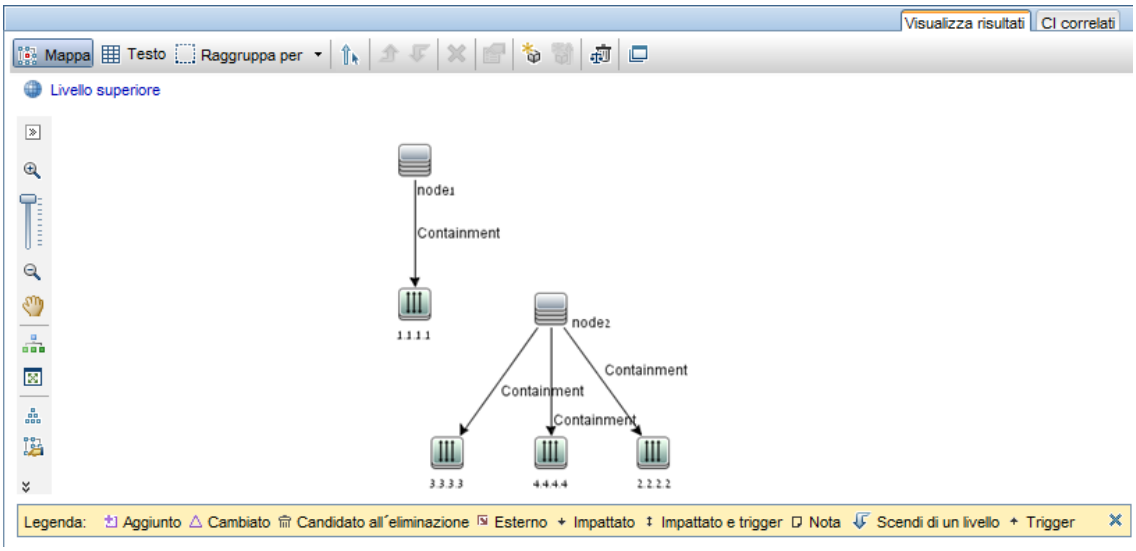
Nel file di mapping creiamo due tipi di CI di destinazione: **Computer** e **IP**. Computer ha una variabile e due attributi. IP ha un attributo.

Questo è il file XML di mapping:

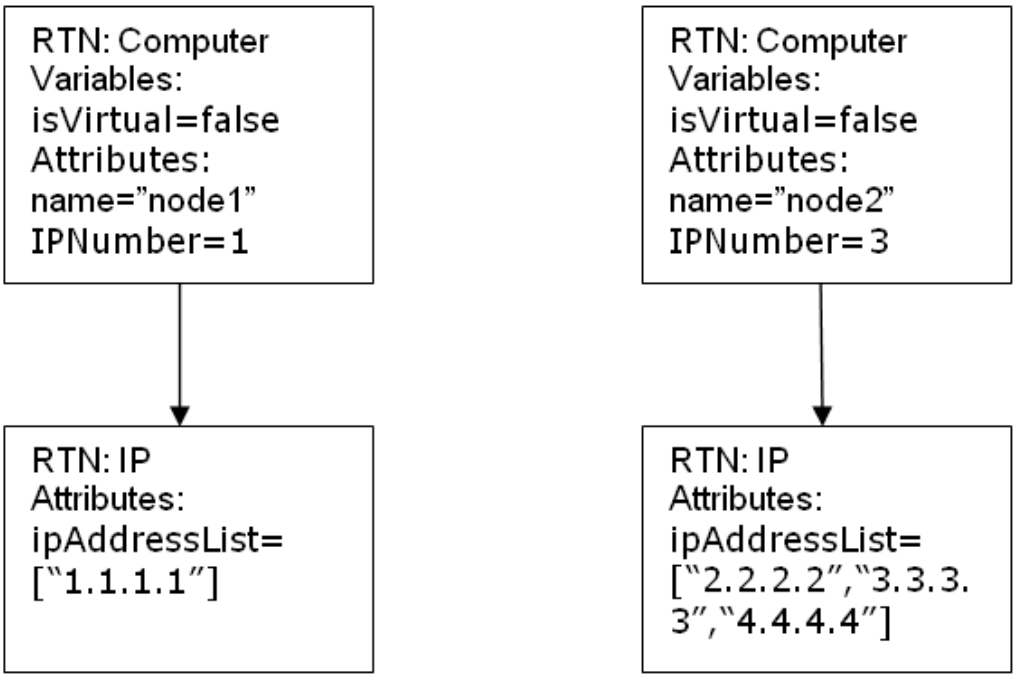
```
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://schemas.hp.com/ucmdb/1/pushAdapter">
        <info>
            <source name="UCMDB" versions="10.0" vendor="HP" />
            <target name="PushProduct" versions="9.3" vendor="HP" />
        </info>
        <import>
            <!—imports the Groovy script file. -->
            <scriptFile path="mappings.scripts.PushFunctions" />
        </import>
<targetcis>
<source_instance_type query-name="Node Push" root-element-name="Root">
 <target_ci_type name="Computer" is-
    valid="(Root['root_iscandidatefordeletion']==null) ? true :
    !Root['root_iscandidatefordeletion']">
  <variable name="isVirtual" datatype="BOOLEAN"
    value="PushFunctions.isVirtual(Root['root_class']) " />

  <target_mapping name="name" datatype="String" value="Root[`name’] " />
  <target_mapping name="ipNumber" datatype="INTEGER"
    value="Root.IpAddress.size()" />
  <target_mapping name="description" datatype="STRING" value=" PushFunctions
    .getDescription(isVirtual)" />
  <target_ci_type name="IP">
    <target_mapping name="ipAddressList" datatype="STRING_LIST"
    value="Root.IpAddress*.getAt(`name’) " />
</target_ci_type>
</target_ci_type>
</targetcis>
</integration>
```

I risultati della query appaiono come segue:



Questo è l'elenco RTN generato in base al file di mapping:



Ogni istanza radice viene mappata separatamente utilizzando il file di mapping. Per cui in questo esempio PushConnector riceve un elenco di due radici RTN.

Nota: L'adattatore Push precedente consentiva di creare un mapping generale di un tipo di CI. Il mapping del nuovo adattatore push è per query TQL. Durante l'esecuzione di un processo di invio dati che utilizza una query denominata **x**, l'adattatore cerca il file di mapping rilevante (quello che possiede l'attributo: query-name=x).

È possibile calcolare i valori nel file di mapping utilizzando la lingua script groovy. Per i dettagli consultare "Il traveler Groovy" nel seguito.

Il traveler Groovy

Accedere ai risultati della query TQL nel modo seguente:

- **Root[attr]** restituisce l'attributo **attr** dell''elemento radice.
- **Root.Query_Element_Name** restituisce un elenco delle istanze CI denominate nella TQL Query_Element_Name collegate al CI radice corrente.
- **Root.Query_Element_Name[2][attr]** restituisce l'attributo **attr**del terzo Query_Element_Name collegato al CI radice corrente.
- **Root.Query_Element_Name*.getAt(attr)** restituisce l'elenco degli attributi **attr** delle istanze CI denominate Query_Element_Name nella TQL collegate al CI radice corrente.

Esistono attributi aggiuntivi a cui il groovy traveler può accedere:

- **cmdb_id** – restituisce l'ID UCMDB ID del CI o della relazione sotto forma di stringa.
- **external_cmdb_id** – restituisce l'ID esterno del CI o della relazione sotto forma di stringa.
- **Element_type** – restituisce il tipo elemento del CI o della relazione sotto forma di stringa.

Il tag di importazione:

```
<import>
<scriptFile path="mappings.scripts.PushFunctions"/>
</import>
```

Ciò significa che viene dichiarata un'importazione per tutti gli script groovy nel file di mapping. In questo esempio, **PushFunctions** è un file di script groovy contenente alcune funzioni statiche a cui è possibile accedere durante il mapping (value=" PushFunctions.foo()")

source_instance_type

Il mapping viene eseguito per TQL, il valore del nome della query è la TQL correlata del mapping corrente. L'asterisco '*' indica che questo file di mapping è associato a tutte le query TQL che iniziano con il prefisso: **Node Push**.

```
<source_instance_type query-name="Node Push*" root-element-name="Root">
```

Il tag source_instance_type indica l'elemento radice che si sta mappando.

Il nome dell'elemento principale deve essere esattamente lo stesso della radice nella TQL.

target_ci_type

Questo tag viene utilizzato per la creazione della RTN.

Il nome dell'attributo rappresenta il nome target_ci_type: name=Computer

L'attributo **is-valid** è un valore boleano calcolato durante il mapping e stabilisce la validità dell'attuale target_ci. target_ci_types non validi non vengono aggiunti all'RTN. In questo esempio, non desideriamo creare un'istanza target_ci_type per cui l'attributo **root_iscandidatefordeletion** in UCMDB è true.

target_ci_type può contenere variabili calcolate durante il mapping:

```
<variable name="vSerialNo" datatype="STRING" value="Root['serial_number']"/>
```

La variabile **vSerialNo** ottiene il valore del **serial_number** della radice corrente.

L'attributo dell'RTN viene creato dal tag **target_mapping**. Il risultato dell'esecuzione dello script groovy nel campo **value** viene assegnato all'attributo RTN.

```
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
```

SerialNo assegna il valore della variabile **vSerialNo**.

