

HP Universal CMDB

Para os Sistemas operacionais Windows e Red Hat Enterprise Linux

Versão do software: 10.00

Guia de Referência do Desenvolvedor

Data de lançamento do documento: Junho de 2012

Data de lançamento do software: Junho de 2012



Avisos Legais

Garantia

As únicas garantias para produtos e serviços HP estão estipuladas nas declarações de garantia expressa que acompanham esses produtos e serviços. Nenhum conteúdo deste documento deve ser interpretado como parte de uma garantia adicional. A HP não se responsabiliza por erros técnicos ou editoriais ou por omissões presentes neste documento.

As informações contidas neste documento estão sujeitas a mudanças sem aviso prévio.

Legenda de Direitos Restritos

Software de computador confidencial. Uma licença válida da HP é necessária para posse, utilização ou cópia. Consistentes com o FAR 12.211 e 12.212, o Software de Computador Comercial, a Documentação de Software de Computador e os Dados Técnicos para Itens Comerciais estão licenciados junto ao Governo dos Estados Unidos sob a licença comercial padrão do fornecedor.

Aviso de Direitos Autorais

© Copyright 2002 - 2012 Hewlett-Packard Development Company, L.P.

Avisos de Marcas Comerciais

Adobe™ é marca comercial da Adobe Systems Incorporated.

Microsoft® e Windows® são marcas registradas da Microsoft Corporation nos Estados Unidos.

UNIX® é marca registrada da The Open Group.

Atualizações da Documentação

A página inicial deste documento contém as seguintes informações de identificação:

- Número de versão do software, que indica a versão do software.
- Data de lançamento do documento, que é alterada a cada vez que o documento é atualizado.
- Data de lançamento do software, que indica a data de lançamento desta versão do software.

Para verificar as atualizações recentes ou se você está utilizando a edição mais recente, vá para:

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

Esse site exige que você se registre para obter um HP Passport e para se conectar. Para se registrar e obter uma ID do HP Passport, vá para:

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

Ou clique no link **New users - please register** (Registro de novos usuários) na página de logon do HP Passport.

Você também receberá edições novas ou atualizadas se assinar o serviço de suporte adequado ao produto. Entre em contato com seu representante de vendas HP para saber mais detalhes.

Suporte

Visite o site de Suporte Online da HP Software em:

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

Esse site fornece informações de contato e detalhes sobre produtos, serviços e suporte oferecidos pela HP Software.

O suporte on-line da HP Software fornece recursos de autoajuda aos clientes. Ele oferece uma maneira rápida e eficiente de acessar ferramentas de suporte técnico interativas necessárias para gerenciar seus negócios. Como um estimado cliente de suporte, você pode aproveitar o site de suporte para:

- Pesquisar documentos com informações de interesse
- Enviar e rastrear os casos de suporte e solicitações de aperfeiçoamentos
- Fazer download dos patches de software
- Gerenciar contratos de suporte
- Procurar contatos de suporte HP
- Revisar informações sobre os serviços disponíveis
- Participar de discussões com outros clientes de software
- Pesquisar e registrar-se para treinamentos de software

A maior parte das áreas de suporte exige que você se registre como usuário de um HP Passport e, em seguida, se conecte. Muitas também requerem um contrato de suporte ativo. Para se cadastrar e obter uma ID do HP Passport, acesse:

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

Para mais informações sobre níveis de acesso, vá para:

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

Isenção de responsabilidade para a versão em PDF da ajuda on-line

Este documento é uma versão em PDF da ajuda on-line. Este arquivo em PDF é fornecido para que você possa imprimir facilmente vários tópicos das informações da ajuda ou ler a ajuda on-line em formato PDF.

Observação: Alguns tópicos não foram convertidos adequadamente em PDF, causando problemas de formatação. Alguns elementos da ajuda on-line foram completamente removidos da versão em PDF. Esses tópicos com problemas podem ser impressos com êxito a partir da ajuda on-line.

Conteúdo

Guia de Referência do Desenvolvedor	1
Conteúdo	6
Criando Adaptadores de Descoberta e Integração	12
Criação e desenvolvimento de adaptadores	13
Visão geral da criação e desenvolvimento de adaptadores	13
Criação de conteúdo	13
O ciclo de desenvolvimento do adaptador	14
Início e preparação da cópia	15
Desenvolvimento e teste	16
Limpeza e documentação	16
Criação do pacote	16
Gerenciamento de fluxo de dados e integração	16
Associando valor comercial ao desenvolvimento da descoberta	17
Pesquisando requisitos de integração	18
Desenvolvendo conteúdo de integração	21
Desenvolvendo conteúdo de descoberta	23
Adaptadores de descoberta e componentes relacionados	23
Separando adaptadores	24
Implementar um adaptador de descoberta	25
Etapa 1: Criar um adaptador	27
Etapa 2: Atribuir um trabalho ao adaptador	34
Etapa 3: Criar código Jython	35
Configurar execução de processo remoto	36
Desenvolvimento de adaptadores Jython	38
Referência da API de Gerenciamento de Fluxo de Dados da HP	38
Criar código Jython	38
Usar arquivos JAR Java externos em Jython	39
Execução do código	39

Modificando scripts prontos	39
Estrutura do arquivo Jython	40
Importações	41
Função principal – DiscoveryMain	41
Definição de funções	41
Geração de resultados pelo script Jython	43
A sintaxe de ObjectStateHolder	43
Instância do Framework	44
Localizando as credenciais corretas (para adaptadores de conexão)	47
Manipulando exceções de Java	48
Oferecer suporte a localização em adaptadores Jython	49
Adicionar suporte para um novo idioma	49
Alterar o idioma padrão	51
Determinar o conjunto de caracteres para codificação	51
Definir um novo trabalho para operar com dados localizados	51
Decodificar comandos sem uma palavra-chave	52
Trabalhar com bundles de recursos	53
Referência de API	54
Campos	54
Argumentos	55
Argumentos	56
Argumentos	56
Trabalhar com o Analisador de Descoberta	56
Tarefas e registros	57
Logs	57
Executar o Analisador de Descoberta a partir do Eclipse	63
Registrar código do DFM	72
Bibliotecas e utilitários Jython	73
Mensagem de erro	77
Visão geral de mensagens de erro	77
Convenções da criação de mensagens	77
Níveis de gravidade de erro	80

Desenvolvimento de adaptadores de banco de dados genéricos	82
Visão geral dos adaptadores de banco de dados genéricos	83
Consultas TQL para o adaptador de banco de dados genérico	83
Reconciliação	84
Hibernar como provedor de JPA	84
Preparar criação do adaptador	87
Preparar o pacote de adaptadores	91
Configurar o adaptador - Método mínimo	94
Configurar o adaptador - Método avançado	99
Implementar um plugin	103
Implantar o adaptador	105
Editar o adaptador	106
Criar um ponto de integração	106
Criar uma visualização	106
Calcular os resultados	107
Visualizar os resultados	107
Visualizar relatórios	108
Habilitar arquivos de log	108
Usar o Eclipse para mapear entre atributos de TEC e tabelas de banco de dados	108
Arquivos de configuração do adaptador	115
O arquivo adapter.conf	116
O arquivo simplifiedConfiguration.xml	117
O arquivo orm.xml	119
O arquivo reconciliation_types.txt	132
O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores) .	132
O arquivo transformations.txt	134
O arquivo discriminator.properties	134
O arquivo replication_config.txt	136
O arquivo fixed_values.txt	136
O arquivo persistence.xml	136
Conversores prontos	137
Plug-ins	141

Exemplos de Configuração	141
Arquivos de log do adaptador	150
Referências externas	151
Solução de problemas e limitações	151
Desenvolvimento de adaptadores Java	153
Visão geral do Federation Framework	153
Fluxo SourceDataAdapter	156
Fluxo SourceChangesDataAdapter	157
Fluxo PopulateDataAdapter	157
Fluxo PopulateChangesDataAdapter	157
Interação de adaptador e mapeamento com o Federation Framework	158
Fluxo do Federation Framework para consultas TQL federadas	158
Interações entre o Federation Framework, servidor, adaptador e mecanismo de mapeamento	160
Fluxo do Federation Framework para população	168
Interfaces de adaptador	170
Interfaces OneNode	170
Interfaces de adaptador de dados	170
Interfaces adicionais	171
Interfaces de adaptador para sincronização	171
Depurar recursos do adaptador	171
Adicionar um adaptador para uma nova fonte de dados externos	171
Implementar o mecanismo de mapeamento	179
Criar um adaptador de amostra	180
Marcas e propriedades de configuração XML	181
Desenvolvimento de adaptadores push	184
Visão geral sobre o desenvolvimento de adaptadores push	184
Sincronização diferencial	184
Preparar os arquivos de mapeamento	185
Escrever scripts Jython	187
Suporte à sincronização diferencial	190
Criar um pacote de adaptador	192

Esquema do arquivo de mapeamento	193
Esquema de resultados de mapeamento	202
Desenvolvendo adaptadores push genéricos aprimorados	205
Visão Geral sobre o desenvolvimento de adaptadores push aprimorados	205
O arquivo de mapeamento	205
O viajante do Groovy	208
Escrever scripts Groovy	211
Implementar interface PushConnector	211
Criar um pacote de adaptador	212
Esquema do arquivo de mapeamento	213
Usando APIs	219
Introdução a APIs	220
Visão geral sobre APIs	220
API do HP Universal CMDB	221
Convenções	221
Usando a API do HP Universal CMDB	221
Estrutura geral de um aplicativo	222
Colocar o arquivo Jar da API no classpath	224
Criar um usuário de integração	224
Referência de API do HP Universal CMDB	226
Casos de uso	226
Exemplos	227
API do Gerenciamento de Fluxo de Dados	228
Visão Geral da API do Gerenciamento de Fluxo de Dados	228
Convenções	228
Serviço Web do Gerenciamento de Fluxo de Dados	229
Chamar o serviço Web	229
Métodos de Gerenciamento de Fluxo de Dados	230
Estruturas de dados	230
Gerenciando métodos de trabalho de descoberta	231
Gerenciando métodos de acionador	232
Métodos de dados de sonda e domínio	234

Métodos de dados de credenciais	236
Métodos de atualização de dados	238
Amostra de código	240

Criando Adaptadores de Descoberta e Integração

Capítulo 1

Criação e desenvolvimento de adaptadores

Este capítulo inclui:

Visão geral da criação e desenvolvimento de adaptadores	13
Criação de conteúdo	13
Desenvolvendo conteúdo de integração	21
Desenvolvendo conteúdo de descoberta	23
Implementar um adaptador de descoberta	25
Etapa 1: Criar um adaptador	27
Etapa 2: Atribuir um trabalho ao adaptador	34
Etapa 3: Criar código Jython	35
Configurar execução de processo remoto	36

Visão geral da criação e desenvolvimento de adaptadores

Antes de começar o planejamento real do desenvolvimento de novos adaptadores, é importante compreender os processos e as interações normalmente associados a esse desenvolvimento.

As seguintes seções podem ajudá-lo a compreender o que você precisa de saber fazer para gerenciar e executar com êxito um projecto de desenvolvimento de descoberta.

Este capítulo

- Têm como pressuposto um conhecimento prático do HP Universal CMDB e familiaridade básica com os elementos do sistema. Seu propósito é ajudar no processo de aprendizagem e não proporciona um manual completo.
- São abordadas as fases de planejamento, pesquisa e implementação de novo conteúdo de descoberta do HP Universal CMDB, juntamente com as diretrizes e as considerações que devem ser levadas em conta.
- Além disso, informações sobre as principais APIs da estrutura do Gerenciamento de Fluxo de Dados são fornecidas. Para ver a documentação completa sobre as APIs disponíveis, consulte *Referência da API de Gerenciamento de Fluxo de Dados do HP Universal CMDB*. (Existem outras APIs informais, porém embora sejam usadas em adaptadores já prontos, podem estar sujeitas à mudança.)

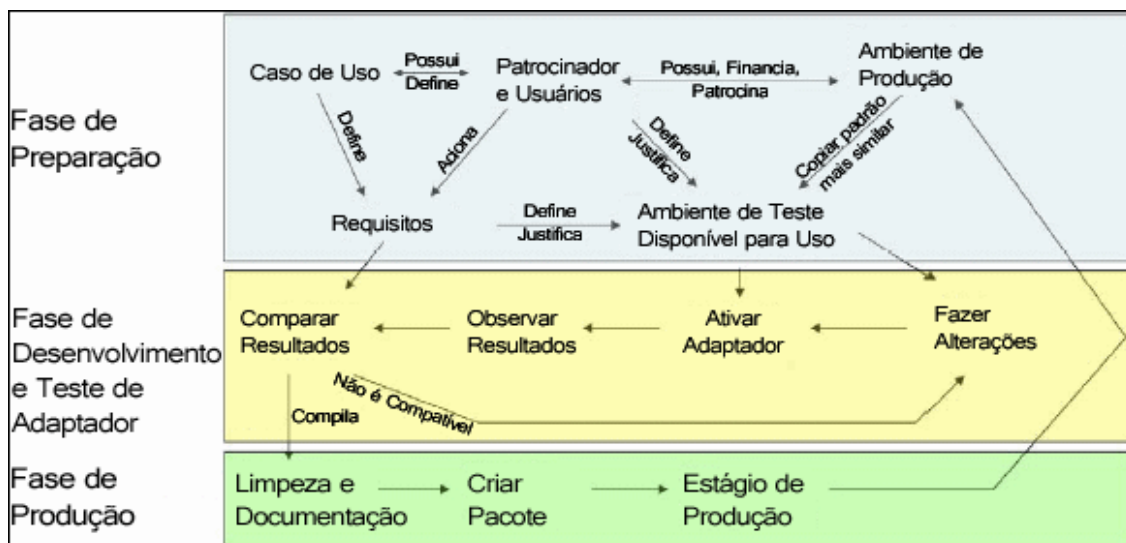
Criação de conteúdo

Esta seção inclui:

- "O ciclo de desenvolvimento do adaptador" abaixo
- "Gerenciamento de fluxo de dados e integração" na página 16
- "Associando valor comercial ao desenvolvimento da descoberta" na página 17
- "Pesquisando requisitos de integração" na página 18

O ciclo de desenvolvimento do adaptador

A seguinte ilustração mostra um fluxograma da escrita de adaptadores. A maior parte do tempo é dedicada à seção intermediária, que representa o loop iterativo do desenvolvimento e teste.



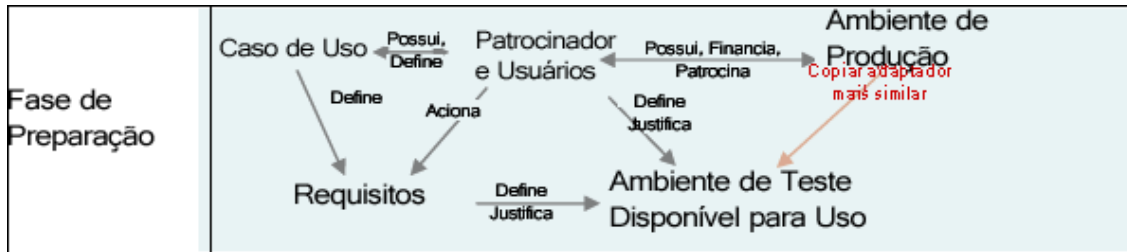
Cada fase do desenvolvimento do adaptador baseia-se na fase anterior.