È possibile definire un target_ci_type come figlio di un altro target_ci_type come segue:

```
<target_ci_type name="Portfolio">
<variable name="vSerialNo" datatype="STRING" value="Root['global_id']"/>
<target_mapping name="CMDBId" datatype="STRING" value="globalId"/>
<target_ci_type name=Asset">
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
</target_ci_type>
</target_ci_type>
```

Il **Portfolio** RTN avrà RTN figlio denominati **Asset**.

for-each-source-ci

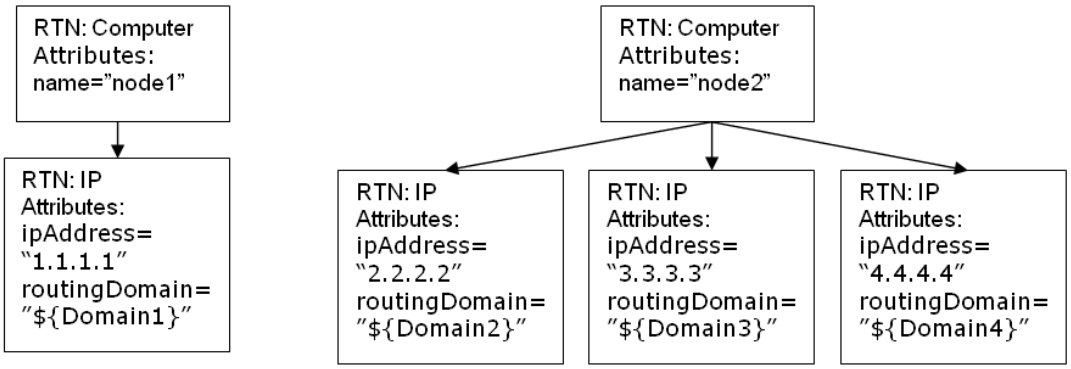
Questo tag elenca i CI specifici dell'istanza radice che contiene i seguenti campi:

- source-cis=" " – l'elenco dei CI per cui viene creato un CI di destinazione. Questo elenco viene definito dal groovy traveler nel campo **Root.IpAddress**.
- count-index=" " – una variabile che contiene l'indice dei CI nell'iterazione corrente del ciclo.
- var-name=" " – il nome del CI nell'iterazione corrente del ciclo.

Modifichiamo il file di mapping di esempio come segue:

```
<target_ci_type name="Computer">
  <target_mapping name="name" datatype="String" value="Root[`name'] " />
   <for-each-source-ci count-index="i" var-name="currIP" source-cis="Root.IpAddress" >
     <target_ci_type name="IP">
         <target_mapping name="ipAddress" datatype="STRING"
     value="Root.IpAddress[i][`name']"/>
         <target_mapping name="routingDomain" datatype="STRING"
     value="currIP['routing_domain']"/>
  </target_ci_type>
 </for-each-source-ci>
 </target_ci_type>
```

L'elenco RTN che si creerà secondo questo file di mapping avrà il seguente aspetto:



dynamic_mapping

Questo tag aggiunge la capacità di creare un mapping di dati dall'archivio dati di destinazione durante la creazione della struttura RTN.

Esempio: Supponiamo che il destinatario sia un database con una tabella denominata **Computer** contenente una colonna **id** e una **name** correlata a **Node.name** in UCMDB. Entrambe le colonne sono univoche. Inoltre, il database ha una tabella denominata **IP** contenente una chiave di riferimento al **parentID** nella tabella Computer. Il 'dynamic_mapping' può creare una mappa che archivia il nome e l'id come <name,id>. In base a questa mappa, l'adattatore può far corrispondere id ai computer e inviare il valore corretto all'attributo parentID attribute nella tabella IP. È possibile utilizzare questa mappa per assegnare un valore all'attributo parentID durante la creazione dell'RTN.

Il mapping viene stabilito da **map_property**. La dynamic_mapping viene eseguita una volta per ogni blocco.

```
<dynamic_mapping name="IdByName " keys-unique="true">
```

L'attributo **name** rappresenta il nome della mappa. L'attributo **keys-unique** indica se le chiavi sono univoche (ogni chiave viene mappata rispetto a un valore oppure rispetto a un insieme di valori).

Il nome della mappa in questo esempio è **IdByName** e possiede chiavi univoche. Al fine di accedere alla mappa nello script, eseguire il comando seguente:

```
DynamicMapHolder.getMap('IdByName')
```

Restituisce un riferimento a quella mappa.

Il tag **map_property** crea la proprietà su cui si basa il mapping.

Esempio:

```
<map_property property-name="SQLQuery" datatype="STRING"
property-value="SELECT name, id FROM Computer"/>
```

In questo esempio il nome della mappatura è **SQLQuery** e il suo valore è un'istruzione SQL che crea la mappa. L'implementazione dei metodi **retrieveUniqueMapping** e **retrieveNonUniqueMapping** per l'interfaccia PushConnector determinerà il contenuto effettivo della mappa restituita.

Variabili globali

Lo script groovy può accedere alle seguenti variabili globali nel file di mapping:

- **Topology** – Type: Topologia. Un'istanza della topologia del blocco corrente.
- **QueryDefinition** - Type: QueryDefinition. Un'istanza della definizione query della TQL corrente.
- **OutputCI** – Type: ResultTreeNode. L'RTN dell'elemento radice nel mapping corrente della struttura.
- **ClassModel** – Type: ClassModel. Un'istanza del modello di classe.
- **CustomerInformation** – Type: CustomerInformation. Informazioni sul cliente che sta eseguendo il processo.
- **Logger** – Type: DataAdapterLogger. Questo registratore è disponibile nell'adattatore per scrivere registri nel framework di accesso a UCMDB.

Scrivere gli script Groovy

In questa sezione è possibile creare il file **PushFunctions.groovy**. Questo file conterrà funzioni statiche utilizzate durante il mapping dell'istanza radice.

```
package mappings.scripts

public class PushFunctions {

	public static boolean isVirtual(def nodeRole){
		return isListContainsOne(def list, "MY_VM", "MY_SIMULATOR");
	}
```

```
public static String getDescription(boolean isVirtual){
               if(isVirtual){
                       return "This is a VM";
               }
               else{
                       return "This is physical machine";
               }
       }


private static boolean isListContainsOne(def list, ...stringList){
               //returns true if the list contains one of the values.
       }
}
```

Implementare l'interfaccia PushConnector

L'implementazione deve supportare i seguenti passaggi di base:

```
public class PushExampleAdapter implements PushAdapterConnector

{

public UpdateResult pushTreeNodes(PushConnectorInput input) throws DataAcces
sException{

// 1. build an UpdateResult instance - the UpdateResult is used to return ma
ppings between the sent ids to the actual ids that entered the data store.
// Also has an update status which allows to pass the status of data that was
actually pushed, detailed status reports on failed IDs, and actions actually
performed on successful ids.
// 2. handle the data:
// a. handle data to add. Can be retrieved by: input.getResultTreeNodes.getD
ataToAdd();
//  b. handle data to update.
//  c. handle data to delete.
// 3. Return the Update result.
       }

public void start(PushDataAdapterEnvironment env) throws DataAccessException{
       // this method is called when the integration point created,
or when the adapter is reloaded
       //(i.e after changing one of the mapping files
```

```
       // and pressing ‘save’).

}



public void testConnection(PushDataAdapterEnvironment env) throws DataAccess
Exception {
               // this method is called when pressing the 'test
connection' button in the
               //creation of the integration point.
               // For example if we push data to RDBMS this method
can create a connection
               //to the database and will run a dummy SQL statement.
               // If it fails it writes an error message to the log
and throws an exception.
       }



Map<Object, Object> retrieveUniqueMapping(MappingQuery mappingQuery){
//This method will create the map according to the given mappingQuery. It wi
ll be called in the
// mapping stage of the adapter execution, before the ‘UpdateResult pushTree
Nodes’ method.
// This method is called when the ‘keys-unique’ attribute of the ‘dynamic_ma
pping’ tag is true.
}
Map<Object, Set<Object>> retrieveNonUniqueMapping(MappingQuery mappingQuery){
 // This method is called when the ‘keys-unique’ attribute of the ‘dynamic_m
apping’ tag is false.
//  In this case a key can be mapped to several values.
}
}
```

Creare un pacchetto adattatore

Assicurarsi che il pacchetto adattatore contenga le seguenti cartelle:

- **adapterCode.** In questa cartella, crearne una denominata **PushExampleAdapter**, che conterrà il file .jar creato per il file PushExampleAdapter.java. Essa conterrà anche una cartella denominata **mappings**, dove è possibile collocare il file di mappatura creato in precedenza, **computerIPMapping.xml**. Nella stessa cartella ce ne sarà anche un'altra denominata **scripts** che contiene il file **PushFunctions.groovy**.
- **discoveryConfigFiles.** Per contenere i file di configurazione, quali i codici di errore, quando si

segnala un errore utilizzando UpdateResult. In questo esempio la cartella è vuota.

- **discoveryPatterns.** Per contenere la tql **push_example_adapter.xml**.
- **tql.** Per contenere la query TQL creata per questo esempio. Questa cartella è facoltativa, ma quando si distribuisce il pacchetto, la TQL viene creata automaticamente.

Schema del file di mapping

Nome e percorso elemento	Descrizione	Attributi
Integrazione	Definisce il contenuto di mapping del file. Deve essere il blocco più esterno nel file tranne per la riga iniziale ed eventuali commenti.	
info (integration)	Definisce informazioni sui repository di dati che vengono integrati.	
source (info)	Il prodotto d'origine	**Nome:** nome **Descrizione:** il nome del prodotto **Obbligatorio?:** obbligatorio **Tipo:** String
		Nome: vendor **Descrizione:** il fornitore del prodotto **Obbligatorio?:** obbligatorio **Tipo:** String
		Nome: versioni **Descrizione:** la versione del prodotto **Obbligatorio?:** obbligatorio **Tipo:** decimale

Nome e percorso elemento	Descrizione	Attributi
target (info)	Il prodotto di destinazione	**Nome:** nome **Descrizione:** il nome del prodotto **Obbligatorio?:** obbligatorio **Tipo:** String
		Nome: vendor **Descrizione:** il fornitore del prodotto **Obbligatorio?:** obbligatorio **Tipo:** String
		Nome: versioni **Descrizione:** la versione del prodotto **Obbligatorio?:** obbligatorio **Tipo:** decimale
Import (integration)	Un elemento contenitore per i file script importati.	
scriptFile (integration>import)	Definisce lo script groovy da importare.	**Nome:** path **Descrizione:** percorso del file script **Obbligatorio?:** obbligatorio **Tipo:** string
Targetcis (integration)	Un elemento contenitore per tipi CI target.	

Nome e percorso elemento	Descrizione	Attributi
Source_instance_type (integration>targetcis)	Definisce l'origine del tipo di istanza CI. Deve essere l'elemento radice come definito nella TQL.	**Nome:** query-name. **Descrizione:** il nome della TQL rilevante **Obbligatorio?:** obbligatorio **Tipo:** String
		Nome: nome **Descrizione:** l'elemento radice come definito nella TQL. **Obbligatorio?:** obbligatorio **Tipo:** String
dynamic_mapping (integration>targetcis> source_instance_type)	Definisce una mappa creata una volta per il blocco.	**Nome:** nome **Descrizione:** il nome della mappa **Obbligatorio?:** obbligatorio **Tipo:** String
		Nome: keys-unique **Descrizione:** afferma se le chiavi sono univoche o meno. **Obbligatorio?:** obbligatorio **Tipo:** boleano

Nome e percorso elemento	Descrizione	Attributi
target_ci_type (integration>targetcis> source_instance_type) -OR- (integration>targetcis> source_instance_type> for-each-source-ci)	Definisce il tipo CI di destinazione da aggiungere all'RTN.	**Nome:** nome **Descrizione:** il nome del tipo CI di destinazione **Obbligatorio?:** obbligatorio **Tipo:** String
		Nome: is-valid **Descrizione:** verifica se il CI corrente sia valido secondo lo script dato. **Obbligatorio?:** non obbligatorio **Tipo:** stringa (script groovy)

Nome e percorso elemento	Descrizione	Attributi
Variable (target_ci_type)	Definisce una variabile per il tipo CI di destinazione.	**Nome:** nome **Descrizione:** il nome della variabile **Obbligatorio?:** obbligatorio **Tipo:** String
		Nome: datatype **Descrizione:** il tipo di dati della variabile. **Obbligatorio?:** obbligatorio **Tipo:** type-enum (può essere uno dei seguenti: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Nome: valore **Descrizione:** il valore da assegnare alla variabile **Obbligatorio?:** obbligatorio **Tipo:** script groovy

Nome e percorso elemento	Descrizione	Attributi
target_mapping (target_ci_type)	Definisce un attributo per il tipo CI di destinazione.	**Nome:** nome **Descrizione:** il nome dell'attributo **Obbligatorio?:** obbligatorio **Tipo:** String
		Nome: datatype **Descrizione:** il tipo di dati della variabile. **Obbligatorio?:** obbligatorio **Tipo:** type-enum (può essere uno dei seguenti: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Nome: valore **Descrizione:** il valore da assegnare alla variabile **Obbligatorio?:** obbligatorio **Tipo:** script groovy

Nome e percorso elemento	Descrizione	Attributi
		Nome: ignore-on-null **Descrizione:** se il valore di mapping della destinazione è null e questo attributo è TRUE, ignorare l'attributo **Obbligatorio?:** non obbligatorio **Tipo:** boleano (script groovy)
before-mapping (target_ci_type)	Uno script groovy eseguito prima del mapping del tipo CI di destinazione.	
after-mapping (target_ci_type)	Uno script groovy eseguito dopo il mapping del tipo CI di destinazione.	

Nome e percorso elemento	Descrizione	Attributi
for-each-source-ci (target_ci_type)	Definisce un'iterazione dei CI specifici dell'istanza radice.	**Nome:** count-index **Descrizione:** indica l'indice del CI ripetuto attuale. **Obbligatorio?:** obbligatorio **Tipo:** String
		Nome: var-name **Descrizione:** la variabile che si riferisce al CI ripetuto corrente. **Obbligatorio?:** non obbligatorio **Tipo:** String
		Nome: source-cis **Descrizione:** il nome del CI nella query da ripetere **Obbligatorio?:** obbligatorio **Tipo:** stringa (script groovy)

Utilizzo delle API

Capitolo 8: Introduzione alle API

Questo capitolo comprende:

Panoramica delle API

Le API seguenti sono incluse con HP Universal CMDB:

- **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 "API di HP Universal CMDB" a pagina 259.
- **API servizio Web UCMDB.** Consente di scrivere le definizioni degli elementi di configurazione e le relazioni topologiche in UCMDB e di leggere le informazioni mediante query TQL e ad hoc. Per i dettagli consultare "API servizio Web di HP Universal CMDB" a pagina 268.
- **API Java di Gestione flusso di dati.** Consente di gestire sonde, processi, trigger e credenziali per Gestione flusso di dati. Per i dettagli consultare "API Java di Gestione flusso di dati" a pagina 302.
- **API servizi Web Gestione flusso di dati.** Consente di gestire sonde, processi, trigger e credenziali per Gestione flusso di dati. Per i dettagli consultare "API servizi Web Gestione flusso di dati" a pagina 304.

Nota: Per rilevare il valore completo della documentazione API, si consiglia di accedere alla documentazione online. La versione PDF non contiene i collegamenti alla documentazione API generata nel formato html.

Capitolo 9: API di HP Universal CMDB

Questo capitolo comprende:

Convenzioni

In questo capitolo vengono utilizzate le seguenti convenzioni:

- UCMDB si riferisce al database stesso di Universal Configuration Management. 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.

Per la documentazione completa sulle API disponibili consultare HP UCMDB API Reference.

Questi file si trovano nella cartella seguente:

\\<directory radice di UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Utilizzo dell'API di HP Universal CMDB

Nota: Utilizzare il presente capitolo insieme all'API Javadoc, disponibile nella Libreria della documentazione online.

L'HP Universal CMDBAPI è 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
- Recuperare le informazioni sulla cronologia di UCMDB

- Eseguire scenari whatif
- Recuperare informazioni sulle relazioni e sugli elementi di configurazione

I metodi di recupero delle informazioni sugli elementi di configurazione e sulle relazioni 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" nel seguito
- "Autorizzazioni" nel seguito

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 delle API di UCMDB" a pagina 265.

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 in CMDB. Tali utenti non sono utenti umani di CMDB, ma piuttosto applicazioni che si connettono a CMDB.

Inoltre, l'utente deve avere l'autorizzazione generale all'azione **Access to SDK** per accedervi.

> **Attenzione:** Il client API può operare anche con utenti comuni purché siano in possesso delle autorizzazioni di autenticazione API. Tuttavia, questa opzione non è consigliata.

Per i dettagli consultare "Creare un utente di integrazione" a pagina 263.

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 dall'oggetto servizio.
3. Ottiene una nuova definizione query dai valori predefiniti.

```
UcmdbServiceProvider provider =
   UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
UcmdbService ucmdbService =
   provider.connect(provider.createCredentials(USERNAME,
   PASSWORD), provider.createClientContext("Test"));
TopologyQueryService queryService = ucmdbService.getTopologyQueryService(
);
TopologyQueryFactory factory = queryService.getFactory();
QueryDefinition queryDefinition = factory.createQueryDefinition("Test Que
ry");
queryDefinition.addNode("Node").ofType("host");
Topology topology = queryService.executeQuery(queryDefinition);
System.out.println("There are " + topology.getAllCIs().size() + " hosts i
n uCMDB");
```

I servizi disponibili da **UcmdbService** sono:

Metodi servizio	Utilizzo
getAuthorizationModelService	Esegue operazioni di autorizzazione (crea utenti e gruppi di utenti, assegna ruoli a utenti e gruppi e così via).
getClassModelService	Informazioni sui tipi di CI e relazioni
getConfigurationService	Gestione impostazioni infrastruttura, per configurazione del server
getDataStoreMgmtService	Esegue una query sulle informazioni dell'archivio dati, inclusi quali CI e attributi saranno federati.
getDDMConfigurationService	Configurare il sistema di gestione flusso di dati
getDDMManagementService	Analizzare e visualizzare l'avanzamento, i risultati e gli errori del sistema di gestione flusso di dati
getDDMZoneService	Importa ed esporta zone di gestione (con le rispettive attività).
getHistoryService	Informazioni sulla cronologia dei CI monitorati (cambiamenti, rimozioni e così via)