Quando você estiver satisfeito com o adaptador e o modo como funciona, estará pronto para inseri-lo em um pacote. Usando o Gerenciador de Pacotes do UCMDB ou a exportação manual dos componentes, crie um arquivo *.zip do pacote. É prática recomendada distribuir e testar esse pacote em outro sistema com o UCMDB antes de sua instalação real, para assegurar-se de que todos os componentes estejam presentes e tenham sido inseridos no pacote com êxito. Para ver detalhes sobre a criação de pacotes, consulte "Gerenciador de Pacotes" na página 1 no *Guia de Administração do HP Universal CMDB*.

As seções a seguir descrevem em detalhes cada um das fases e mostram as etapas e melhores práticas essenciais:

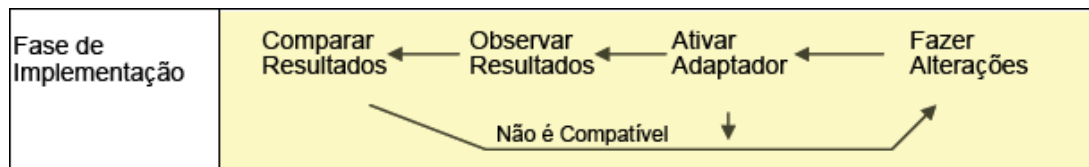
- "Fase de pesquisa e preparação" na página seguinte
- "Desenvolvimento e teste do adaptador" na página seguinte
- "Empacotamento e produtização do adaptador" na página 16

Fase de pesquisa e preparação



A fase de pesquisa e preparação abrange as razões comerciais e os casos de uso, além de esclarecer os requisitos necessários para desenvolver e testar o adaptador.

- Se você planeja alterar um adaptador existente, a primeira etapa técnica é fazer backup desse adaptador e assegurar-se de que é possível retorná-lo ao seu estado original. Se você planeja criar um adaptador novo, copie o adaptador mais similar e salve-o usando um nome apropriado. Para obter detalhes, consulte "[Painel Recursos](#)" na página 1 no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.
- Pesquise como o adaptador deve coletar dados:
 - Use ferramentas/protocolos externos para obter os dados
 - Desenvolva como o adaptador deve criar ECs baseado nos dados
 - Você já sabe como deve ser um adaptador similar
- Determine o adaptador com o maior número de características em comum de acordo com:
 - ECs criados
 - Protocolos usados (SNMP)
 - Tipo de destino (por tipo e versão do sistema operacional, e assim por diante)
- Copie o pacote inteiro.
- Descompacte-o no espaço de trabalho e renomeie os arquivos do adaptador (XML) e Jython (.py).



Desenvolvimento e teste do adaptador

A fase de desenvolvimento e teste do adaptador é um processo altamente iterativo. Quando o adaptador começa a tomar forma, você começa a testá-lo em relação aos casos de uso, faz mudanças, testa outra vez e repete esse processo até que o adaptador atenda aos requisitos.

Início e preparação da cópia

- Altere partes do XML do adaptador: Nome (id) na linha 1, tipos de EC criados e nome do script Jython chamado.

- Verifique se a cópia obtém resultados idênticos ao adaptador original.
- Insira comentários para deixar de fora a maior parte do código, especialmente o código que produz os resultados mais importantes

Desenvolvimento e teste

- Use outro código de amostra para desenvolver mudanças
- Execute-o para testar o adaptador
- Use uma exibição dedicada para validar resultados complexos, e uma pesquisa para validar resultados simples

Empacotamento e produtização do adaptador

A fase de empacotamento e produtização do adaptador é a última fase do desenvolvimento. Como uma melhor prática, deve-se realizar uma fase final de eliminação de elementos, documentos e comentários remanescentes da depuração, para verificar as considerações sobre segurança etc., antes de passar ao empacotamento. Você deve sempre ter pelo menos um documento leia-me para explicar o funcionamento essencial do adaptador. Alguém (ou você mesmo) pode precisar usar esse adaptador posteriormente e qualquer documentação, mesmo que limitada, poderá ser muito útil.

Limpeza e documentação

- Remova a depuração
- Insira comentários para todas as funções e alguns comentários no início da seção principal
- Crie uma TQL e exibição de amostra para o teste de usuários

Criação do pacote

- Exporte adaptadores, TQL etc. para o Gerenciador de Pacotes. Para obter detalhes, consulte "[Gerenciador de Pacotes](#)" na página 1 no *Guia de Administração do HP Universal CMDB*.
- Verifique todas as dependências do seu pacote em relação a outros pacotes, por exemplo, se os ECs criados por aqueles pacotes são ECs de entrada do seu adaptador.
- Use o Gerenciador de Pacotes para criar um arquivo zip do pacote. Para obter detalhes, consulte "[Gerenciador de Pacotes](#)" na página 1 no *Guia de Administração do HP Universal CMDB*.
- Teste a implementação removendo partes do novo conteúdo e reimplementando, ou implementando em outro sistema de teste.

Gerenciamento de fluxo de dados e integração

Os adaptadores do DFM são capazes de realizar a integração com outros produtos. Considere as seguintes definições:

- O DFM coleta conteúdo específico de muitos destinos.
- A integração coleta vários tipos de conteúdo de um sistema.

Observe que essas definições não distinguem os métodos de coleta. O DFM também não faz essa distinção. O processo de desenvolvimento de um novo adaptador é o mesmo usado para desenvolver uma nova integração. Você faz a mesma pesquisa e as mesmas escolhas em relação a adaptadores novos ou existentes, escreve os adaptadores da mesma maneira, e assim por diante. Somente algumas coisas são diferentes:

- A programação do adaptador final. Os adaptadores de integração podem ter de ser executados com mais frequência do que os de descoberta, mas isso depende dos casos de uso.
- ECs de entrada:
 - Integração: acionador que não seja um EC a ser executado sem dados de entrada: uma origem ou nome de arquivo é passado por meio do parâmetro do adaptador.
 - Descoberta: usa ECs comuns do CMDB para a entrada de dados.

Em projetos de integração, você quase sempre reutilizará um adaptador existente. A direção da integração (do HP Universal CMDB a outro produto, ou de outro produto ao HP Universal CMDB) pode afetar sua abordagem do desenvolvimento. Há pacotes específicos disponíveis que você pode copiar para seus próprios usos, usando técnicas comprovadas.

Do HP Universal CMDB para outro projeto:

- Crie uma TQL que produza os ECs e as relações a serem exportados.
- Use um adaptador wrapper genérico para executar a TQL e insira os resultados num arquivo XML que possa ser lido pelo produto externo.

Observação: Para obter exemplos de pacotes do campo, contate Suporte da HP Software.

Para integrar outro produto ao HP Universal CMDB, o adaptador de integração funcionará de acordo com o modo como o outro produto disponibiliza seus dados:

Tipo de integração	Exemplo de referência a reutilizar
Acesso direto ao dados do produto	HP ED
Ler um arquivo .csv ou .xml produzido por uma exportação	HP ServiceCenter
Acesso à API de um produto	BMC Atrium/Remedy

Associando valor comercial ao desenvolvimento da descoberta

O caso de uso do desenvolvimento de novo conteúdo de descoberta deve ser guiado por um caso e um plano de negócios a fim de produzir valor comercial. Isso significa que o objetivo de mapear componentes do sistema a ECs e adicioná-los ao CMDB é criar valor comercial.

O conteúdo talvez não seja sempre usado para mapeamento de aplicativo, contudo essa é uma etapa intermediária em muitos casos de uso. Qualquer que seja o propósito do uso desse conteúdo, seu plano deve responder às seguintes perguntas nessa abordagem:

- Quem é o consumidor? O que o consumidor fará com a informação fornecida pelos ECs (e os relacionamentos entre eles)? Qual é o contexto de negócios em que os ECs e os relacionamentos deverão ser visualizados? Quem é o consumidor desses ECs: uma pessoa, um produto ou ambos?
- Uma vez obtida a combinação perfeita de ECs e relacionamentos no CMDB, como isso será usado para produzir valor comercial?
- Como deve ser o mapeamento perfeito?
 - Que termo descreveria de forma mais significativa os relacionamentos entre ECs?
 - Que tipos de EC seriam os mais importantes a incluir?
 - Que é propósito e usuário final do mapa?
- Como seria o layout perfeito do relatório?

Uma vez estabelecidas as razões comerciais, a próxima etapa é converter o valor comercial em um documento. Isto significa a representação do mapa perfeito usando uma ferramenta de desenho e a compreensão do impacto e das dependências entre ECs, relatórios, registro de mudanças, que mudanças são importantes, monitoração, conformidade e valor comercial adicional segundo as exigências dos casos de uso.

Esse desenho (ou modelo) é chamado de **plano gráfico**.

Por exemplo, caso seja vital que o aplicativo saiba que um determinado arquivo de configuração foi alterado, o arquivo deverá ser mapeado e vinculado ao EC apropriado (ao qual se relaciona) no mapa desenhado.

Trabalhe em conjunto com especialista da área que seja o usuário final do conteúdo desenvolvido. Esse especialista deve indicar as entidades essenciais (ECs com atributos e relacionamentos) que devem existir no CMDB para proporcionar valor comercial.

É possível usar a abordagem de fornecer um questionário ao proprietário do aplicativo (neste caso, o especialista). O proprietário deve poder especificar os objetivos e o plano gráfico descritos acima. O proprietário deve fornecer ao menos a arquitetura atual do aplicativo.

Você deve mapear apenas dados essenciais e nenhum dado desnecessário: você poderá aprimorar o adaptador mais tarde. O objetivo deve ser estabelecer uma descoberta limitada que funcione e proporcione valor. Mapear grandes quantidades de dados produz mapas mais impressionantes, porém seu desenvolvimento pode ser desconcertante e demorado.

Uma vez que o modelo e o valor comercial estejam claros, passe à etapa seguinte. Essa etapa pode ser revisitada à medida que informações mais concretas sejam fornecidas nas etapas seguintes.

Pesquisando requisitos de integração

O pré-requisito dessa etapa é um **plano gráfico** dos ECs e relacionamentos que devem ser descobertos pelo DFM, o que deve incluir os atributos que deverão ser descobertos. Para obter detalhes, consulte "[Visão geral da criação e desenvolvimento de adaptadores](#)" na página 13.

Esta seção inclui os seguintes tópicos:

- ["Alterando um adaptador existente" abaixo](#)
- ["Escrevendo um novo adaptador" abaixo](#)
- ["Pesquisa de modelo" abaixo](#)
- ["Pesquisa da tecnologia" na página seguinte](#)
- ["Diretrizes para escolher modos de acesso aos dados" na página seguinte](#)
- ["Resumo" na página 21](#)

Alterando um adaptador existente

Você pode alterar um adaptador existente quando encontrar um adaptador já disponível no mercado, porém:

- Ele não descobrirá atributos específicos necessários
- Um tipo específico de destino (sistema operacional) talvez não seja descoberto ou seja descoberto incorretamente
- Um relacionamento específico talvez não seja descoberto ou criado

Se você acredita que um adaptador existente faça parte do trabalho (mas não todo o trabalho), sua abordagem deve ser avaliar os adaptadores existentes e verificar se um deles faz quase todo o trabalho necessário; se fizer, você poderá alterá-lo.

Você deve avaliar também se um adaptador de campo existente está disponível. Adaptadores de campo são adaptadores de descoberta que estão disponíveis mas que têm de ser personalizados. Contate o Suporte da HP Software para receber a lista atual de adaptadores de campo.

Escrevendo um novo adaptador

Um adaptador novo precisa ser desenvolvido:

- Quando for mais rápido escrever um adaptador do que introduzir manualmente informações no CMDB (geralmente, cerca de 50 a 100 ECs e relacionamentos) ou quando a tarefa tiver de ser repetida mais de uma vez.
- Quando a necessidade justificar o esforço.
- Se adaptadores predefinidos ou de campo não estiverem disponíveis.
- Se os resultados puderem ser reutilizados.
- Quando o ambiente de destino ou seus dados estiverem disponíveis (você não pode descobrir o que não pode ver).

Pesquisa de modelo

- Consulte o modelo da classe do UCMDb (Gerenciador de Tipo de EC) e faça a correspondência entre as entidades e as relações de seu **plano gráfico** e os TECs existentes. É altamente recomendável aderir ao modelo atual para evitar complicações possíveis durante a atualização de versão. Se você precisar estender o modelo, deverá criar novos TECs, já que uma atualização poderá substituir os TECs predefinidos.
- Se algumas entidades, relações ou atributos estiverem ausentes no modelo atual, você deverá criá-los. É preferível criar um pacote com esses TECs (que posteriormente vão conter toda a descoberta, exibições e outros artefatos relacionados a esse pacote), uma vez que você deverá

ser capaz de implementar esses TECs em cada instalação do HP Universal CMDB.

Pesquisa da tecnologia

Após ter verificado que o CMDB contém os ECs relevantes, a etapa seguinte será decidir como recuperar esses dados dos sistemas relevantes.

Recuperar dados geralmente envolve usar um protocolo para acessar uma porção do gerenciamento do aplicativo, dados reais do aplicativo ou bancos de dados e arquivos de configuração relacionados ao aplicativo. Todas as fontes de dados que possam fornecer informações em um sistema são valiosas. A pesquisa da tecnologia exige o conhecimento profundo do sistema em questão e, às vezes, criatividade.

No caso de aplicativos criados internamente, pode ser útil fornecer um questionário ao proprietário do aplicativo. Nesse questionário, o proprietário deve listar todas as áreas no aplicativo que podem fornecer informações necessárias ao plano gráfico e valor comercial. Essas informações devem incluir (mas não se limitar a) bancos de dados de gerenciamento, arquivos de configuração, arquivos de log, interfaces de gerenciamento, programas da administração, serviços Web, mensagens ou eventos enviados, e assim por diante.

No caso de produtos obtidos no mercado, você deve concentrar-se na documentação, fóruns ou suporte do produto. Procure guias de administração, plug-ins e guias de integração, guias de gerenciamento e assim por diante. Se ainda assim não encontrar os dados das interfaces de gerenciamento, leia sobre os arquivos de configuração do aplicativo, entradas do registro, arquivos de log, logs de evento NT e quaisquer artefatos do aplicativo que controlem sua operação correta.

Diretrizes para escolher modos de acesso aos dados

Importância: Selecione fontes ou uma combinação de fontes que forneçam o maior volume de dados. Se uma única fonte fornecer a maioria das informações, enquanto o resto das informações estiver distribuído ou for inacessível, tente avaliar o valor das informações ausentes em relação ao esforço ou risco de obtê-las. Às vezes, você pode decidir reduzir o plano gráfico se o valor ou custo não justificarem o esforço investido.