Metodi servizio	Utilizzo
getImpactAnalysisService	Eseguire lo scenario dell'analisi impatto (anche noto come **correlazione**)
getLicensingService	Recupera informazioni sulle licenze installate nel sistema.
getMultipleCMDBService	Converte tra ID globali e ID di UCMDB.
getMultiTenancyService	Crea, legge, aggiorna ed elimina titolari.
getPersistencyService	Salva in modo permanente i dati binari in coppie di valori chiave.
getQueryManagementService	Gestire l'accesso alle query: salvare, eliminare, elencare esistenti Si occupa inoltre della convalida delle query e dell'individuazione delle dipendenze delle query
getReconciliationService	Fornisce funzionalità di identificazione e unione.
getResourceBundleManagementService	Assegnazione di tag alle risorse (servizi di "creazione pacchetti"). Consente la creazione esplicita di nuovi tag e la rimozione dei tag da tutte le risorse contrassegnate
getResourceManagementService	Distribuisce pacchetti di risorse (query TQL, viste, utenti e così via) al sistema.
getSecurityService	Verifica se le credenziali sono valide.
getServerService	Recupera informazioni generiche sul sistema.
getSnapshotService	Fornire servizi per la gestione delle istantanee (scattare, salvare, confrontare e così via)
getSoftwareSignatureService	Definisce gli elementi software che il sistema Gestione flusso di dati deve individuare.
getStateService	Fornire servizi per la gestione degli stati (elencare, aggiungere, rimuovere e così via)
getSystemHealthService	Fornire servizi integrità del sistema (indicatori di prestazione del sistema base, metriche di capacità e disponibilità)
getTopologyQueryService	Ottenere informazioni sull'universo IT
getTopologyUpdateService	Cambiare le informazioni nell'universo IT
getUcmdbVersion	Recupera informazioni sulle versioni e le build di UCMDB e dei Content Pack.

Metodi servizio	Utilizzo
getViewArchiveService	Visualizzare i servizi di archiviazione risultati. Consente il salvataggio del risultato della vista corrente e il recupero dei risultati precedentemente salvati
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

Il client comunica con il server tramite HTTP(s).

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 dove il `localhost` è il computer in cui è installato UCMDB e facendo clic sul collegamento **Download del client API**.

Mettere il file **.jar** nel classpath prima di compilare o eseguire l'applicazione.

Nota: L'utilizzo del file .jar delle API Java di UCMDB richiede l'installazione della versione JRE 6 o successiva.

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 l'SDK del client di UCMDB di essere autentificato nell'SDK del server ed eseguire le API. Le applicazioni scritte con questo insieme di API devono accedere con le credenziali dell'utente di integrazione.

Attenzione: È possibile connettersi anche utilizzando un normale utente UCMDB (ad esempio, admin). Tuttavia, questa opzione non è consigliata. Per connettersi con un utente UCMDB, è necessario attribuire le autorizzazioni di autenticazione per l'API.

Per creare un utente di integrazione:

1. Avviare il browser Web e specificare l'indirizzo del server come segue:

   `http://localhost:8080/jmx-console`.

   Potrebbe essere necessario effettuare l'accesso con nome utente e password.

2. In `UCMDB`, fare clic su **service=UCMDB Authorization Services**.

3. Individuare l'operazione **createUser**. Questo metodo accetta i seguenti parametri:
   - **customerId**. L'ID cliente.
   - **username**. Il nome dell'utente di integrazione.
   - **userDisplayName**. Il nome visualizzato dell'utente di integrazione.
   - **userLoginName**. Il nome di accesso dell'utente di integrazione.
   - **password**. La password dell'utente di integrazione.
4. Fare clic su **Invoke**.
5. In un ambiente a titolarità unica, individuare il metodo **setRolesForUser** e immettere i seguenti parametri:
   - **userName**. Il nome dell'utente di integrazione.
   - **roles.** SuperAdmin.

   Fare clic su **Invoke**.
6. In un ambiente a multi-titolarità, individuare il metodo **grantRolesToUserForAllTenants** ed inserire i seguenti parametri per assegnare il ruolo in connessione con tutti i titolari:
   - **userName**. Il nome dell'utente di integrazione.
   - **roles.** SuperAdmin.

   Fare clic su **Invoke**.

   In alternativa, per assegnare il ruolo in connessione con i titolari specifici, richiamare il metodo **grantRolesToUserForTenants** utilizzando lo stesso userName e i valori del parametro dei ruoli. Per il parametro **tenantNames**, immettere i titolari necessari.
7. Creare più utenti o chiudere la JMX Console.
8. Accedere a UCMDB come amministratore.
9. Dalla scheda **Amministrazione**, eseguire **Gestione pacchetti**.
10. Fare clic sull'icona **Crea pacchetto personalizzato**.
11. Immettere un nome per il nuovo pacchetto e fare clic su **Avanti**.
12. Nella scheda Selezione risorse, in **Impostazioni**, fare clic su **Utenti**.
13. Selezionare uno o più utenti creati utilizzando la JMX Console.

14. Fare clic su **Avanti** e quindi su **Fine**. Il nuovo pacchetto viene visualizzato nell'elenco Nome pacchetto in Gestione pacchetti.

15. Distribuire il pacchetto agli utenti che eseguiranno le applicazioni API.

    Per i dettagli consultare la sezione "Come 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** (**removeUser**, **resetPassword**, **UserAuthenticate**, e così via).

Ci sono due utenti di sistema preconfigurati. È consigliabile cambiare le relative password dopo l'installazione utilizzando il metodo **resetPassword**.

- **sysadmin**/**sysadmin**
- **UISysadmin**/**UISysadmin** (questo è anche il superutente di integrazione **SuperIntegrationUser**).

  Se si cambia la password UISysadmin utilizzando **resetPassword**, è necessario effettuare quanto segue:

  i. Nella JMX Console, individuare il servizio **UCMDB-UI:name=UCMDB Integration**.

  ii. Eseguire **setCMDBSuperIntegrationUser** con il nome utente e la nuova password dell'utente di integrazione.

Casi di utilizzo delle API di UCMDB

I casi di utilizzo elencati in questa sezione prevedono due sistemi:

- HP Universal CMDB server
- Un sistema di terze parti contenente un repository degli elementi di configurazione

In questa sezione vengono trattati i seguenti argomenti:

- "Popolamento di CMDB" alla pagina successiva
- "Interrogazione di CMDB " alla pagina successiva
- "Interrogazione del modello di classe" alla pagina successiva
- "Analisi dell'impatto del cambiamento " alla pagina successiva

Popolamento di CMDB

Casi di utilizzo:

- Una gestione asset di terze parti aggiorna CMDB con le informazioni disponibili solo nella gestione asset
- Diversi sistemi di terze parti popolano CMDB per creare un CMDB centrale che rilevi i cambiamenti ed esegua l'analisi di 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 di 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 da 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 del 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

Consultare i seguenti esempi di codice:

- Create a Connection
- Create and Execute an Ad Hoc Query
- Create and Execute a View
- Add and Delete Data
- Execute an Impact Analysis
- Query the Class Model
- Query a History Sample

Questi file si trovano nella directory seguente:

\\<directory radice di UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\JavaSDK_Samples

Capitolo 10: API servizio Web di HP Universal CMDB

Questo capitolo comprende:

Convenzioni

In questo capitolo vengono utilizzate le seguenti convenzioni:

- UCMDB si riferisce al database stesso di Universal Configuration Management. 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.

Per la documentazione completa sulle strutture di richiesta e risposta, consultare HP UCMDB Web Service API Reference. Questi file si trovano nella cartella seguente:

<directory radice di UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\CMDB_Schema\webframe.html

HP Universal CMDB Panoramica API servizio Web

Nota: Utilizzare il presente capitolo insieme alla documentazione dello schema UCMDB, disponibile nella Libreria della documentazione online.

L'HP Universal CMDBAPI servizio Web è utilizzata per integrare le applicazioni conHP 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 sugli elementi di configurazione e sulle relazioni 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" nel seguito
- "Autorizzazioni" alla pagina successiva

Utilizzi dell'API

L'API servizi Web UCMDB viene utilizzata per soddisfare diversi 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 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 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 in CMDB per usufruire delle capacità di interrogazione di CMDB.
- Altri sistemi, come ad esempio il sistema Release Control (CCM), possono utilizzare i metodi dell'analisi impatto per cambiare l'analisi.

Autorizzazioni

Per accedere al file WSDL per il servizio Web, visitare la pagina: **http://localhost:8080/axis2/services/UcmdbService?wsdl**. È necessario fornire all'amministratore del server le credenziali dell'utente per visualizzare il file WSDL.

L'utente deve avere l'autorizzazione all'azione generale **Run Legacy API** per accedervi.

La tabella seguente mostra le ulteriori autorizzazioni necessarie per ogni comando API servizio Web:

Comando API servizio Web	Autorizzazioni necessarie
addCIsAndRelations deleteCIsAndRelations updateCIsAndRelations	Azione generale: **Data Update**
executeTopologyQueryByName(AdHoc) executeTopologyQueryByNameWithParameters(AdHoc) executeTopologyQueryWithParameters(AdHoc)	Azione generale: **Esegui query in base a definizione** Per ciascuna query: autorizzazione **Vista**
getTopologyQueryExistingResultByName getTopologyQueryResultCountByName releaseChunks pullTopologyMapChunks getCINeighbours getFilteredCIsByType getCIsById getCIsByType getRelationsById	Azione generale: **Visualizza CI** Per ciascuna query: autorizzazione **Vista**

Comando API servizio Web	Autorizzazioni necessarie
getQueryNameOfView	Azione generale: **Visualizza CI** Per ciascuna vista: autorizzazione **Vista**
getChangedCIs	Azione generale: **Visualizza cronologia**, **Visualizza CI**
calculateImpact getImpactPath getImpactRulesByGroupName getImpactRulesByNamePrefix	Azione generale: **Run Impact Analysis**
getAllClassesHierarchy getClassAncestors getCmdbClassDefinition	Nessuno

Nota: Quando il contesto radice è stato cambiato in UCMDB, seguire questi passaggi per abilitare l'accesso all'API servizio Web:

1. Aprire il file di configurazione **\UCMDB\UCMDBServer\deploy\axis2\WEB-INF\web.xml** e individuare la sezione seguente:

   ```
   <servlet-class>
   org.apache.axis2.transport.http.AxisServlet
   </servlet-class>
   ```

   Aggiungere di seguito queste righe:

   ```
   <init-param>
   <param-name>axis2.find.context</param-name>
   <param-value>false</param-value>
   </init-param>
   ```

2. Aprire il file di configurazione **\UCMDB\UCMDBServer\deploy\axis2\WEB-INF\conf\axis2.xml** e individuare la riga seguente:

   ```
   <parameter name="enableSwA" locked="false">false</parameter>
   ```

   Aggiungere di seguito questa riga:

   ```
   <parameter name="contextRoot" locked="false">test1/setup1/axis2
   </parameter>
   ```

   dove **test1/setup1** è il contesto radice.

Per rimuovere il contesto radice, rimuovere il testo aggiunto al percorso.

3. Riavviare il server UCMDB.

Chiamare il servizio Web

Utilizzare le tecniche di programmazione SOAP standard nelle API servizio Web di HP Universal CMDB per consentire le 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 dei campi 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 a CMDB consultare "Casi di utilizzo delle API servizio Web UCMDB" a pagina 300.

Interrogazione di CMDB

CMDB viene interrogato utilizzando le API descritte in " Metodi di query UCMDB" a pagina 282. Le query e gli elementi di CMDB restituiti contengono sempre gli ID reali di UCMDB . Per esempi di utilizzo dei metodi di query consultare Query Example.

In questa sezione vengono trattati i seguenti argomenti:

Calcolo della risposta JIT

Per tutti i metodi di query, il server UCMDB 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 dell'esecuzione di una query restituiti all'applicazione client possono differire dai risultati della stessa query visualizzati nell'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 pagina 292 e "releaseChunks" a pagina 293.

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. 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 queste proprietà quando l'applicazione client deve passare un elemento `QualifierProperties` (un elenco di qualificatori che possono essere applicati alle proprietà). 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 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 `PropertiesListe` 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 CMDB con le API di aggiornamento. Per i dettagli dei metodi API consultare " Metodi di aggiornamento UCMDB" a pagina 294.

Questo compito include i passaggi seguenti:

- "Parametri di aggiornamento di UCMDB" alla pagina successiva
- "Utilizzo dei tipi ID con i metodi di aggiornamento" alla pagina successiva

Parametri di aggiornamento di UCMDB

Questo argomento descrive i parametri utilizzati solo dai metodi di aggiornamento del servizio.

- **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 `required`o 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 in 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" in precedenza.

- **Inserimento dei nuovi elementi di configurazione in 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 pagina 294 e "Metodi di query UCMDB" a pagina 282.

- **Inserimento delle nuove relazioni in 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 in CMDB, i CI già esistenti devono essere identificati da un ID CMDB reale o devono essere specificati in una raccolta `referencedCIs`.

Query del 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 tipi CI. Per i dettagli consultare CI Type Manager nella *Guida alla modellazione di HP Universal CMDB*.

Questa sezione fornisce informazioni sui seguenti metodi che restituiscono informazioni sui CIT e sulle relazioni:

- "getClassAncestors" nel seguito
- "getAllClassesHierarchy" nel seguito
- "getCmdbClassDefinition" alla pagina successiva

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 pagina 279.
className	Nome tipo. Per i dettagli consultare "Type Name" a pagina 280.

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 pagina 279.

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 della 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" alla pagina successiva.
className	Nome tipo. Per i dettagli consultare "Parametri generali di UCMDB " nel seguito.

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 punta ai 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 esempi di utilizzo dei metodi di analisi impatto consultare Impact Analysis Example.

Parametri generali di UCMDB

Questa sezione descrive i parametri più comuni dei metodi del servizio.

In questa sezione vengono trattati i seguenti argomenti:

- "CmdbContext" nel seguito
- "ID" nel seguito
- "Attributi chiave" nel seguito
- "Tipi di ID" nel seguito
- "CIProperties" nel seguito
- "Type Name" alla pagina successiva
- "Elemento di configurazione (CI, Configuration item)" alla pagina successiva
- "Relation" alla pagina successiva

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 Add/Edit Attribute Dialog Box nella Guida alla modellazione di HP Universal CMDB. Per informazioni sull'identificazione degli attributi chiave dall'interno dell'applicazione client API consultare "getCmdbClassDefinition" alla pagina precedente.

Tipi di ID

Un elemento `ID` può contenere un ID reale o un ID temporaneo.

Un ID reale è una stringa assegnata da CMDB che identifica un'entità all'interno del database. Un ID temporaneo può essere una stringa univoca all'interno della richiesta corrente.

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 in 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.

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)" nel seguito e "Relation" nel seguito.

Le raccolte di proprietà sono:

- `dateProps` - 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 UCMDB" per aggiornare un `CI`, l'elemento `ID` può contenere un ID di CMDB reale o un ID temporaneo assegnato dal client. Se viene utilizzato un ID temporaneo, impostare il flag `temp` su true. Quando si elimina un elemento, l'`ID` può essere vuoto. I " Metodi di query UCMDB" accettano `ID` reali come parametri di input e restituiscono `ID` reali nei risultati delle query.

`type` può essere un qualsiasi nome tipo definito in Gestione tipi CI. Per i dettagli consultare CI Type Manager nella Guida alla modellazione di HP Universal CMDB.

L'elemento `props`è una raccolta CIProperties. Per i dettagli consultare "Parametri generali di UCMDB " a pagina 278.

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 utilizzano i " Metodi di aggiornamento UCMDB" per aggiornare una `Relation`, il valore dell'ID della `Relation` può essere un ID di CMDB reale o un ID temporaneo. Quando si elimina un elemento, l'`ID` può essere vuoto. I " Metodi di query UCMDB" accettano `ID` reali come parametri di input e restituiscono `ID` reali nei risultati delle 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 in CMDB. Per ulteriori informazioni sulle classi o sui tipi consultare "Query del modello di classe di UCMDB " a pagina 277.

Per i dettagli consultare CI Type Manager 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 `props`è una raccolta CIProperties. Per i dettagli consultare "CIProperties" a pagina 279.

Parametri di output UCMDB

Questa sezione descrive i parametri di output più comuni dei metodi del servizio. Per i dettagli consultare la online schema documentation.

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 di 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 insieme 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" in precedenza).
- `relationNodes`. Questo è uno o più `RelationNode` (consultare "RelationNode" in precedenza).

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 pagina 273.

Metodi di query UCMDB

Questa sezione fornisce informazioni sui metodi seguenti:

executeTopologyQueryByNameWithParameters

Il metodo `executeTopologyQueryByNameWithParameters` recupera un elemento `topologyMap` che corrisponde 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 pagina 279.
queryName	Nome della TQL con parametri in CMDB per il quale recuperare la mappa.
parameterizedNodeList	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 specifico.

Output

Parametro	Commenti
topologyMap	Per i dettagli consultare "TopologyMap" alla pagina precedente.
chunkInfo	Per i dettagli consultare "ChunkInfo" alla pagina precedente e "Elaborazione delle risposte di grandi dimensioni" a pagina 273.

executeTopologyQueryWithParameters

Il metodo `executeTopologyQueryWithParameters` recupera un elemento `topologyMap` che corrisponde alla query con parametri.

La query viene passata nell'argomento `queryXML`. I valori per i parametri della query vengono passati nell'argomento `parameterizedNodeList`. 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.

Per usare una TQL esportata come input di questo metodo, procedere come segue:

1. Avviare un browser Web e specificare il seguente indirizzo: **http://localhost:8080/jmx-console**.

   Potrebbe essere necessario effettuare l'accesso con nome utente e password.

2. Fare clic su **UCMDB:service=TQL Services**.

3. Individuare l'operazione **exportTql**.

   - Nella casella del parametro **customerId** immettere **1** (valore predefinito).
   - Nella casella del parametro **patternName** immettere un nome TQL valido.

4. Fare clic su **Invoke**.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
queryXML	Una stringa XML che rappresenta una TQL senza tag di risorse.
parameterizedNodeList	Le condizioni che ciascun nodo deve soddisfare per essere incluso nei risultati della query.