Reutilização: Se o HP Universal CMDB já incluir o suporte a um protocolo de conexão específico, essa é uma boa razão para reutilizá-lo. Isso significa que a estrutura do DFM é capaz de fornecer um cliente e configuração já prontos para a conexão. Do contrário, você pode precisar investir no desenvolvimento da infraestrutura. Você pode ver os protocolos de conexão do HP Universal CMDB com suporte atualmente: **Gerenciamento de Fluxo de Dados > Configuração da Sonda de Fluxo de Dados > painel Domínios e Sondas**. Para obter detalhes, consulte [Domains and Probes Pane](#) no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Você pode adicionar protocolos novos adicionando novos ECs ao modelo. Para obter detalhes, contate o Suporte da HP Software.

Observação: Para acessar dados do Registro do Windows, você pode usar o WMI ou NTCmd.

Segurança: O acesso à informação geralmente exige as credenciais (nome e senha de usuário), que são inseridas no CMDB e mantidas seguras no produto. Se possível, e se adicionar medidas de segurança não entrar em conflito com outros princípios definidos, escolha as credenciais ou protocolos que sejam menos sensíveis mas que atendam às necessidades de acesso. Por exemplo, se a informação estiver disponível tanto via a JMX (interface de administração padrão,

limitada) como via Telnet, é preferível usar JMX, pois inerentemente ela fornece acesso limitado e (geralmente) nenhum acesso à plataforma subjacente.

Conveniência: Algumas interfaces de gerenciamento podem incluir recursos mais avançados. Por exemplo, pode ser mais fácil enviar consultas (SQL, WMI) do que navegar árvores de informação ou criar expressões regulares para a análise.

Perfil do desenvolvedor: Desenvolvedores de adaptadores tendem a ter preferência quanto à tecnologia usada. Isso pode ocorrer quando duas tecnologias fornecerem quase a mesma informação a um custo similar em outros fatores.

Resumo

O resultado dessa etapa é um documento que descreve os métodos de acesso e a informação relevante que pode ser extraída por cada método. O documento deve conter também um mapeamento entre as fontes e os dados relevantes do plano gráfico.

Cada método de acesso deve ser marcado de acordo com as instruções acima. Finalmente, você agora tem um plano sobre que fontes descobrir e que informação extrair de cada fonte no modelo de plano gráfico (que agora deve estar mapeado ao modelo do UCMDb correspondente).

Desenvolvendo conteúdo de integração

Antes de criar uma nova integração, você deve compreender as exigências da integração:

- A integração deve copiar dados no CMDB? Os dados devem ser controlados por seu histórico? A fonte de dados é confiável?

A **população** de dados é necessária

- A integração deve federar dados imediatamente para exibições e consultas TQL? A precisão das mudanças feitas nos dados é um fator crucial? A quantidade de dados é grande demais para copiar para o CMDB, mas a quantidade de dados solicitados é geralmente pequena?

A **federação** é necessária.

- A integração deve enviar dados por push a fontes de dados remotas?

O **push de dados** é necessário.

Observação: Os fluxos de federação e população podem ser configurados para a mesma integração, para a flexibilidade máxima.

Para ver detalhes sobre os diferentes tipos de integração, veja "[Integration Studio](#)" na página 1 no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Quatro opções diferentes estão disponíveis para criar adaptadores de integração:

1. Adaptador Jython
 - O padrão de descoberta clássico
 - Escrito em Jython
 - Usado para a população

Para obter detalhes, consulte "[Desenvolvimento de adaptadores Jython](#)" na página 38.

2. Adaptador Java

- Um adaptador que implementa uma das interfaces de adaptador na estrutura de SDK da federação.
- Pode ser usado para umas ou várias tarefas de federação, população ou push de dados (de acordo com a implementação exigida).
- Escrito desde o início em Java, o que permite escrever código para conexão a qualquer fonte ou destino possível.
- Apropriado para os trabalhos que conectam uma única fonte de dados ou destino.

Para obter detalhes, consulte "[Desenvolvimento de adaptadores Java](#)" na página 153.

3. Adaptador de banco de dados genérico

- Um adaptador abstrato baseado no adaptador Java que utiliza a estrutura de SDK de federação.
- Permite a criação de adaptadores que conectam a repositórios de dados externos.
- Dá suporte à federação e à população (com um plugin Java implementado para o suporte às mudanças).
- Relativamente fácil de definir, uma vez que é baseado principalmente em XML e arquivos de configuração de propriedades.
- A configuração principal é baseada em um arquivo **orm.xml** que faz o mapeamento entre classes do UCMDB e colunas de banco de dados.
- Apropriado para trabalhos que conectam uma única fonte de dados.

Para obter detalhes, consulte "[Desenvolvimento de adaptadores de banco de dados genéricos](#)" na página 82.

4. Adaptador push genérico

- Um adaptador abstrato baseado no adaptador Java (estrutura de SDK de federação) e no adaptador Jython.
- Permite a criação de adaptadores que enviam dados por push para destinos remotos.
- Relativamente fácil de definir; você precisa definir somente o mapeamento entre classes do UCMDB e XML, e um script Jython que envie os dados por push para o destino.
- Apropriado para trabalhos que conectam um único destino de dados.
- Usado para o envio por push de dados.

Para obter detalhes, consulte "[Desenvolvimento de adaptadores push](#)" na página 184.

A tabela a seguir mostra as características de cada adaptador:

Fluxo/adaptador	Adaptador Jython	Adaptador Java	Adaptador de BD genérico	Adaptador push
População	X	X	X	

Federação		X	X	
Push de dados		X		X

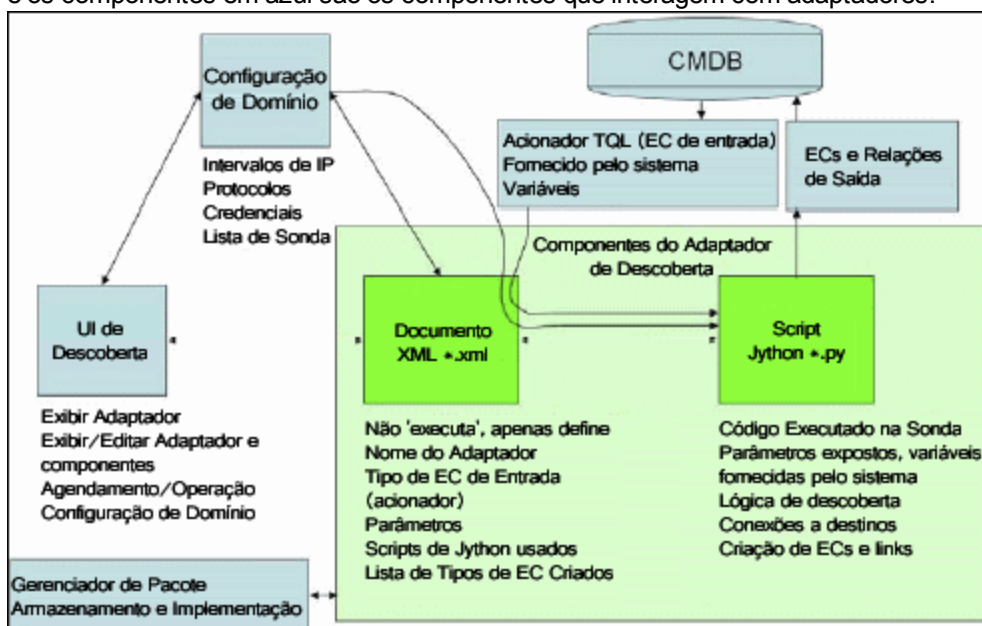
Desenvolvendo conteúdo de descoberta

Esta seção inclui:

- "Adaptadores de descoberta e componentes relacionados " abaixo
- "Separando adaptadores" na página seguinte

Adaptadores de descoberta e componentes relacionados

O diagrama a seguir mostra os componentes de um adaptador e os componentes com os quais eles interagem para executar a descoberta. Os componentes em verde são os adaptadores em si, e os componentes em azul são os componentes que interagem com adaptadores.



Note que a composição mínima de um adaptador é dois arquivos: um documento XML e um script Jython. A estrutura da descoberta, incluindo ECs de entrada, credenciais e bibliotecas fornecidas pelo usuário, é exposta ao adaptador no tempo de execução. Ambos os componentes do adaptador de descoberta são administrados usando o Gerenciamento de Fluxo de Dados. Em termos de operação, eles são armazenados no próprio CMDB; embora o pacote externo permaneça, nenhuma referência é feita a ele durante a operação. O Gerenciador de Pacote permite preservar a nova capacidade de conteúdo de descoberta e integração.

ECs de entrada são fornecidos ao adaptador por uma TQL, e são expostos ao script do adaptador em variáveis fornecidas pelo sistema. Parâmetros do adaptador também são fornecidos como dados de destino, portanto você pode configurar a operação do adaptador de acordo com uma função específica do adaptador.

O aplicativo DFM é usado para criar e testar novos adaptadores. Você usa as páginas Painel de Controle de Descoberta, Gerenciamento do Adaptador e Configuração da Sonda de Fluxo de Dados durante a escrita do adaptador.

Os adaptadores são armazenados e transportados como pacotes. O aplicativo Gerenciador de Pacote e o console JMX são usados para criar pacotes com os adaptadores recém-criados, e para implementar adaptadores em sistemas novos.

Separando adaptadores

Tecnicamente, uma descoberta inteira pode ser definida em um único adaptador. Mas as boas práticas de design exigem que um sistema complexo seja separado em componentes mais simples e mais fáceis de serem gerenciados.

A seguir estão diretrizes e melhores práticas para dividir o processo do adaptador:

- A descoberta deve ser feita em etapas. Cada etapa deve ser representada por um adaptador que deve mapear uma área ou camada do sistema. Os adaptadores devem depender da descoberta feita na etapa ou camada precedente para continuar a descoberta do sistema. Por exemplo, o adaptador A é acionado pelo resultado de uma TQL de servidor de aplicativos e faz o mapeamento da camada de servidor de aplicativos. Como parte desse mapeamento, um componente de conexão JDBC é mapeado. O adaptador B registra um componente de conexão JDBC como uma TQL acionadora e usa os resultados do adaptador A para acessar a camada do banco de dados (por exemplo, usando o atributo URL do JDBC) e faz o mapeamento da camada de banco de dados.
- **O paradigma da conexão em duas etapas:** A maioria de sistemas exige credenciais para o acesso aos seus dados. Isso significa que uma combinação de usuário/senha precisa ser inserida nesses sistemas. O administrador do DFM fornece ao sistema as informações de credenciais de uma maneira segura, e pode dar diversas credenciais de logon prioritizadas. Isso é chamado de **dicionário de protocolo**. Se o sistema não for acessível (seja qual for o motivo) não há razão para executar uma descoberta adicional. Se a conexão for bem-sucedida, é preciso que exista uma maneira de indicar que credenciais foram usadas com êxito, para garantir o acesso futuro da descoberta.

Essas duas fases resultam na separação dos dois adaptadores nos seguintes casos:

- **Adaptador de conexão:** Esse é um adaptador que aceita um acionador inicial e procura por um agente remoto naquele acionador. Ele faz isso testando todas as entradas no dicionário de protocolo que correspondam ao tipo desse agente. Se for bem-sucedido, esse adaptador fornece como resultado um EC de agente remoto (SNMP, WMI etc.), que também aponta para a entrada correta no dicionário de protocolo para garantir conexões futuras. Esse EC de agente torna-se então parte de um acionador do adaptador de conteúdo.
- **Adaptador de conteúdo:** O pré-requisito desse adaptador é a conexão bem-sucedida do adaptador precedente (pré-requisito especificado pelas TQLs). Esses tipos de adaptador já não precisam mais pesquisar o dicionário de protocolo inteiro, pois têm uma maneira de obter as credenciais corretas do EC de agente remoto e usá-las para fazer logon no sistema descoberto.
- Considerações de programação diferentes também podem influenciar a divisão da descoberta. Por exemplo, um sistema pode ser consultado somente fora do horário comercial, portanto mesmo que faça sentido juntar esse adaptador a outro adaptador de descoberta de outro

sistema, a diferença na programação significa que você deve criar dois adaptadores.

- A descoberta de interfaces de gerenciamento diferentes ou tecnologias diferentes de descoberta do mesmo sistema devem ser colocadas em adaptadores separados. Isso permite ativar o método de acesso apropriado para cada sistema ou organização. Por exemplo, algumas organizações têm o acesso WMI às máquinas mas não têm agentes SNMP instalados nelas.

Implementar um adaptador de descoberta

Uma tarefa do DFM tem como objetivo acessar sistemas remotos (ou locais), modelando dados extraídos como ECs, e salvar os ECs no CMDB. A tarefa consiste nas seguintes etapas:

1. Criar um adaptador.

Você configura um arquivo de adaptador que contém o contexto, os parâmetros e os tipos de resultado selecionando os scripts que deverão fazer parte do adaptador. Para obter detalhes, consulte ["Etapa 1: Criar um adaptador" na página 27](#).

2. Criar um assistente de Descoberta.

Você configura um trabalho com informações de programação e uma consulta acionadora. Para obter detalhes, consulte ["Etapa 2: Atribuir um trabalho ao adaptador" na página 34](#).

3. Editar código de descoberta.

Você pode editar o Jython ou o código Java contido nos arquivos do adaptador e que se refere à estrutura do DFM. Para obter detalhes, consulte ["Etapa 3: Criar código Jython" na página 35](#).

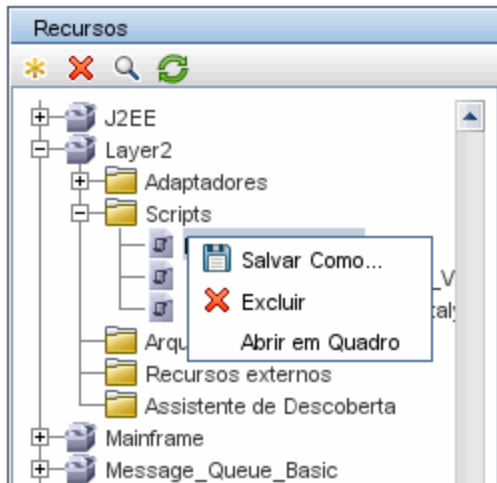
Para escrever novos adaptadores, você cria os componentes acima, e cada um deles é automaticamente associado ao componente da etapa precedente. Por exemplo, após você criar um trabalho e selecionar o adaptador relevante, o arquivo do adaptador será associado ao trabalho.

Código do adaptador

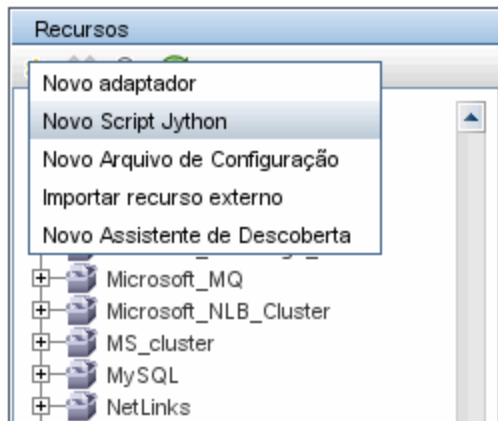
A implementação da conexão ao sistema remoto, a consulta aos seus dados e o seu mapeamento como dados do CMDB são executados pelo código Jython. Por exemplo, o código contém a lógica para a conexão ao banco de dados e a extração de seus dados. Nesse caso, o código espera receber uma URL do JDBC, nome de usuário, senha, portar e assim por diante. Esses parâmetros são específicos para cada instância do banco de dados que responde à consulta TQL. Você define essas variáveis no adaptador (nos dados do EC acionador) e quando o trabalho é executado, esses detalhes específicos são passados ao código para a execução.