Output

Parametro	Commenti
topologyMap	Per i dettagli consultare "TopologyMap" a pagina 282.
chunkInfo	Per i dettagli consultare "ChunkInfo" a pagina 282 e "Elaborazione delle risposte di grandi dimensioni" a pagina 273.

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 pagina 279.
ids	L'elenco degli ID dei CI radice i cui CI correlati vengono controllati per rilevare eventuali cambiamenti. Solo gli ID di 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
getChangedCIsResponseList	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 pagina 279.
ID	L'ID dei 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 pagina 280.
CIProperties	I dati da restituire su ciascun elemento di configurazione, denominati Layout query nell'interfaccia utente. Per i dettagli consultare "TypedProperties" a pagina 275.
relationProperties	I dati da restituire su ciascuna relazione (denominati Layout query nell'interfaccia utente). Per i dettagli consultare "TypedProperties" a pagina 275.

Output

Parametro	Commenti
topology	Per i dettagli consultare "Topology" a pagina 282.
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 pagina 279.
CIsTypedProperties	Raccolta typedproperties. Per i dettagli consultare "Altri elementi di specifica delle proprietà" a pagina 275.
IDs	Solo gli ID di CMDB reali sono validi in questa raccolta .

Output

Parametro	Commenti
CIs	Raccolta di elementi CI.
chunkInfo	Per i dettagli consultare "ChunkInfo" a pagina 282 e "Elaborazione delle risposte di grandi dimensioni" a pagina 273.

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 pagina 279.
type	Il nome classe. Per i dettagli consultare "Type Name" a pagina 280.
proprietà	I dati da restituire su ciascun elemento di configurazione. Per i dettagli consultare "CustomProperties" a pagina 275.

Output

Parametro	Commenti
CIs	Raccolta di elementi CI.
chunkInfo	Per i dettagli consultare: "ChunkInfo" a pagina 282 e"Elaborazione delle risposte di grandi dimensioni" a pagina 273.

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	Date 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 conservato per mantenere la compatibilità con le versioni precedenti. Nelle versioni precedenti, l'operatore era utilizzato solo con le proprietà della data/ora di creazione e modifica.
Equal	Stringa e numerico
EqualIgnoreCase	String
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.
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

Operatore	Tipo di condizione/Commenti
Like	String 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	String 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	Date 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 conservato per mantenere la compatibilità con le versioni precedenti. Nelle versioni precedenti, l'operatore era utilizzato solo con le proprietà della data/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(11f);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.FloatOperator.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.setPropertyNames(Arrays.asList("name","size"));
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties);
```

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
type	Il nome classe. Per i dettagli consultare "Type Name" a pagina 280. Il tipo può essere uno dei tipi definiti utilizzando Gestione tipi CI. Per i dettagli consultare CI Type Manager nella *Guida alla modellazione di HP Universal CMDB*.
proprietà	I dati da restituire su ciascun CI (denominati Layout query nell'interfaccia utente). Per i dettagli consultare "CustomProperties" a pagina 275.
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 pagina 282 e"Elaborazione delle risposte di grandi dimensioni" a pagina 273.

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 pagina 279.
viewName	Nome di una vista, ovvero un sottoinsieme del modello classe in CMDB.

Output

Parametro	Commenti
queryName	Nome della TQL in 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 pagina 279.
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
topologyMap	Per i dettagli consultare "TopologyMap" a pagina 282.
chunkInfo	Per i dettagli consultare "ChunkInfo" a pagina 282 e "Elaborazione delle risposte di grandi dimensioni" a pagina 273.

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 pagina 279.
queryName	Il nome di una TQL.
countInvisible	Se true, l'output include i CI definiti come invisibili nella query.

Output

Parametro	Commenti
getTopologyQueryResultCountByNameResponse	Il numero delle istanze che corrispondono alla query.

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 pagina 282 e"Interrogazione di CMDB" a pagina 272.

L'applicazione client deve essere in grado di gestire le mappe parziali.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
ChunkRequest	Il numero del blocco da recuperare e il `ChunkInfo` restituito dal metodo di query.
queryTypedProperties	Una raccolta di set di proprietà per recuperare gli elementi di uno specifico tipo CI.

Output

Parametro	Commenti
topologyMap	Per i dettagli consultare "TopologyMap" a pagina 282.
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,chunkRe
quest);
```

```
    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
            }
        }
}
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, chunk
Request);
    PullTopologyMapChunksResponse res =
        ucmdbService.pullTopologyMapChunks(req);
        for(int m=0 ;
            m < res.getTopologyMap().getCINodes().getCINodes().size();
            m++) {


            CIs cis =
            res.getTopologyMap().getCINodes().getCINodes().get(m).getCIs();
            for(int i=0 ; i < cis.getCIs().size(); 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 pagina 279.
chunksKey	L'identificatore dei dati sul server con blocchi. La chiave è un elemento di `ChunkInfo`.

Metodi di aggiornamento UCMDB

Questa sezione fornisce informazioni sui metodi seguenti:

- "addCIsAndRelations" nel seguito
- "addCustomer" alla pagina successiva
- "deleteCIsAndRelations" alla pagina successiva
- "removeCustomer" a pagina 296
- "updateCIsAndRelations" a pagina 296

addCIsAndRelations

Il metodo `addCIsAndRelations` aggiunge e aggiorna CI e relazioni.

Se i CI o le relazioni non esistono in CMDB, questi vengono aggiunti e le relative proprietà vengono impostate in base al contenuto dell'argomento `CIsAndRelationsUpdates`.

Se i CI o le relazioni esistono in 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 pagina 279.
updateExisting	Impostare su **true** per aggiornare gli elementi già 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 pagina 276.
ignoreValidation	Se true, non viene eseguito alcun controllo prima di aggiornare CMDB.
dataStore	Informazioni responsabile cambiamento.

Output

Parametro	Commenti
createdIDsMapList	L'elenco degli ID client mappati agli ID di CMDB. Per i dettagli vedere la descrizione precedente.
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 da 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 pagina 279.

Parametro	Commenti
CIsAndRelationsUpdates	Gli elementi da eliminare. Per i dettagli consultare "CIsAndRelationsUpdates" a pagina 276.
dataStore	Informazioni responsabile cambiamento.

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 in 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 pagina 279.
CIsAndRelationsUpdates	Gli elementi da aggiornare. Per i dettagli consultare "CIsAndRelationsUpdates" a pagina 276.
ignoreValidation	Se true, non viene eseguito alcun controllo prima di aggiornare CMDB.
dataStore	Informazioni responsabile cambiamento.

Output

Parametro	Commenti
createdIDsMapList	L'elenco degli ID client mappati agli ID di CMDB. Per i dettagli consultare "addCIsAndRelations" a pagina 294.

Metodi analisi impatto di UCMDB

Questa sezione fornisce informazioni sui metodi seguenti:

- "calculateImpact" nel seguito
- "getImpactPath" nel seguito
- "getImpactRulesByNamePrefix" alla pagina successiva

calculateImpact

Il metodo `calculateImpact` calcola quali CI sono influenzati da un dato CI in base alle regole definite in CMDB.

Questo mostra l'effetto di un'attivazione evento della regola. L'output `identifier di` calculateImpact viene utilizzato come input per "getImpactPath" nel seguito.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
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 pagina 282.
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" in precedenza è utilizzato come argomenti di input `identifier` di `getImpactPath`.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.

Parametro	Commenti
identifier	La chiave per la risposta del server restituita da calculateImpact.
relation	Una relazione basata su una delle "ShallowRelation"restituite 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 `ID`, `type`, `end1ID`, `end2ID`, `rule` e `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 pagina 279.
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`.

API servizio Web stato effettivo

L'API servizio Web stato effettivo viene utilizzata principalmente da Service Manager per recuperare informazioni sullo stato effettivo di un particolare ID CMDB o ID global e di un ID cliente specifico. L'API trova una query corrispondente nella cartella **Integration/SM Query** ed esegue la TQL con ID CMDB o ID globale come condizione, quindi restituisce i risultati della query.

URL servizio Web: http://[machine_name]:8080/axis2/services/ucmdbSMService

Schema del servizio Web: http://[machine_name]:8080/axis2/services/ucmdbSMService?xsd=xsd0

Flusso

Quando viene chiamato il metodo API, esso prova a trovare una query adatta nella cartella **Integration/SM Query**. Tenta di confrontare il tipo di CMDBID/GlobalID richiesto con una delle query della cartella citata, cercando prima un **QueryElement** con il nome **Root** e, se non trovato, tentando di utilizzare un **QueryNode** qualsiasi dello stesso tipo del CMDBID/GlobalID richiesto. Quando trova una Query e QueryNode adatto, pone CMDBID/GlobalID come una condizione sul QueryNode ed esegue la query. Il risultato viene poi restituito al chiamante dell'API.

Manipolazione del risultato mediante le trasformazioni

In alcuni casi si devono applicare delle ulteriori trasformazioni all'XML risultante (ad esempio, sommare le dimensioni di tutti i dischi e aggiungere quella somma come un attributo aggiuntivo al CI). Per aggiungere ulteriori trasformazioni ai risultati TQL, posizionare una risorsa chiamata **[tql_name].xslt** nella configurazione dell'adattatore come segue: **Gestione adattatori > ServiceDeskAdapter7-1 >File di configurazione > [tql_name].xslt.**

Registri di API servizio Web stato effettivo

La configurazione registro di UCMDB risiede in: **UCMDBServer/Conf/log** in vari file ***.properties**.

Per visualizzare i registri del flusso Stato effettivo SM:

1. Aprire il file **cmdb_soaapi.properties** e cambiare il livello di registro in DEBUG come segue: **loglevel=DEBUG**.
2. Aprire il file **fcmdb.properties** il livello di registro in DEBUG come segue: **loglevel=DEBUG**.
3. Attendere 1 minuto per consentire al server di recuperare i cambiamenti.
4. Eseguire Stato effettivo da SM.
5. Visualizzare i seguenti file di registro in **UCMDBServer/Runtime/log**:
   - cmdb.soaapi.log
   - fcmdb.log

Abilitazione dello stato effettivo dei CI replicati dopo il cambiamento del contesto radice

Se si è cambiato il contesto radice utilizzato per accedere a UCMDB, è necessario eseguire i seguenti cambiamenti alla configurazione per abilitare Stato effettivo dei CI replicati:

1. in **UCMDBServer\deploy\axis2\WEB-INF**, aprire il file **web.xml**.
2. Aggiungere il seguente parametro **servlet init** all'AxisServlet (incollare queste quattro righe dopo la riga 28):

```
<init-param>
```

```
<param-name>axis2.find.context</param-name>
<param-value>false</param-value>
</init-param>
```

Questa impostazione evita che Axis2 tenti di calcolare il contesto radice e di cercarla esplicitamente in **axis2.xml**.

3. In **UCMDBServer\deploy\axis2\WEB-INF\conf**, aprire il file **axis2.xml**.
4. Nella riga 58, rimuovere i commenti dal parametro **contextRoot** e modificarlo come segue:

```
<parameter name="contextRoot" locked="false">test/axis2</parameter>
```

(dove **test** è il nuovo contesto radice in **cmdb.xml**).

> **Nota:** Non ci sono barre all'inizio di **test/axis2**.

Casi di utilizzo delle API servizio Web UCMDB

I seguenti casi di utilizzo presuppongono due sistemi:

- HP Universal CMDB server
- Un sistema di terze parti contenente un repository degli elementi di configurazione

In questa sezione vengono trattati i seguenti argomenti:

- "Popolamento di CMDB" nel seguito
- "Interrogazione di CMDB" alla pagina successiva
- "Interrogazione del modello di classe" alla pagina successiva
- "Analisi dell'impatto del cambiamento" alla pagina successiva

Popolamento di CMDB

Casi di utilizzo:

- Una gestione asset di terze parti aggiorna CMDB con le informazioni disponibili solo nella gestione asset
- Diversi sistemi di terze parti popolano 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 di 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 da 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 del 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

Consultare gli esempi di codice seguenti:

- The Example Base Class
- Query Example
- Update Example
- Class Model Example
- Impact Analysis Example

Questi file si trovano nella directory seguente:

\\<directory radice di UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\WebServiceAPI_Samples

Capitolo 11: API Java di Gestione flusso di dati

Questo capitolo comprende:

Utilizzo dell'API Java di Gestione flusso di dati

Nota: Utilizzare questo capitolo insieme al DFM API Javadoc, disponibile nella Libreria della documentazione online.

Questo capitolo spiega come strumenti di terze parti o personalizzati possono utilizzare l'API Java di Gestione flusso di dati di HP per gestire il flusso dati. L'API fornisce metodi per:

- **Gestire credenziali**. Visualizzare, aggiungere, aggiornare e rimuovere.
- **Gestire processi**. Visualizzare, attivare e disattivare lo stato.
- **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**. Dati generali su domini e sonde.

I servizi seguenti sono disponibili nel pacchetto dei servizi di individuazione:

- **DDMConfigurationService.** Servizi per configurare le Data Flow Probe, i cluster, gli intervalli IP e le credenziali. Il server Universal Discovery può essere configurato con un file XML o tramite la Data Flow Probe.
- **DDMManagementService.** Servizi per analizzare e visualizzare l'avanzamento, i risultati e gli errori dell'esecuzione di Universal Discovery.
- **DDMSoftwareSignatureService.** Servizi per definire gli elementi software che verranno individuati dai componenti Data Flow Probe. Le definizioni si applicano a livello di sistema. Se viene definito più di un componente Data Flow Probe, le definizioni si applicano a tutti i componenti.
- **DDMZoneService.** Servizi per gestire l'individuazione basata sulle zone.

Oltre a questi servizi, vi sono API client di Gestione flusso di dati utilizzate per la creazione degli adattatori Jython. Per i dettagli consultare "Sviluppo degli adattatori Jython" a pagina 38.

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 in CMDB. Tali utenti non sono utenti umani di CMDB, ma piuttosto applicazioni che si connettono a CMDB.

Inoltre, l'utente deve avere l'autorizzazione generale all'azione **Access to SDK** per accedervi.

Attenzione: Il client API può operare anche con utenti comuni purché siano in possesso delle autorizzazioni di autenticazione API. Tuttavia, questa opzione non è consigliata.

Per i dettagli consultare "Creare un utente di integrazione" a pagina 263.

Capitolo 12: API servizi Web Gestione flusso di dati

Questo capitolo comprende:

Panoramica di API servizi Web Gestione flusso di dati

Questo capitolo spiega come strumenti di terze parti o personalizzati possono utilizzare l'API servizio Web Gestione flusso di dati per gestire il flusso di dati.

L'API servizio Web Gestione flusso di dati è utilizzata per integrare le applicazioni con HP Universal CMDB. L'API fornisce dei metodi per:

- **gestire credenziali**. Visualizzare, aggiungere, aggiornare e rimuovere.
- **gestire processi**. Visualizzare, attivare e disattivare lo stato.
- **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**. Dati generali su domini e sonde.

Gli utenti del servizio Web Gestione flusso di dati di HP devono conoscere:

- Le specifiche SOAP
- Un linguaggio di programmazione orientato all'oggetto come C++, C# o Java
- HP Universal CMDB
- Gestione flusso di dati

L'utente deve avere l'autorizzazione all'azione generale **Run Legacy API** per accedervi.

Per la documentazione completa sulle operazioni disponibili consultare *HP Universal Discovery Schema Reference*. Questi file si trovano nella cartella seguente:

<directory radice di UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_Schema\webframe.html

Convenzioni

In questo capitolo vengono utilizzate le seguenti convenzioni:

- Questo stile `Elemento` indica che un elemento è un'entità nel database o un elemento definito nello schema che include le strutture passate o restituite da metodi. Il testo normale indica che l'elemento viene discusso in un contesto generale.
- Gli elementi e gli argomenti del metodo di Gestione flusso di dati sono immessi con l'iniziale maiuscola o minuscola specificata nello schema. Ciò solitamente significa che un nome della classe o un riferimento generico a un'istanza della classe è in maiuscolo. Un elemento o argomento di un metodo non è in maiuscolo. Ad esempio, una `credenziale` è un elemento del tipo `Credenziale` passato a un metodo.

Chiamata del servizio Web

L'API del servizio Web HP Gestione flusso di dati consente le chiamate dei metodi lato server utilizzando le tecniche di programmazione SOAP standard. 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`, il servizio 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.

Per chiamare il sevizio, usare.

- `Protocollo: http o https (a seconda della configurazione del server)`
- `URL: <Server UCMDB>:8080/axis2/services/DiscoveryService`
- `Password predefinita: "admin"`
- `Nome utente predefinito: "admin"`

I programmatori SOAP possono accedere al WSDL dall'indirizzo:

- `axis2/services/DiscoveryService?wsdl`

Metodi e strutture dati di Gestione flusso di dati

Questa sezione elenca i metodi e le strutture dati dell'API del servizio Web Gestione flusso di dati e fornisce un breve riepilogo dei loro usi. Per la documentazione completa della richiesta e della risposta di ciascuna operazione consultare *HP Universal Discovery Schema Reference*.

In questa sezione vengono trattati i seguenti argomenti:

- "Strutture dei dati" alla pagina successiva
- "Gestione dei metodi del processo di individuazione" alla pagina successiva

Strutture dei dati

Queste sono alcune strutture di dati utilizzate nell'API servizi Web Gestione flusso di dati.

CIProperties

`CIProperties` è una raccolta di raccolte. Ogni raccolta contiene le proprietà di un tipo di dati differente. Ad esempio, ci può essere una raccolta `dateProps`, una raccolta `strListProps`, una raccolta `xmlProps` e così via.

Ogni tipo di raccolta contiene le proprietà singole di un dato tipo. I nomi di questi elementi delle proprietà sono identici a ciò che contengono ma al singolare. Ad esempio, `dateProps` contiene elementi `dateProp`. Ciascuna proprietà è una coppia nome-valore.

Vedere CIProperties in *HP Universal Discovery Schema Reference*.

IPList

Un elenco di elementi `IP`, ognuno dei quali contiene un indirizzo IPv4 o IPv6.

Consultare IPList in *HP Universal Discovery Schema Reference*.

IPRange

Un `IPRange` ha due elementi, `Start` e `End`. Ciascun elemento contiene un elemento `Address`, che è un indirizzo IPv4 o IPv6.

Vedere IPRange in *HP Universal Discovery Schema Reference*.

Scope

Due `IPRanges`. `Exclude` è una raccolta di `IPRanges` da escludere dal processo. `Include` è una raccolta di `IPRanges` da includere nel processo.

Vedere Scope in *HP Universal Discovery Schema Reference*

Gestione dei metodi del processo di individuazione

activateJob

Attiva il processo specificato.

Vedere " Codice di esempio" a pagina 316.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
JobName	Il nome del processo.

deactivateJob

Disattiva il processo specificato.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
JobName	Il nome del processo.

dispatchAdHocJob

Invia un processo sulla sonda ad hoc. Il processo deve essere attivo e contenere il CI trigger specificato.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
JobName	Il nome del processo.
CIID	ID del CI trigger.
ProbeName	Il nome della sonda.
Timeout	In millisecondi

getDiscoveryJobsNames

Restituisce l'elenco dei nomi di processo.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.

Output

Parametro	Commenti
strList	L'elenco dei nomi di processo.

isJobActive

Controlla se il processo è attivo.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
JobName	Il nome del processo da verificare.

Output

Parametro	Commenti
JobState	True se il processo è attivo.

Gestione dei metodi di attivazione

addTriggerCI

Aggiunge un nuovo CI trigger al processo specificato.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
JobName	Il nome del processo.
CIID	ID del CI trigger.

addTriggerTQL

Aggiunge una nuova TQL trigger al processo specificato.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
JobName	Il nome del processo.
TqlName	Il nome della TQL da aggiungere.

disableTriggerTQL

Impedisce alla TQL di attivare il processo ma non lo rimuove permanentemente dall'elenco delle query che attivano il processo.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
JobName	Il nome del processo.

removeTriggerCI

Rimuove il CI specificato dall'elenco dei CI che attivano il processo.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
JobName	Nome del processo.
CIID	ID del CI trigger.

removeTriggerTQL

Rimuove la TQL specificata dall'elenco delle query che attivano il processo.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
JobName	La raccolta dei nomi di processo da verificare.
CIID	L'ID della TQL da rimuovere.

setTriggerTQLProbesLimit

Limita le sonde in cui la TQL è attiva nel processo all'elenco specificato.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
JobName	Il nome del processo.
tqlName	Il nome della TQL.
probesLimit	L'elenco delle sonde in cui la TQL è attiva.

Dominio e metodi dei dati della sonda

getDomainType

Restituisce il tipo di dominio.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il nome del dominio.

Output

Parametro	Commenti
domainType	Il tipo di dominio.

getDomainsNames

Restituisce i nomi dei domini correnti.

Vedere " Codice di esempio" a pagina 316.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.

Output

Parametro	Commenti
domainNames	L'elenco dei nomi del dominio.

getProbeIPs

Restituisce gli indirizzi IP della sonda specificata.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il dominio da verificare.
probeName	Il nome di una sonda utilizzata su quel dominio.

Output

Parametro	Commenti
probeIPs	L'"IPList" degli indirizzi nella sonda.

getProbesNames

Restituisce i nomi delle sonde nel dominio specificato.

Vedere " Codice di esempio" a pagina 316.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il dominio da verificare.

Output

Parametro	Commenti
probesName	L'elenco delle sonde sul dominio.

getProbeScope

Restituisce la definizione ambito della sonda specificata.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il dominio da verificare.
probeName	Il nome della sonda.

Output

Parametro	Commenti
probeScope	L'"Scope" della sonda.

isProbeConnected

Controlla se la sonda specificata è connessa.

Vedere " Codice di esempio" a pagina 316.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il dominio da verificare.
probeName	Il dominio da verificare.

Output

Parametro	Commenti
isConnected	True se la sonda è connessa.

updateProbeScope

Imposta l'ambito della sonda specificata sostituendo l'ambito esistente.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il dominio.
probeName	La sonda da aggiornare.
newScope	L'"Scope" da impostare per la sonda.

Metodi dei dati delle credenziali

addCredentialsEntry

Aggiunge una voce delle credenziali al protocollo per il dominio specificato.

Vedere " Codice di esempio" a pagina 316.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il dominio da aggiornare.
protocolName	Il nome del protocollo.
credentialsEntryParameters	La raccolta "CIProperties" delle nuove credenziali.

Output

Parametro	Commenti
credentialsEntryID	L'ID CI della voce della nuova credenziale.

getCredentialsEntriesIDs

Restituisce gli ID delle credenziali definite per il protocollo specificato.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il dominio per il quale ottenere le credenziali.
protocolName	Il nome di un protocollo utilizzato su quel dominio.

Output

Parametro	Commenti
credentialsEntryIDs	L'elenco degli ID delle credenziali per il protocollo sul dominio.

getCredentialsEntry

Restituisce le credenziali definite per il protocollo specificato. Gli attributi crittografati vengono restituiti vuoti.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il dominio per il quale ottenere le credenziali.
protocolName	Il nome di un protocollo utilizzato su quel dominio.
credentialsEntryID	L'ID delle credenziali da ottenere.

Output

Parametro	Commenti
credentialsEntryParameters	La raccolta "CIProperties" delle credenziali.

removeCredentialsEntry

Rimuove le credenziali specificate dal protocollo.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il dominio.
protocolName	Il nome di un protocollo utilizzato sul dominio.
credentialsEntryID	L'ID delle credenziali da rimuovere.

updateCredentialsEntry

Imposta i nuovi valori per le proprietà della voce delle credenziali specificate.

Le proprietà esistenti vengono eliminate e impostate le nuove. Qualsiasi proprietà, il cui valore non è impostato in questa chiamata, viene lasciata indefinita.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
domainName	Il dominio in cui aggiornare le credenziali.
protocolName	Il nome di un protocollo utilizzato sul dominio.
credentialsEntryID	L'ID delle credenziali da aggiornare.
credentialsEntryParameters	La raccolta "CIProperties" da impostare come proprietà per le credenziali.

Metodi di aggiornamento dati

rediscoverCIs

Individua i trigger che hanno individuato gli oggetti CI specificati e riesegue questi trigger. **rediscoverCIs** viene eseguito in modo asincrono. Chiamare **checkDiscoveryProgress** per determinare quando la reindividuazione è completa.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
CmdbIDs	Raccolta di ID degli oggetti da reindividuare.

Output

Parametro	Commenti
isSucceed	True se riesce la reindividuazione dei CI.

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 a **rediscoverCIs** è completata.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
CmdbIDs	Raccolta di ID degli oggetti nella chiamata di reindividuazione da rilevare.

Output

Parametro	Commenti
progress	Un processo completo ha un avanzamento di 1. Processi non completi hanno una frazione minore di 1.

rediscoverViewCIs

Individua i trigger che hanno creato i dati per popolare la vista specificata e riesegue quei trigger. **rediscoverViewCIs** viene eseguito in modo asincrono. Chiamare **checkViewDiscoveryProgress** per determinare quando la reindividuazione è completa.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
viewName	Le viste da verificare.

Output

Parametro	Commenti
isSucceed	True se riesce la reindividuazione dei CI.

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.

Input

Parametro	Commenti
cmdbContext	Per i dettagli consultare "CmdbContext" a pagina 279.
viewName	La raccolta di viste da verificare.

Output

Parametro	Commenti
progress	Un processo completo ha un avanzamento di 1. Processi non completi hanno una frazione minore di 1.

Codice di esempio