O adaptador pode fazer referência a esse código por um nome de classe Java ou um nome de script Jython. Nesta seção, discutiremos como escrever código do DFM na forma de scripts Jython.

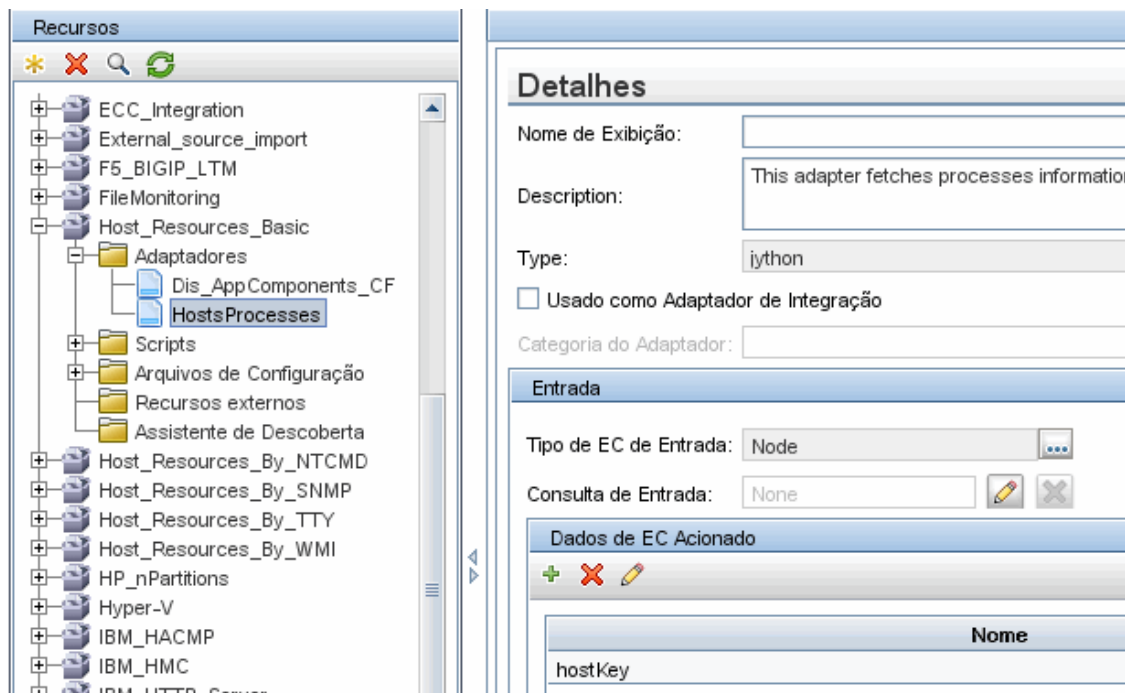
Um adaptador pode conter uma lista de scripts a serem usados ao executar uma descoberta. Ao criar um novo adaptador, você geralmente cria um novo script e o atribui ao adaptador. Um novo script inclui modelos básicos, mas você pode usar um dos outros scripts como modelos clicando com o botão direito do mouse e selecionando **Salvar como**:



Para ver detalhes sobre como escrever novos scripts Jython, consulte "[Etapa 3: Criar código Jython](#)" na página 35. Você adiciona scripts no painel Recursos:



Os scripts na lista são executados um após o outro, na ordem em que são definidos no adaptador:



Observação: Um script deve ser especificado mesmo que esteja sendo usado unicamente como uma biblioteca por outro script. Nesse caso, o script de biblioteca deve ser definido antes do script que vai utilizá-lo. Nesse exemplo, o script `processdbutils.py` é uma biblioteca usada pelo último script `host_processes.py`. As bibliotecas se diferenciam de scripts executáveis comuns pela ausência da função `DiscoveryMain()`.

Etapa 1: Criar um adaptador

Um adaptador pode ser considerado como a definição de uma função. Essa função determina uma definição da entrada, executa a lógica na entrada, define a saída e fornece um resultado.

Cada adaptador especifica a entrada e a saída: A entrada e a saída são ECs acionadores definidos especificamente no adaptador. O adaptador extrai dados do EC acionador de entrada e passa esses dados como parâmetros ao código. (Às vezes, os dados de ECs relacionados também são passados ao código. Para ver detalhes, consulte "[Janela ECs Relacionados](#)" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.) Com a exceção desses parâmetros de entrada específicos do EC acionador que são passados a ele, o código de um adaptador é genérico.

Para ver detalhes sobre componentes de entrada, consulte "[Conceitos do Universal Discovery](#)" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Esta seção inclui os seguintes tópicos:

- "[Definir a entrada do adaptador \(TEC acionador e consulta de entrada\)](#)" na página seguinte
- "[Definir saída do adaptador](#)" na página 31
- "[Substituir parâmetros do adaptador](#)" na página 32

- ["Substituir seleção de sonda - Opcional" na página 33](#)
- ["Configurar um classpath para um processo remoto - Opcional" na página 34](#)

1. Definir a entrada do adaptador (TEC acionador e consulta de entrada)

Você usa os componentes TEC acionador e consulta de entrada para definir ECs específicos como entrada de dados do adaptador:

- O TEC acionador define que TEC será usado como entrada do adaptador. Por exemplo, no caso de um adaptador de descoberta de IPs, o TEC de entrada é Rede.
- A consulta de entrada é uma consulta comum, editável e que define a consulta a ser executada no CMDB. A consulta de entrada define limitações adicionais no TEC (por exemplo, se a tarefa exige um `hostID` ou um atributo `application_ip`) e pode definir mais dados de EC se necessário ao adaptador.

Se o adaptador exigir informações adicionais dos ECs relacionados ao EC acionador, você poderá acrescentar nós adicionais à TQL de entrada. Para obter detalhes, consulte ["Exemplo de definição de consulta de entrada:" na página seguinte a seguir e "Adicionar nós de consulta e relacionamentos a uma consulta TQL" no Guia de Modelagem do HP Universal CMDB.](#)

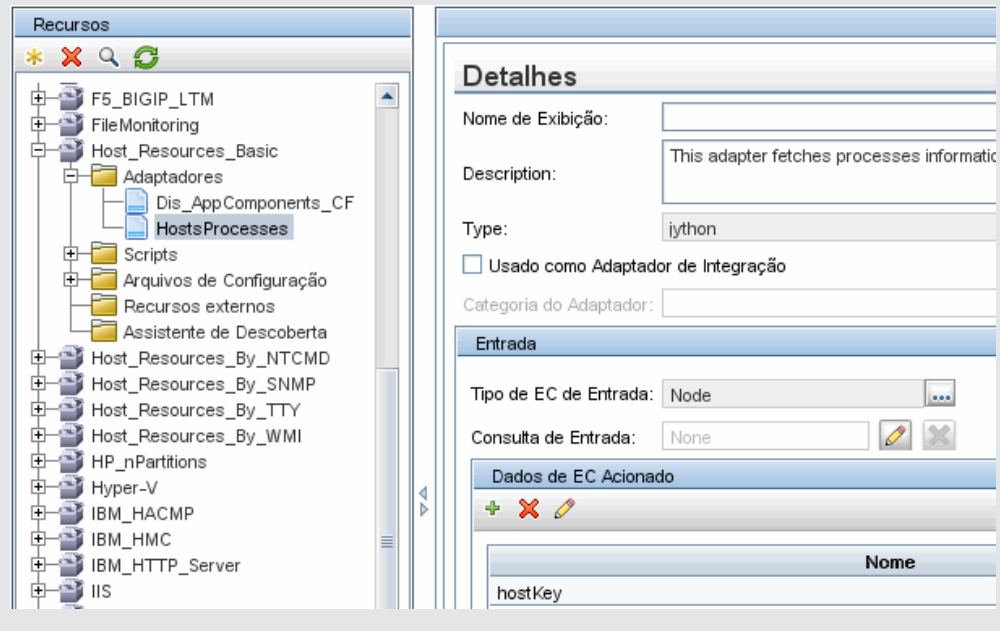
- Os dados do EC acionador contêm todas as informações necessárias no EC acionador, assim como informações sobre outros nós na TQL de entrada, caso tenham sido definidos. O DFM usa variáveis para recuperar dados dos ECs. Quando a tarefa é baixada na Sonda, as variáveis de dados do EC acionador são substituídas pelos valores reais que existem nos atributos de instâncias reais de EC.

Exemplo da definição de TEC acionador:

Neste exemplo, um TEC acionador define que ECs de IP são permitidos no adaptador.

- a. Acesse **Gerenciamento de Fluxo de Dados > Gerenciamento do Adaptador**. Selecione o adaptador **HostProcesses (Pacotes > Host_Resources_Basic > Adaptadores > HostProcesses)**.
- b. Encontre a caixa Tipo de EC de Entrada. Para obter detalhes, consulte ["Guia Definição do Adaptador"](#) no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.
- c. Clique no botão para abrir a caixa de diálogo Escolher Classe Descoberta. Para ver detalhes, consulte ["Caixa de diálogo Escolher Classe Descoberta"](#), no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.
- d. Selecione o TEC.

Neste exemplo, o EC IP (Host) é permitido no adaptador:

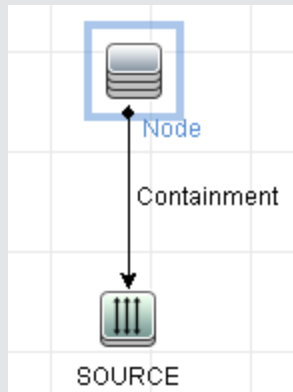


Exemplo de definição de consulta de entrada:

Neste exemplo, a consulta TQL de entrada define que o EC IpAddress (configurado no exemplo anterior como o TEC acionador) deve ser conectado a um EC Node.

- Acesse **Gerenciamento de Fluxo de Dados > Gerenciamento do Adaptador**. Encontre a caixa TQL de Entrada. Clique no botão **Editar** para abrir o Editor de TQL de Entrada. Para obter detalhes, consulte "[Janela Editor de Consulta de Entrada](#)" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.
- No Editor de TQL de Entrada, nomeie o nó **ORIGEM** do EC acionador: clique com o botão direito do mouse no nó e escolha **Propriedades do Nó de Consulte**. Na caixa **Nome de Elemento**, mude o nome para **ORIGEM**.
- Adicione um EC **Node** e um relacionamento **Containment** ao EC IpAddress. Para ver detalhes sobre como trabalhar com o Editor de TQL de Entrada, consulte "[Janela Editor de Consulta de Entrada](#)" no *Guia de Gerenciamento de Fluxo de Dados do HP*

Universal CMDB.



O EC **IpAddress** é conectado a um EC **Node**. O TQL de entrada consiste em dois nós, **Node** e **IpAddress**, os quais estão vinculados. O EC **IpAddress** é chamado de **SOURCE**.

Exemplo da adição de variáveis à consulta TQL de entrada:

Neste exemplo, você adiciona as variáveis `DIRECTORY` e `CONFIGURATION_FILE` à consulta TQL de entrada criada no exemplo anterior. Essas variáveis ajudam a definir o que deve ser descoberto: neste caso, os arquivos de configuração que residem nos hosts vinculados aos IPs que você precisa descobrir.

- a. Exiba a TQL de entrada criada no exemplo anterior.

Acesse **Gerenciamento de Fluxo de Dados > Gerenciamento do Adaptador**. Encontre o painel Dados de EC Acionado. Para obter detalhes, consulte "[Guia Definição do Adaptador](#)" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

- b. Adicione variáveis à TQL de entrada. Para obter detalhes, acesse **Gerenciamento de Fluxo de Dados > Gerenciamento do Adaptador**. Encontre o painel Dados de EC Acionado. Para ver detalhes, consulte o campo Variáveis em "[Guia Definição do Adaptador](#)" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

A imagem mostra uma janela de diálogo intitulada 'Editor de Parâmetros'. Ela contém três campos de entrada: 'Nome' (um campo de texto curto), 'Valor' (um campo de texto longo) e 'Descrição' (um campo de texto longo). Na parte inferior da janela, há dois botões: 'OK' e 'Cancelar'.

Exemplo da substituição de variáveis por dados reais:

Nesse exemplo, as variáveis substituem os dados do EC **IpAddress** pelos valores verdadeiros das instâncias reais do EC **IpAddress** em seu sistema.

Os dados de EC acionado do EC **IpAddress** incluem uma variável `fileName`. Essa variável permite a substituição do nó **CONFIGURATION_DOCUMENT** na TQL de entrada pelos valores reais do arquivo de configuração situado em um host:

Nome	Valor
filename	\${SOURCE.credential_id}
hostKey	\${SOURCE.host_key}

Os dados do EC acionador são carregados na Sonda com todas as variáveis substituídas por valores reais. O script do adaptador inclui um comando para usar **DFM Framework** para recuperar os valores reais das variáveis definidas:

```
Framework.getTriggerCIData ("ip_address ")
```

As variáveis `fileName` e `path` usam os atributos `data_name` e `document_path` do nó **CONFIGURATION_DOCUMENT** (definido no Editor de Consulta de Entrada – consulte o exemplo anterior).

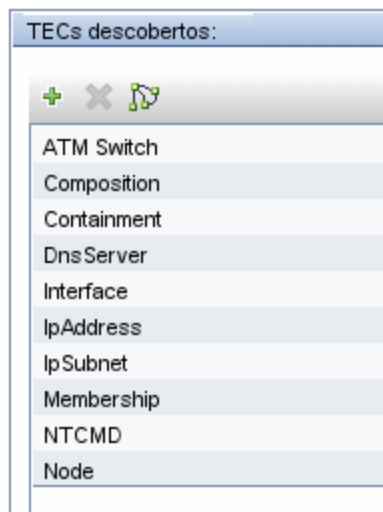
Clique [aqui](#) para ver uma ilustração.

As variáveis `Protocol`, `credentialsId` e `ip_address` usam os atributos `root_class`, `credentials_id` e `application_ip`:

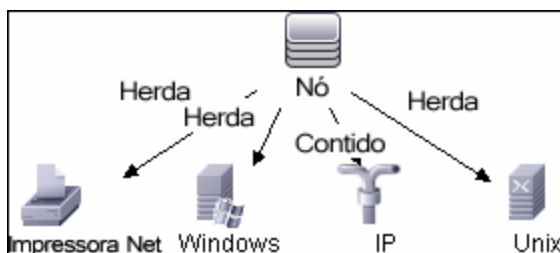
Ch...	≡ Nome de Exibição	Nome	Tipo	Descrição	Valor Padrão	Visível
	Create Time	create_time	date	When was ...		✓
	Created By	data_source	string			✓
	credentials_id	Reference	string	Reference ...		✓
?	Deletion Candidate ...	root_deletioncandida...	integer	What is the...	20	✓
	Description	description	string	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	✓

2. Definir saída do adaptador

A saída do adaptador é uma lista de ECs descobertos (**Gerenciamento de Fluxo de Dados > Gerenciamento do Adaptador > guia Definição do Adaptador > TECs descobertos**) e os vínculos entre eles:



Você pode ver o TECs também como um mapa da topologia, isto é, os componentes e a maneira em que estão vinculados junto (clique no botão **Exibir TECs Descobertos como um Mapa**):



Os ECs descobertos são retornados pelo código do DFM (isto é, o script Jython) no formato `ObjectStateHolderVector` do UCMDB. Para obter detalhes, consulte ["Geração de resultados pelo script Jython"](#) na página 43.