```
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.*;
import com.hp.ucmdb.generated.types.*;
public class test {
    static final String HOST_NAME = "<my_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 = "<my_password>";
    private static final String USERNAME = "<my_username>";

    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 credential
s");
            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", newCredsPropertie
s);
        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, C
IProperties 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 Ge
tDomainsNamesRequest(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=" + isConn
ected);
            }
        }
    }


    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.AUTHENTICATE,auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service ", e);
```

```
        }
        return discoveryService;
    }
}
```

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(cmdbCont
ext)).getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create credential
s");
            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", newCredsPropertie
s);
        setStringProperties("ntadminprotocol_ntdomain", "test doamin", newCredsP
roperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(new AddCredentialsEntryRequest(doma
inName,  "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, C
IProperties 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 Ge
tDomainsNamesRequest(cmdbContext ));
        // Go over all the domains
```

```
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName = result.getDomainNames().getStrValue(0);
            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(new GetProbesNamesRequest(domain
Name, 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 IsProbeConnectedReques
t(domainName, probeName, cmdbContext));
                Boolean isConnected = connectedRequest.getIsConnected();
                // Do something ...
                System.out.println("probe " + probeName + " isconnect=" + isConn
ected);
            }
        }
    }

    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.toSt
ring());

            // Authenticate info
            HttpTransportProperties.Authenticator auth = new HttpTransportProper
ties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);
            serviceStub._getServiceClient().getOptions().setProperty(HTTPConstan
ts.AUTHENTICATE,auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service ", e);
        }
        return discoveryService;
    }
} // End class
```

Inviateci i vostri commenti!

Se avete commenti sul documento, è possibile contattare via e-mail il team che si occupa della documentazione. Se sul vostro sistema è già configurato un client di posta, fare click sul collegamento sopra per aprire un nuovo messagio di posta elettronica contenente nell'oggetto le seguenti informazioni:

Commento su Guida di riferimento per lo sviluppatore (Universal CMDB 10.10)

Digitare il commento nel testo dell'e-mail e fare clic sui Invia.

Se non è stato configurato nessun client di posta, copiare le informazioni indicate sopra in un nuovo messaggio utilizzando un client di Web mail e indirizzare il commento a SW-Doc@hp.com.