Exemplo da saída do adaptador:

Neste exemplo, você define que TECs farão parte da saída do EC IP.

- Acesse **Gerenciamento de Fluxo de Dados > Gerenciamento do Adaptador**.
- No painel Recursos, selecione **Rede > Adaptadores > NSLOOKUP_on_Probe**.
- Na guia Definição do Adaptador, localize o painel TECs Descobertos.
- Os TECs que devem fazer parte da saída do adaptador estarão listados. Adicione ou remova TECs da lista. Para obter detalhes, consulte ["Guia Definição do Adaptador"](#) no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

3. Substituir parâmetros do adaptador

Para configurar um adaptador para mais de um trabalho, você pode substituir parâmetros de adaptador. Por exemplo, o adaptador `SQL_NET_Dis_Connection` é usado pelos trabalhos `MSSQL Connection by SQL` e `Oracle Connection by SQL`.

Exemplo da substituição de um parâmetro do adaptador:

Este exemplo ilustra a substituição de um parâmetro do adaptador de modo que um adaptador possa ser usado para descobrir bancos de dados do Microsoft SQL Server e Oracle.

- a. Acesse **Gerenciamento de Fluxo de Dados > Gerenciamento do Adaptador**.
- b. No painel Recursos, selecione **Database_Basic > Adaptadores > SQL_NET_Dis_Connection**.
- c. Na guia Definição do Adaptador, localize o painel **Parâmetros do Adaptador**. O parâmetro `protocolType` tem o valor **todos**:

Nome	Valor
protocolType	all

- d. Clique com botão direito do mouse no adaptador **SQL_NET_Dis_Connection_MsSql** e escolha **Ir para o Trabalho de Descoberta > MSSQL Connection by SQL**.
- e. Exiba a guia Propriedades. Localize o painel Parâmetros:

Substituir	Nome	Valor
<input checked="" type="checkbox"/>	protocolType	MicrosoftSQLServer

O valor `todos` é substituído pelo valor `MicrosoftSQLServer`.

Observação: O trabalho **Oracle Connection by SQL** inclui o mesmo parâmetro porém o valor é substituído por um valor Oracle.

Para ver detalhes sobre a adição, exclusão ou edição de parâmetros, consulte "[Guia Definição do Adaptador](#)" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

O DFM começa a procurar por instâncias do Microsoft SQL Server de acordo com esse parâmetro.

4. Substituir seleção de sonda - Opcional

No servidor do UCMDb, há um mecanismo de distribuição que leva os ECs recebidos pelo UCMDb e escolhe automaticamente qual sonda deve executar o trabalho para cada EC acionador de acordo com uma das opções a seguir.

- **Para o tipo de EC de endereço IP:** adote a sonda definida para esse IP.
- **Para o tipo de EC de software em execução:** use os atributos `application_ip` e `application_ip_domain` e escolha a sonda definida para o IP no domínio relevante.
- **Para outros tipos de EC:** adote o IP do nó de acordo com o nó relacionado do EC (se ele existir).

A seleção de sonda automática é feita de acordo com o nó relacionado do EC. Após obter o nó relacionado ao EC, o mecanismo de distribuição escolhe um dos IPs do nó e escolhe a sonda de acordo com as definições do escopo de rede da Sonda.

Nos casos a seguir, você precisa especificar a sonda manualmente e não usar o mecanismo de distribuição automático:

- Você já sabe qual sonda deve ser executada para o adaptador e não precisa do mecanismo de distribuição automático para selecionar a sonda (por exemplo, se o EC acionador for o gateway da sonda).
- A seleção de sonda automática pode falhar. Isso pode acontecer nas seguintes situações:
 - Um EC acionador não possui um nó relacionado (como o TEC de rede).
 - Um nó do EC acionador possui IPs múltiplos, cada um pertencente a uma sonda diferente.

Para resolver esses problemas, você pode especificar a sonda que deve ser usada com o adaptador, como a seguir:

- a. Na seção Seleção de Sonda, selecione **Substituir seleção de sonda padrão** como mostrado abaixo.

A imagem mostra uma janela de configuração de software com o título "Definição do Adaptador" e "Configuração do Adaptador". A janela está dividida em duas seções principais:

- Seleção de Sonda:** Possui um checkbox selecionado com o rótulo "Substituir seleção de sonda padrão". Ao lado dele há um campo de texto contendo o código de substituição "\${SOURCE.name}".
- Opções de Execução:** Possui quatro controles de usuário:
 - "Criar log de comunicação:" com um menu suspenso atualmente em "Em falhas".
 - "Incluir resultados no log de comunicação:" com botões de opção "Sim" e "Não", onde "Não" está selecionado.
 - "Máx. de threads:" com um campo de entrada de texto vazio.
 - "Tempo máx. de execução:" com um campo de entrada de texto vazio.

- b. Na caixa Sonda, digite a sonda que deve ser usada para a tarefa.

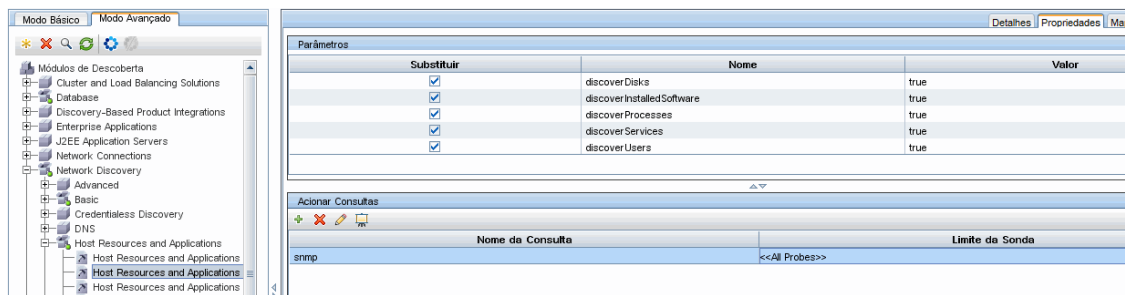
5. Configurar um classpath para um processo remoto - Opcional

Para obter detalhes, consulte "[Configurar execução de processo remoto](#)" na página 36.

Etapa 2: Atribuir um trabalho ao adaptador

Cada adaptador tem um ou mais trabalhos associados que definem a política de execução. Os trabalhos permitem o agendamento do mesmo adaptador de modo diferente de acordo com conjuntos diferentes de ECs acionados, e também permitem passar parâmetros diferentes a cada conjunto.

Os trabalhos aparecem na árvore Módulos de Descoberta, e essa é a entidade que o usuário ativa, conforme a imagem abaixo.



Escolher um TQL acionador

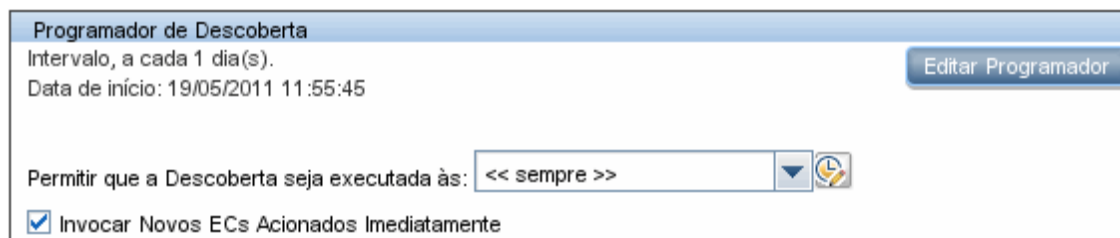
Cada trabalho é associado a TQLs acionadoras. Essas TQLs acionadoras publicam os resultados que são usados como ECs acionadores de entrada para o adaptador desse trabalho.

Uma TQL acionadora pode adicionar restrições a uma TQL de entrada. Por exemplo, se os resultados de uma TQL de entrada forem IPs conectados ao SNMP, os resultados de uma TQL acionadora podem ser IPs conectados ao SNMP no intervalo 195.0.0.0-195.0.0.10.

Observação: Uma TQL acionadora deve fazer referência aos mesmos objetos aos quais a TQL de entrada se refere. Por exemplo, se uma TQL de entrada fizer uma consulta quanto aos IPs que executam SNMP, você não poderá definir uma TQL acionadora (para o mesmo trabalho) para fazer uma consulta quanto a IPs conectados a um host, porque alguns dos IPs podem não estar conectados a um objeto SNMP, conforme necessário à TQL de entrada.

Definir informações de agendamento

A informação de programação da Sonda especifica quando executar o código em ECs acionadores. Se a caixa de seleção **Invocar Novos ECs Acionados Imediatamente** estiver marcada, o código também será executado uma vez em cada EC acionador quando alcançar a Sonda, qualquer que sejam as configurações de programações futuras.



Para cada ocorrência programada de cada trabalho, a Sonda executa o código em todos os ECs acionadores acumulados do trabalho. Para obter detalhes, consulte [Discovery Scheduler Dialog Box](#) no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Substituir parâmetros do adaptador

Ao configurar um trabalho, você pode substituir os parâmetros do adaptador. Para obter detalhes, consulte "[Substituir parâmetros do adaptador](#)" na página 32.

Etapa 3: Criar código Jython

HP Universal CMDB O usa scripts Jython para escrita no adaptador. Por exemplo, os scripts `SNMP_Connection.py` são usados pelo adaptador `SNMP_NET_Dis_Connection` para testar e

conectar a máquinas usando SNMP. Jython é uma linguagem baseada em Python e desenvolvida em Java.

Para ver detalhes sobre como trabalhar em Jython, consulte estes sites:

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

Para obter detalhes, consulte "Criar código Jython" na página 38.

Configurar execução de processo remoto

Você pode executar a descoberta para um trabalho de descoberta em um processo separado do processo da Sonda de Fluxo de Dados.

Por exemplo, você pode executar o trabalho em um processo remoto separado se o trabalho usa bibliotecas jar que são de uma versão diferente das bibliotecas da sonda ou incompatíveis com as bibliotecas da sonda.

Você também pode executar o trabalho em um processo remoto separado se o trabalho potencialmente consumir muita memória (apresentar muitos dados) e você deseja isolar a sonda da possíveis problemas OutOMemory.

Para configurar um trabalho para ser executado como um processo remoto, defina os seguintes parâmetros no arquivo de configuração do adaptador:

Parâmetro	Descrição
remoteJVMArgs	Parâmetros JVM para o processo de Java remoto.
runInSeparateProcess	Quando definido como verdadeiro , o trabalho de descoberta será executado em um processo separado.
remoteJVMClasspath	<p>(Opcional) Permite a personalização do classpath do processo remoto, substituindo o classpath da sonda padrão. Isso é útil se houver uma incompatibilidade de versão entre os jars da sonda e os jars personalizados necessários para a descoberta definida pelo cliente.</p> <p>Se o parâmetro remoteJVMClasspath não estiver definido ou estiver vazio, o classpath da sonda padrão será usado.</p> <p>Se você desenvolver um novo trabalho de descoberta e você deseja garantir que a versão da biblioteca do jar da sonda não colida com as bibliotecas do jar do trabalho, você deve usar pelo menos o classpath mínimo necessário para executar a descoberta básica. O classpath mínimo é definido no arquivo DataFlowProbe.properties no parâmetro basic_discovery_minimal_classpath.</p> <p>Exemplos de personalização de remoteJVMClasspath:</p> <ul style="list-style-type: none"> • Para pré-anexar ou anexar jars personalizados ao classpath da sonda padrão, personalize o parâmetro remoteJVMClasspath como a seguir:

Parâmetro	Descrição
	<p><code>custom1.jar;%classpath%;custom2.jar -</code></p> <p>Nesse caso, custom1.jar é colocado antes do classpath da sonda padrão e custom2.jar é anexado ao classpath da sonda.</p> <ul style="list-style-type: none">• Para usar o classpath mínimo, personalize o parâmetro remoteJVMClasspath como a seguir: <p><code>custom1.jar;%minimal_classpath%;custom2.jar</code></p>

Capítulo 2

Desenvolvimento de adaptadores Jython

Este capítulo inclui:

Referência da API de Gerenciamento de Fluxo de Dados da HP	38
Criar código Jython	38
Oferecer suporte a localização em adaptadores Jython	49
Trabalhar com o Analisador de Descoberta	56
Executar o Analisador de Descoberta a partir do Eclipse	63
Registrar código do DFM	72
Bibliotecas e utilitários Jython	73

Referência da API de Gerenciamento de Fluxo de Dados da HP

Para ver a documentação completa sobre as APIs disponíveis, consulte *Referência da API de Gerenciamento de Fluxo de Dados da HP*. Esses arquivos estão localizados na seguinte pasta:

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

Criar código Jython

HP Universal CMDB O usa scripts Jython para escrita no adaptador. Por exemplo, os scripts `SNMP_Connection.py` são usados pelo adaptador `SNMP_NET_Dis_Connection` para testar e conectar a máquinas usando SNMP. Jython é uma linguagem baseada em Python e desenvolvida em Java.

Para ver detalhes sobre como trabalhar em Jython, consulte estes sites:

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

A seção a seguir descreve a escrita real de código Jython no DFM Framework. Esta seção trata especificamente dos pontos de contato entre o script Jython e o Framework que ele chama, além de descrever também as bibliotecas e utilitários Jython que devem ser usados sempre que possível.

Observação:

- Os scripts escritos para o DFM devem ser compatíveis com Jython versão 2.1.
- Para ver a documentação completa sobre as APIs disponíveis, consulte *Referência da API de Gerenciamento de Fluxo de Dados da HP*.

Esta seção inclui os seguintes tópicos:

- "Usar arquivos JAR Java externos em Jython" abaixo
- "Execução do código" abaixo
- "Modificando scripts prontos" abaixo
- "Estrutura do arquivo Jython" na página seguinte
- "Geração de resultados pelo script Jython" na página 43
- "Instância do Framework" na página 44
- "Localizando as credenciais corretas (para adaptadores de conexão)" na página 47
- "Manipulando exceções de Java" na página 48

Usar arquivos JAR Java externos em Jython

Ao implantar novos scripts Jython, as bibliotecas Java externas (arquivos JAR) ou os arquivos executáveis de terceiros às vezes são necessários como arquivos de utilitários Java, arquivos de conexão (por exemplo, arquivos JAR de Driver JDBC) ou arquivos executáveis (por exemplo, o **nmap.exe** é usado para descoberta sem credenciais).

Esses recursos devem ser reunidos no pacote na pasta **External Resources**. Qualquer recurso colocado nessa pasta é enviado automaticamente para qualquer Sonda que se conecta ao servidor HP Universal CMDB.

Além disso, quando a descoberta é iniciada, qualquer recurso de arquivo JAR é carregado no caminho de classe de Jython, tomando todas as classes contidas nele disponíveis para importação e utilização.

Execução do código

Depois que um trabalho é ativado, uma tarefa com todas as informações necessárias é baixada para a Sonda.

A Sonda começa a executar o código do DFM usando as informações especificadas na tarefa.

O fluxo de código Jython começa a ser executado de uma entrada principal no script, executa o código para descobrir ECs e fornece os resultados de um vetor de ECs descobertos.

Modificando scripts prontos

Ao fazer modificações em script prontos, faça somente alterações mínimas no script e coloque quaisquer métodos necessários em um script externo. Você pode controlar as alterações com mais eficiência e, ao migrar para uma versão mais recente do HP Universal CMDB, seu código não será substituído.

Por exemplo, a seguinte linha única de código em um script pronto chama um método que calcula um nome de servidor Web de uma maneira específica de aplicativo:

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

A lógica mais complexa que decide como calcular esse nome está contida em um script externo:

```
# implement customer specific processing for 'servername' attribute of  
httpplugin  
#  
def PlugInProcessing(servername, transportHN, mam_utils_handle):  
    # support application-specific HTTP plug-in naming  
    if servername == "appsrv_instance":  
        # servername is supposed to match up with the j2ee  
server name, however some groups do strange things with their  
        # iPlanet plug-in files. this is the best work-around  
we could find. this join can't be done with IP address:port  
        # because multiple apps on a web server share the same  
IP:port for multiple websphere applications  
        logger.debug('httpcontext_webapplicationserver  
attribute has been changed from [' + servername + '] to [' +  
transportHN[:5] + '] to facilitate websphere enrichment')  
        servername = transportHN[:5]  
    return servername
```

Salve o script externo na pasta External Resources. Para obter detalhes, consulte "[Painel Recursos](#)" na página 1 no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*. Se você adicionar esse script a um pacote, poderá usar esse script para outros trabalhos também. Para ver detalhes sobre como trabalhar com o Gerenciador de Pacotes, consulte "[Gerenciador de Pacotes](#)" na página 1 no *Guia de Administração do HP Universal CMDB*.

Durante a atualização, a alteração feita na única linha de código é substituída pela nova versão do script pronto, por isso será necessário substituir a linha. Entretanto, o script externo não será substituído.

Estrutura do arquivo Jython

O arquivo Jython consiste em três partes em uma ordem específica:

1. Importações
2. Função Principal - [DiscoveryMain](#)
3. Definições de funções (opcional)

Veja a seguir um exemplo de um 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
```

Importações

As classes Jython são espalhadas pelos namespaces hierárquicos. Na versão 7.0 ou posterior, diferentemente das versões anteriores, não há importações implícitas e cada classe usada precisa ser importada explicitamente. (Essa alteração foi feita para melhorar o desempenho e permitir melhor entendimento do script Jython porque não oculta os detalhes necessários.)

- Para importar um script Jython:

```
importar agente de log
```

- Para importar uma classe Java:

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

Função principal – DiscoveryMain

Cada arquivo de script executável Jython contém uma função principal: [DiscoveryMain](#).

A função `DiscoveryMain` é a principal entrada do script e a primeira função executada. A função principal pode chamar outras funções definidas nos scripts:

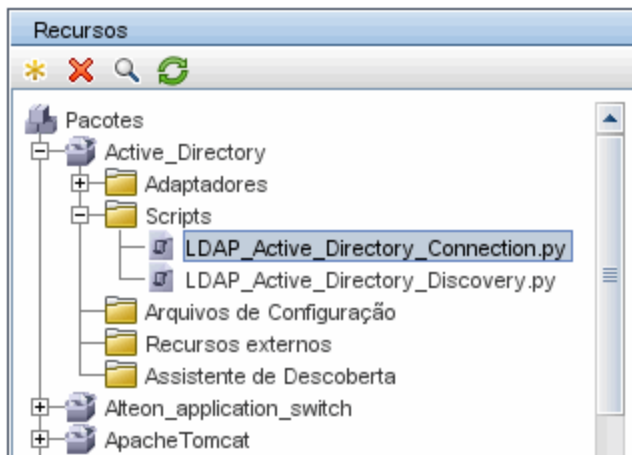
```
def DiscoveryMain(Framework):
```

O argumento `Framework` precisa ser especificado na definição da função principal. Esse argumento é usado pela função principal para recuperar informações necessárias à execução dos scripts (como informações sobre o EC Acionador e os parâmetros) e também serve para relatar erros que ocorrem durante a execução do script.

Você pode criar um script Jython sem nenhum método principal. Esses scripts são usados como scripts de biblioteca chamados a partir de outros scripts.

Definição de funções

Cada script pode conter funções adicionais chamadas a partir do código principal. Cada função dessas pode chamar outra, que existe no script atual ou em outro script (use a instrução `import`). Observe que, para usar outro script, você precisa adicioná-lo à seção `Scripts` do pacote:



Exemplo de uma função que chama outra função:

No exemplo a seguir, o código principal chama o método `doQueryOSUsers(..)` que chama um método interno `doOSUserOSH(..)`:

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

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

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

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

Se esse script for uma biblioteca global relevante para muitos adaptadores, você poderá adicioná-lo à lista de scripts no arquivo de configuração `jythonGlobalLibs.xml`, em vez de adicioná-lo a

cada adaptador (**Gerenciamento do Adaptador > Painel Recursos > AutoDiscoveryContent > Arquivos de Configuração**).

Geração de resultados pelo script Jython

Cada script Jython é executado em um EC Acionador específico e termina com os resultados retornados pelo valor de retorno da função `DiscoveryMain`.

O resultado do script na verdade é um grupo de ECs e links que serão inseridos ou atualizados no CMDB. O script retorna esse grupo de ECs e links no formato de `ObjectStateHolderVector`.

A classe `ObjectStateHolder` é uma maneira de representar um objeto ou link definido no CMDB. O objeto `ObjectStateHolder` contém o nome do TEC e uma lista de atributos e seus valores. O `ObjectStateHolderVector` é um vetor de instâncias `ObjectStateHolder`.

A sintaxe de ObjectStateHolder

Esta seção explica como criar os resultados do DFM em um modelo do UCMDB.

Exemplo de configuração de atributos nos ECs:

A classe `ObjectStateHolder` descreve o gráfico de resultados do DFM. Cada EC e link (relacionamento) é colocado em uma instância da classe `ObjectStateHolder` como no exemplo de código Jython a seguir:

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

- A Linha 1 cria um EC de tipo **siebelappserver**.
- A Linha 2 cria um atributo chamado **data_name** com um valor de **sblsvrName** que é um conjunto de variáveis Jython com o valor descoberto para o nome do servidor.
- A Linha 3 define um atributo não-chave que é atualizado no CMDB.
- A Linha 4 é a criação de containment (o resultado é um gráfico). Ela especifica que esse servidor de aplicativo está condito em um host (outra classe `ObjectStateHolder` no escopo).

Observação: Cada EC sendo relatado pelo script Jython precisa incluir valores para todos os atributos-chave do Tipo de EC do EC.

Exemplo de relacionamentos (links):

O exemplo de link a seguir explica como o gráfico é representado:

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

- A Linha 1 cria o link (que também é a classe `ObjectStateHolder`. A única diferença é que `route` é um Tipo de EC de link).
- As Linhas 2 e 3 especificam os nós na extremidade de cada link. Para fazer isso, use os

atributos **end1** e **end2** do link que precisa ser especificado (porque são os atributos-chave mínimos de cada link). Os valores de atributo são as instâncias de `ObjectStateHolder`. Para ver detalhes sobre End 1 e End 2, consulte "[Link](#)" na página 1 no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Cuidado: Um link é direcional. Você deve verificar se os nós End 1 e End 2 correspondem a TECs válidos em cada extremidade. Se os nós não forem válidos, o objeto de resultado falhará na validação e não será relatado corretamente. Para ver detalhes, consulte "[Relacionamentos de tipos de EC](#)" na página 1, no *Guia de Modelagem do HP Universal CMDB*.

Exemplo de vetor (reunindo ECs):

Depois de criar objetos com atributos, além de links com objetos em suas extremidades, você precisará agrupá-los. Para fazer isso, adicione-os a uma instância `ObjectStateHolderVector` da seguinte maneira:

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

Para ver detalhes sobre como relatar esse resultado composto para o Framework para que ele possa ser enviado ao servidor CMDB, consulte o método [sendObjects](#).

Depois que o gráfico de resultados for montado em uma instância `ObjectStateHolderVector`, será necessário retorná-lo ao DFM Framework para que seja inserido ao CMDB. Para fazer isso, retorne a instância `ObjectStateHolderVector` como sendo o resultado da função `DiscoveryMain()`.

Observação: para ver detalhes sobre como criar **OSH** para TECs comuns, consulte "[modeling.py](#)" na página 75 em "[Bibliotecas e utilitários Jython](#)" na página 73.

Instância do Framework

A instância do Framework é o único argumento fornecido na função principal no script Jython. Trata-se de uma interface que pode ser usada para recuperar as informações necessárias para executar o script (por exemplo, informações sobre ECs Acionadores e parâmetros do adaptador) e que também serve para relatar erros que ocorrem durante a execução do script. Para obter detalhes, consulte "[Referência da API de Gerenciamento de Fluxo de Dados da HP](#)" na página 38.

O uso correto da instância do Framework é transmiti-la como argumento para cada método que a usa.

Exemplo:

```
def DiscoveryMain(Framework):
    OSHVResult = helperMethod (Framework)
    return OSHVResult
def helperMethod (Framework):
    ....
    probe_name = Framework.getDestinationAttribute('probe_
```

```
name')
    ...
    return result
```

Esta seção descreve os principais usos do Framework:

- "Framework.getTriggerCIData(String attributeName)" abaixo
- "Framework.createClient(credentialsId, props)" abaixo
- "Framework.getParameter (String parameterName)" na página seguinte
- "Framework.reportError(String message) e Framework.reportWarning(String message)" na página 47

Framework.getTriggerCIData(String attributeName)

Essa API proporciona a etapa intermediária entre os dados de EC Acionador definidos no adaptador e no script.

Exemplo de recuperação de informações de credenciais:

Solicite as seguintes informações de dados de EC Acionador:

Dados de EC Acionado	
Nome	Valor
filename	\${SOURCE.credential_id}
hostKey	\${SOURCE.host_key}

Para recuperar as informações de credenciais da tarefa, use esta API:

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

Framework.createClient(credentialsId, props)

Faça uma conexão com uma máquina remota criando um objeto cliente e executando comandos nesse cliente. Para criar um cliente, recupere a classe `ClientFactory`. O método `getClientFactory()` recebe o tipo do protocolo de cliente solicitado. As constantes do protocolo são definidas na classe `ClientsConsts`. Para ver detalhes sobre credenciais e protocolos aceitos, consulte *Guia de Conteúdo de Descoberta e Integração do HP Universal CMDB*.

Exemplo de criação de uma instância de cliente para a ID de Credenciais:

Para criar uma instância `Client` para a ID de credenciais:

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

Agora você pode usar a instância `Client` para se conectar à máquina ou aplicativo relevante.

Exemplo de criação de um cliente WMI e execução de uma consulta WMI:

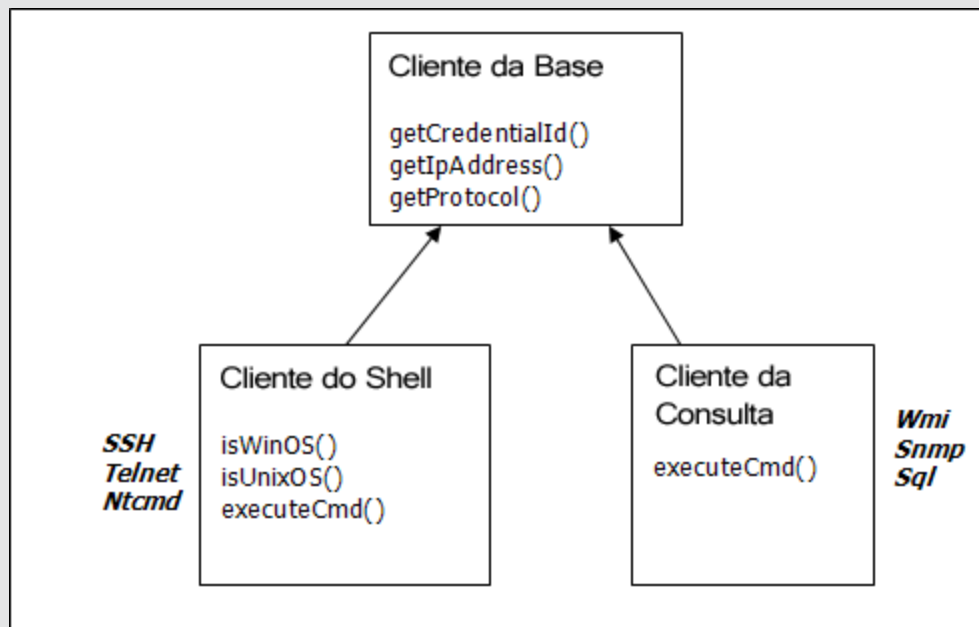
Para criar um cliente WMI e executar uma consulta WMI usando o cliente:

```
wmiClient = Framework.createClient(credential)
resultSet = wmiClient.executeQuery("SELECT TotalPhysicalMemory
FROM Win32_LogicalMemoryConfiguration")
```

Observação: para fazer com que a API `createClient()` funcione, adicione o seguinte parâmetro aos parâmetros de dados de EC Acionador: **credentialsId = `#{SOURCE.credentials_id}`** in the Triggered CI Data pane. Ou adicione manualmente a ID de credenciais ao chamar a função:

```
wmiClient = clientFactory().createClient(credentials_id).
```

O diagrama a seguir ilustra a hierarquia dos clientes, com suas APIs comumente aceitas:



Para ver detalhes sobre os clientes e suas APIs aceitas, consulte [BaseClient](#), [ShellClient](#) e [QueryClient](#) no *HP Discovery and Dependency Mapping Schema Reference*. Esses arquivos estão localizados na seguinte pasta:

```
<Diretório Raiz do UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\leng\APIs\DDM_Schema\webframe.html
```

Framework.getParameter (String parameterName)

Além de recuperar informações sobre o EC Acionador, geralmente você precisa recuperar um valor de parâmetro do adaptador. Por exemplo:

Parâmetros		
Substituir	Nome	Valor
<input checked="" type="checkbox"/>	protocolType	MicrosoftSQL Server

Exemplo de recuperação do valor do parâmetro protocolType:

Para recuperar o valor do parâmetro `protocolType` do script Jython, use a seguinte API:

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

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

Alguns erros (por exemplo, falha na conexão, problemas de hardware, tempos limites) podem ocorrer durante uma execução de script. Quando esse tipo de erro é detectado, o Framework pode relatar o problema. A mensagem relatada chega ao servidor e é exibida para o usuário.

Exemplo de um erro de relatório e mensagem:

O diagrama a seguir ilustra o uso da API `reportError(<Mensagem_de_Erro>)`:

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

Você pode usar uma das APIs – `Framework.reportError(String message)`, `Framework.reportWarning(String message)` – para relatar um problema. A diferença entre as duas APIs é que, ao relatar um erro, a Sonda salva um arquivo de log de comunicação com os parâmetros da sessão inteira no sistema de arquivos. Dessa maneira, é possível controlar a sessão e entender melhor o erro.

Para ver detalhes sobre mensagens de erro, consulte "[Mensagem de erro](#)" na página 77.

Localizando as credenciais corretas (para adaptadores de conexão)

Um adaptador que tenta se conectar a um sistema remoto precisa testar todas as credenciais possíveis. Um dos parâmetros necessários ao criar um cliente (através do `ClientFactory`) é a ID de credenciais. O script de conexão obtém acesso a possíveis conjuntos de credenciais e testa-os um a um com o método `clientFactory.getAvailableProtocols()`. Quando um conjunto de credenciais obtém êxito, o adaptador relata um objeto de conexão de EC no host desse EC acionador (com a ID de credenciais que corresponde ao IP) ao CMDB. Os adaptadores subsequentes podem usar esse EC de objeto de conexão diretamente para se conectar ao

conjunto de credenciais (ou seja, os adaptadores não precisam testar todas as credenciais possíveis outra vez).

O exemplo a seguir mostra como obter todas as entradas do protocolo SNMP. Observe que aqui o IP é obtido dos dados de EC Acionador (# Get the Trigger CI data values).

O script de conexão solicita todas as credenciais de protocolo possíveis (# Go over all the protocol credentials) e testa-as em loop até que uma obtenha êxito (resultVector). Para ver detalhes, consulte a entrada **two-phase connect paradigm** em "[Separando adaptadores](#)" na página 24.

```
import logger
from appilog.collectors.clients import ClientsConsts
from appilog.common.system.types.vectors import
ObjectStateHolderVector
    def mainFunction(Framework):
resultVector = ObjectStateHolderVector()
    # Get the Trigger CI data values
    ip_address = Framework.getDestinationAttribute('ip_address')
    ip_domain = Framework.getDestinationAttribute('ip_domain')
    # Create the client factory for SNMP
    clientFactory = framework.getClientFactory(ClientsConsts.SNMP_
PROTOCOL_NAME)
    protocols = clientFactory.getAvailableProtocols(ip_address,
ip_domain)

    connected = 0
    # Go over all the protocol credentials
    for credentials_id in protocols:
        client = None
        try:
            # try to connect to the snmp agent
            client = clientFactory.createClient(credentials_id)
            // Query the agent
            ....
            # connection succeed
            connected = 1
        except:
            if client != None:
                client.close()
    if (not connected):
        logger.debug('Failed to connect using all credentials')
    else:
        // return the results as OSHV
        return resultVector
```

Manipulando exceções de Java

Algumas classes Java lançam uma exceção após falha. É recomendável executar o comando catch na exceção e manipulá-la, caso contrário ela fará com que o adaptador seja encerrado inesperadamente.

Ao executar o comando `catch` em uma exceção conhecida, na maioria dos casos você deve imprimir o rastreamento de pilha no log e emitir uma mensagem adequada à UI, por exemplo:

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

Se a exceção não for fatal e o script puder prosseguir, você deverá omitir a chamada para o método `reportError()` e permitir que o script prossiga.

Oferecer suporte a localização em adaptadores Jython

O recurso de localidade multilíngue permite que o DFM funcione em diferentes idiomas de sistemas operacionais e possibilita personalizações apropriadas em tempo de execução.

Anteriormente ao Content Pack 3.00, o DFM usava codificação especificada estatisticamente para tratar a saída de todos os destinos da rede. Entretanto, essa abordagem não atende a uma rede de TI multilíngue: para descobrir hosts com diferentes idiomas de sistema operacional, os administradores da Sonda sempre precisavam repetir, manualmente e várias vezes, a execução dos trabalhos do DFM com diferentes parâmetros de trabalho. Esse procedimento gerava um grave impacto na carga da rede e, pior ainda, bloqueava vários recursos essenciais do DFM, como invocação imediata do trabalho em um EC acionador ou atualização automática de dados no UCMDB pelo Gerenciador de Programação.

Os seguintes idiomas de localidade são aceitos por padrão: japonês, russo e alemão. A localidade padrão é inglês.

Esta seção inclui:

- ["Adicionar suporte para um novo idioma" abaixo](#)
- ["Alterar o idioma padrão" na página 51](#)
- ["Determinar o conjunto de caracteres para codificação" na página 51](#)
- ["Definir um novo trabalho para operar com dados localizados" na página 51](#)
- ["Decodificar comandos sem uma palavra-chave" na página 52](#)
- ["Trabalhar com bundles de recursos" na página 53](#)
- ["Referência de API" na página 54](#)

Adicionar suporte para um novo idioma

Esta tarefa descreve como adicionar suporte a um novo idioma.

Esta tarefa inclui as seguintes etapas:

- "Adicionar um bundle de recursos (arquivos *.properties)" abaixo
- "Declarar e registrar o objeto de idioma" abaixo

1. **Adicionar um bundle de recursos (arquivos *.properties)**

Adicione um bundle de recursos de acordo com o trabalho que será executado. A tabela a seguir lista os trabalhos do DFM e o bundle de recursos usado por cada trabalho:

Trabalho	Nome base do bundle de recursos
Monitor de Arquivos por Shell	langFileMonitoring
Recursos e Aplicativos de Host por Shell	langHost_Resources_By_TTY, langTCP
Hosts por Shell Usando NSLOOKUP no Servidor DNS	langNetwork
Conexão de Host por Shell	langNetwork
Coletar Dados de Rede por Shell ou SNMP	langTCP
Recursos e Aplicativos de Host por SNMP	langTCP
Conexão do Microsoft Exchange por NTCMD, Topologia do Microsoft Exchange por NTCMD	msExchange
MS Cluster por NTCMD	langMsCluster

Para obter detalhes sobre bundles, consulte "Trabalhar com bundles de recursos" na página 53.

2. **Declarar e registrar o objeto de idioma**

Para definir um novo idioma, adicione as duas linhas de código a seguir ao script **shellutils.py**, que no momento contém a lista de todos os idiomas aceitos. O script é incluído no pacote **AutoDiscoveryContent**. Para visualizar o script, acesse a janela Gerenciamento do Adaptador. Para obter detalhes, consulte "Janela Gerenciamento do Adaptador" na página 1 no Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB.

- a. Declare o idioma da seguinte maneira:

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

Para ver detalhes sobre idioma de classe, consulte "Referência de API" na página 54. Para ver detalhes sobre o objeto Localidade de Classe, consulte <http://java.sun.com/j2se/1.5.0/docs/api/java/util/Locale.html>. Você pode usar uma localidade existente ou definir uma nova.

- b. Registre o idioma adicionando-o à seguinte coleção:

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

Alterar o idioma padrão

Se não for possível determinar o idioma do sistema operacional, o padrão será usado. O idioma padrão é especificado no arquivo **shellutils.py**.

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

Para alterar o idioma padrão, inicialize a variável `DEFAULT_LANGUAGE` com um idioma diferente. Para obter detalhes, consulte "Adicionar suporte para um novo idioma" na página 49.

Determinar o conjunto de caracteres para codificação

O conjunto de caracteres adequado para decodificar a saída de comando é determinado em tempo de execução. A solução multilíngue se baseia nos seguintes fatos e pressuposições:

1. É possível determinar o idioma do sistema operacional de maneira independente da localidade, por exemplo, executando o comando **chcp** no Windows ou o comando **locale** no Linux.
2. O relacionamento entre idioma e codificação é bem conhecido e pode ser definido estaticamente. Por exemplo, o idioma russo tem duas das codificações mais conhecidas: `Cp866` e `Windows-1251`.
3. Um conjunto de caracteres para cada idioma é preferencial, por exemplo, o conjunto de caracteres preferencial para o idioma russo é `Cp866`. Isso significa que a maioria dos comandos produz saída nessa codificação.
4. A codificação em que é fornecida a saída de comando seguinte é imprevisível, mas é uma das codificações possíveis em um determinado idioma. Por exemplo, ao trabalhar com uma máquina com Windows e localidade russo, o sistema fornece a saída de comando **ver** em `Cp866`, mas o comando **ipconfig** é fornecido em `Windows-1251`.
5. Um comando conhecido produz palavras-chave conhecidas em sua saída. Por exemplo, o comando **ipconfig** contém a forma traduzida da cadeia de caracteres **IP-Address**. Assim, a saída de comando **ipconfig** contém **IP-Address** para o sistema operacional em inglês, **IP-Адрес** para o sistema operacional em russo, **IP-Adresse** para o sistema operacional em alemão e assim por diante.

Depois de descoberto o idioma em que a saída de comando é produzida (# 1), os conjuntos de caracteres possíveis ficam limitados a um ou dois (# 2). Além disso, são conhecidas quais palavras-chave estão contidas nessa saída (# 5).

A solução, portanto, é decodificar a saída de comando com uma das codificações possíveis pesquisando uma palavra-chave no resultado. Se a palavra-chave for encontrada, o conjunto de caracteres atual será considerado o correto.

Definir um novo trabalho para operar com dados localizados

Esta tarefa descreve como escrever um novo trabalho que pode operar com dados localizados.

Os scripts Jython normalmente executam comandos e analisam sua saída. Para receber essa saída de comando com decodificação adequada, use a API para a classe **ShellUtils**. Para obter detalhes, consulte "[Visão geral da API de serviço Web do RTSM \(HP Universal CMDB\)](#)" na página 1.

Esse código normalmente assume a seguinte forma:

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

1. Crie um cliente:

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

2. Crie uma instância da classe **ShellUtils** e adicione o idioma do sistema operacional a ela. Se o idioma não for adicionado, o idioma padrão será usado (normalmente inglês):

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

Durante a inicialização do objeto, o DFM detecta automaticamente o idioma da máquina e define a codificação preferencial com base no objeto `Language` predefinido. A codificação preferencial é a primeira instância que aparece na lista de codificação.

3. Recupere o bundle de recursos apropriado do **shellclient** usando o método **getLanguageBundle**:

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

4. Recupere uma palavra-chave do bundle de recursos, adequado a um comando específico:

```
strWindowsIPAddress = languageBundle.getString('windows_ipconfig_
str_ip_address')
```

5. Invoque o método **executeCommandAndDecode** e repasse a palavra-chave a ele no objeto **ShellUtils**:

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

O **ShellUtils** `object` também é necessário para vincular um usuário à referência da API (em que esse método é descrito em detalhes).

6. Analise a saída normalmente.

Decodificar comandos sem uma palavra-chave

A abordagem atual para localização usa uma palavra-chave para decodificar toda a saída de comando. Para obter detalhes, consulte a etapa "[Recupere uma palavra-chave do bundle de](#)

recursos, adequado a um comando específico:" em "Definir um novo trabalho para operar com dados localizados" na página 51.

Entretanto, outra abordagem usa uma palavra-chave para decodificar somente a primeira saída de comando e decodifica outros comandos com o conjunto de caracteres usado para decodificar o primeiro comando. Para fazer isso, use os métodos **getCharsetName** e **useCharset** do objeto **ShellUtils**.

O caso de uso regular funciona da seguinte maneira:

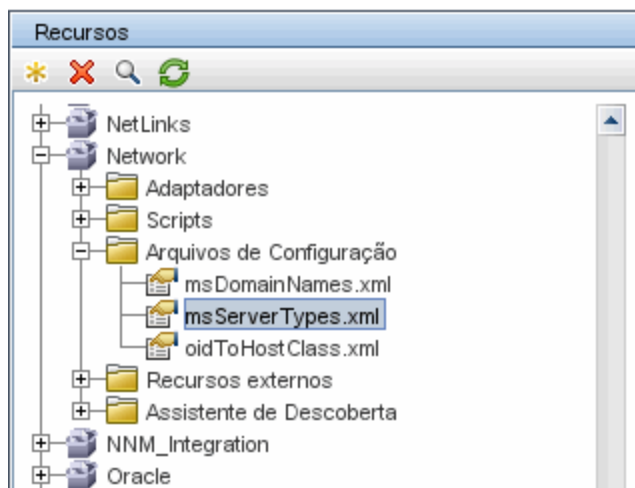
1. Invoque o método **executeCommandAndDecode** uma vez.
2. Obtenha o nome do conjunto de caracteres de uso mais recente através do método **getCharsetName**.
3. Faça com que **shellUtils** use esse conjunto de caracteres por padrão invocando o método **useCharset** no objeto **ShellUtils**.
4. Invoque o método **execCmd** de **ShellUtils** uma ou mais vezes. A saída é retornada com o conjunto de caracteres especificado na etapa anterior. Não ocorre nenhuma operação de decodificação adicional.

Trabalhar com bundles de recursos

Um bundle de recursos é um arquivo que tem uma extensão **properties** (***.properties**). Um arquivo **properties** pode ser considerado um dicionário que armazena dados no formato de **key = value**. Cada linha em um arquivo **properties** contém uma associação **key = value**. A principal funcionalidade de um bundle de recursos é retornar um valor pela sua chave.

Os bundles de recursos estão localizados na máquina da Sonda:

C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles. Eles são baixados do Servidor UCMDB como qualquer outro arquivo de configuração. Podem ser editados, adicionados ou removidos, na janela Recursos. Para obter detalhes, consulte "[Painel Arquivo de Configuração](#)" na página 1 no Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB.



Ao descobrir um destino, o DFM normalmente precisa analisar o texto da saída de comando ou do conteúdo de arquivo. Essa análise geralmente se baseia em uma expressão regular. Diferentes idiomas exigem diferentes expressões regulares para ser usadas para análise. Para que o código seja escrito somente uma vez para todos os idiomas, todos os dados específicos de idioma

precisam ser extraídos para os bundles de recursos. Há um bundle de recursos para cada idioma. (Apesar de ser possível que um bundle de recursos contenha dados para diferentes idiomas, no DFM um bundle de recursos sempre contém dados para um idioma.)

O script Jython em si não inclui dados específicos de idioma embutidos no código-fonte (por exemplo, expressões regulares específicas de idioma). O script determina o idioma do sistema remoto, carrega o bundle de recursos adequado e obtém todos os dados específicos de idioma por uma chave específica.

No DFM, os bundles de recursos tem um formato de nome específico: `<nome_base>_<identificador_de_idioma>.properties`, por exemplo, `langNetwork_spa.properties`. (O bundle de recursos padrão tem o seguinte formato: `<nome_base>.properties`, por exemplo, `langNetwork.properties`.)

O formato `nome_base` reflete a finalidade desse bundle. Por exemplo, **langMsCluster** significa que o bundle de recursos contém recursos específicos de idioma usados pelos trabalhos do MS Cluster.

O formato `identificador_de_idioma` é um acrônimo de três letras usado para identificar o idioma. Por exemplo, `rus` indica o idioma russo e `ger` indica o idioma alemão. Esse identificador de idioma é incluído na declaração do objeto `Language`.

Referência de API

Esta seção inclui:

- "A classe do idioma" abaixo
- "O método `executeCommandAndDecode`" na página seguinte
- "O método `getCharsetName`" na página seguinte
- "O método `useCharset`" na página seguinte
- "O método `getLanguageBundle`" na página 56
- "O campo `osLanguage`" na página 56

A classe do idioma

Essa classe encapsula informações sobre o idioma, como sufixo do bundle de recursos, codificação possível e assim por diante.

Campos

Nome	Descrição
<code>locale</code>	Objeto Java que representa uma localidade.
<code>bundlePostfix</code>	Sufixo do bundle de recursos. Este sufixo é usado nos nomes de arquivos de bundles de recursos para identificar o idioma. Por exemplo, o bundle langNetwork_ger.properties inclui um sufixo de bundle ger .
<code>charsets</code>	Os conjuntos de caracteres usados para codificar esse idioma. Cada idioma

Nome	Descrição
	pode ter vários conjuntos de caracteres. Por exemplo, o idioma russo normalmente tem as codificações <code>Cp866</code> e <code>Windows-1251</code> .
<code>wmiCodes</code>	A lista de códigos WMI usados pelo sistema operacional Microsoft Windows para identificar o idioma. Todos os códigos possíveis estão listados em http://msdn.microsoft.com/en-us/library/aa394239(VS.85).aspx (na seção <code>OSLanguage</code>). Um dos métodos para identificar o idioma do sistema operacional é consultar o sistema operacional de classe WMI para a propriedade <code>OSLanguage</code> .
<code>codepage</code>	Página de código usada com um idioma específico. Por exemplo, <code>866</code> é usado para máquinas em russo e <code>437</code> para máquinas em inglês. Um dos métodos para identificar o idioma do sistema operacional é recuperar a página de código padrão (por exemplo, pelo comando <code>chcp</code>).

O método `executeCommandAndDecode`

Esse método precisa ser usado pelos scripts Jython de lógica de negócios. Ele encapsula a operação de decodificação e retorna uma saída de comando decodificada.

Argumentos

Nome	Descrição
<code>cmd</code>	O comando real que será executado.
<code>keyword</code>	A palavra-chave que será usada para a operação de decodificação.
<code>framework</code>	O objeto Framework repassado para cada script Jython executável no DFM.
<code>timeout</code>	O tempo limite do comando.
<code>waitForTimeout</code>	Especifica se o cliente deve aguardar quando o tempo limite for excedido.
<code>useSudo</code>	Especifica se <code>sudo</code> deve ser usado (relevante somente para clientes de máquina com UNIX).
<code>language</code>	Permite especificar o idioma diretamente em vez de detectar automaticamente um idioma.

O método `getCharsetName`

Este método retorna o nome do conjunto de caracteres de uso mais recente.

O método `useCharset`

Este método define o conjunto de caracteres na instância `ShellUtils`, que usa esse conjunto de caracteres para decodificação de dados inicial.

Argumentos

Nome	Descrição
charsetName	O nome do conjunto de caracteres, por exemplo, tipo de EC, por exemplo, windows-1251 or UTF-8.

Consulte também "O método `getCharsetName`" na página anterior.

O método `getLanguageBundle`

Este método deve ser usado para obter o bundle de recursos correto. Isso substitui a seguinte API:

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

Argumentos

Nome	Descrição
baseName	O nome do bundle sem o sufixo de idioma, por exemplo, langNetwork.
language	O objeto do idioma. <code>ShellUtils.osLanguage</code> deve ser repassado aqui.
framework	O Framework, objeto comum que é repassado para cada script Jython executável no DFM.

O campo `osLanguage`

Este campo contém um objeto que representa o idioma.

Trabalhar com o Analisador de Descoberta

A ferramenta Analisador de Descoberta serve para fins de depuração ao desenvolver pacotes, scripts ou qualquer outro tipo de conteúdo. A ferramenta executa um trabalho com base em um destino remoto e retorna logs que contêm informações, avisos e detalhes de erros, além de resultados de ECs descobertos.

Observe que os resultados nem sempre são relatados à UI. Isso ocorre porque os resultados são relatados de duas maneiras e só há suporte a uma delas. Além disso, não há suporte ao log de comunicação do Eclipse.

Ao executar a ferramenta do Eclipse, o arquivo **DataFlowProbe.properties** (**C:\hp\UCMDB\DataFlowProbe\conf\DataFlowProbe.properties**) deve conter o seguinte parâmetro definido como **verdadeiro**:

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

Para obter detalhes, consulte "Executar o Analisador de Descoberta a partir do Eclipse" na página 63.

Em todos os outros casos (quando a ferramenta é executada do arquivo **cmd** ou enquanto a Sonda está em execução), esse sinalizador precisa ser definido como **false**:


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

Tarefas e registros

Um arquivo de tarefas contém dados relativos a uma tarefa que será executada. A tarefa consiste em informações como o nome do trabalho e os parâmetros obrigatórios que definem o EC acionador, por exemplo, o endereço de destino remoto.

Um arquivo de registro contém informações de tarefas, além dos resultados de uma execução específica, ou seja, a comunicação detalhada (incluindo uma resposta) entre a Sonda ou o Analisador de Descoberta (qualquer módulo que tenha executado a tarefa) e o destino remoto.

Uma tarefa que é definida por um arquivo de tarefa pode ser executada com base em um destino remoto, enquanto uma tarefa que é definida por um arquivo de registro (que contém dados adicionais sobre uma execução específica) pode ser executada e também reproduzida (ou seja, pode reproduzir a mesma execução documentada no arquivo de registro).

Logs

Os logs fornecem informações sobre a execução mais recente, da seguinte maneira:

- **Log geral.** Este log inclui todos os dados de informações, erros e avisos que ocorreram durante a execução.
- **Log de comunicação.** Este log contém a comunicação detalhada entre o Analisador de Descoberta e o destino remoto (incluindo sua resposta). Após a execução, o log pode ser salvo como um arquivo de registro.
- **Log de resultados.** Exibe uma lista de ECs descobertos. O tempo de aparição de cada EC depende do design dos adaptadores e scripts.

É possível salvar todos os logs juntos ou cada um separadamente. Quando você salva todos os logs, eles são salvos juntos sob um único nome.

Se você executar novamente um arquivo de registro, os mesmos dados serão exibidos no log de comunicação, sendo que a única diferença será o horário da execução.

Limitação: Os logs de Comunicação e Resultados não estão disponíveis ao executar o Analisador de Descoberta através do Eclipse.

Esta seção inclui as seguintes etapas:

- ["Pré-requisitos" na página seguinte](#)
- ["Acessar o Analisador de Descoberta" na página seguinte](#)
- ["Definir uma tarefa" na página 59](#)
- ["Definir uma nova tarefa" na página 59](#)
- ["Recuperar um registro" na página 60](#)
- ["Abrir um arquivo de tarefa" na página 61](#)

- "Importar uma tarefa do banco de dados" na página 61
- "Editar uma tarefa" na página 61
- "Salvar a tarefa e os logs" na página 61
- "Executar a tarefa" na página 61
- "Enviar um resultado de tarefa ao servidor" na página 62
- "Configurações de importação" na página 62
- "Pontos de interrupção" na página 62
- "Configurar o Eclipse" na página 62

1. Pré-requisitos

- A Sonda precisa estar instalada. (O Analisador de Descoberta é instalado como parte do processo de instalação da Sonda e compartilha recursos com ela.)
- A Sonda não precisa estar em execução enquanto você trabalha com o Analisador de Descoberta.

Entretanto, se a Sonda já tiver sido executada com base em um Servidor UCMDB, todos os recursos necessários já estarão baixados para o sistema de arquivos. Se a Sonda ainda não tiver sido executada, você poderá carregar os recursos necessários pelo Analisador de Descoberta através do menu Configurações. Para obter detalhes, consulte "[Configurações de importação](#)" na página 62.

- O Servidor CMDB não precisa estar instalado.

2. Acessar o Analisador de Descoberta

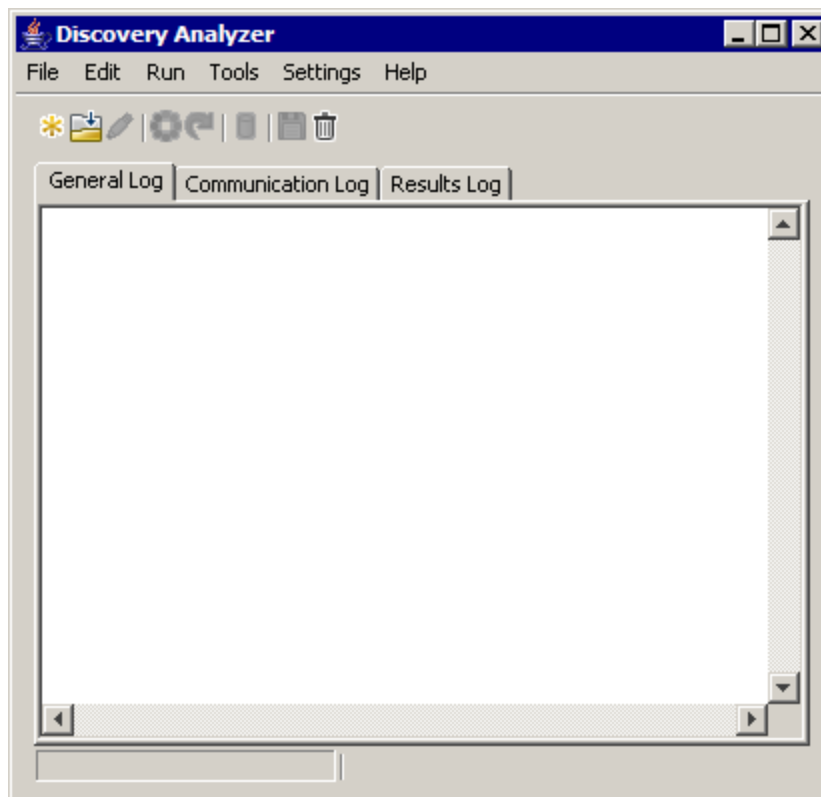
Acesse o Analisador de Descoberta de uma das seguintes maneiras:

- Ao trabalhar com o Eclipse.

A instalação da Sonda vem com um espaço de trabalho do Eclipse padrão localizado em **C:\hp\UCMDB\DataFlowProbeltools\discoveryAnalyzerWorkspace**. Esse espaço de trabalho inclui um script Jython para iniciar o Analisador de Descoberta (**startDiscoveryAnalyzerScript.py**) além de um link para todos os scripts do DFM. Se você iniciar a ferramenta dessa maneira, poderá localizar os pontos de interrupção nos scripts Jython para fins de depuração.

- Diretamente, clicando duas vezes no arquivo na seguinte pasta: **C:\hp\UCMDB\DataFlowProbeltools\discoveryAnalyzer.cmd**. Para ver detalhes, consulte a próxima seção.

A janela Analisador de Descoberta será aberta:




3. Definir uma tarefa

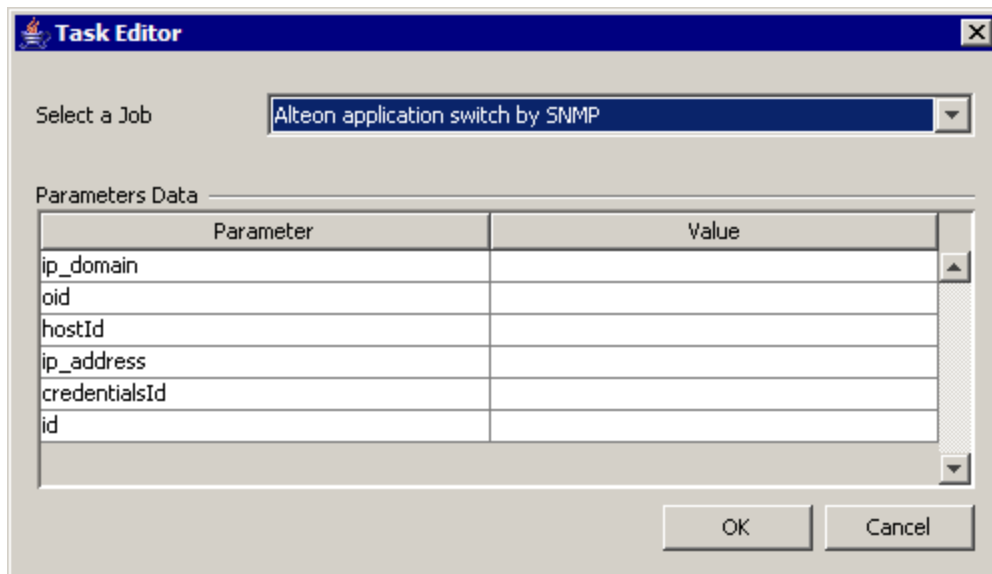
Defina uma tarefa usando um dos métodos a seguir:

- Definindo uma nova tarefa. Para obter detalhes, consulte ["Definir uma nova tarefa"](#) abaixo.
- Importando uma tarefa de um arquivo de registro. Para obter detalhes, consulte ["Recuperar um registro"](#) na página seguinte.
- Importando uma tarefa salva de um arquivo de tarefas. Para obter detalhes, consulte ["Abrir um arquivo de tarefa"](#) na página 61.
- Recuperando um trabalho do banco de dados interno da Sonda. Para obter detalhes, consulte ["Importar uma tarefa do banco de dados"](#) na página 61.

4. Definir uma nova tarefa

- a. Exiba o Editor de Tarefas: clique no botão **Nova Tarefa** .

O Editor de Tarefas exibe uma lista de trabalhos que existem no momento no sistema de arquivos. Essa lista é atualizada sempre que a Sonda recebe tarefas do servidor ou os pacotes são implantados manualmente do menu Configurações.



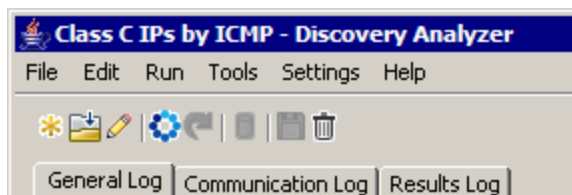
- b. Select a job.
- c. Insira valores para todos os parâmetros.

Os parâmetros exibidos aqui são os parâmetros do adaptador do DFM. Eles podem ser visualizados no painel Parâmetros de Padrão de Descoberta na guia Assinatura de Padrão. Para obter detalhes, consulte ["Guia Definição do Adaptador" na página 1 no Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB](#).

Todos os campos são obrigatórios (a menos que o script de um trabalho exija que o campo seja vazio).

Para parâmetros que exigem uma ID ou valor de entrada de ID de credenciais, você pode usar IDs criados aleatoriamente: clique com o botão direito do mouse na caixa de valor e selecione **Gerar ID do CMDB aleatória** ou **Selecionador de Credencial**.

A tarefa agora está ativa e o nome da tarefa aberta é exibido na barra de título:



- d. Continue com o procedimento para definir uma tarefa. Para obter detalhes, consulte ["Salvar a tarefa e os logs" na página seguinte](#).

5. Recuperar um registro

Você pode definir uma tarefa abrindo um arquivo de registro que contém dados relativos a uma execução específica. Se uma tarefa for definida dessa maneira, você poderá reproduzir a execução específica selecionando a opção de reprodução. (Se uma tarefa for executada novamente, as respostas serão recebidas dos dados armazenados no arquivo de registro e não do destino remoto.)

Selecione **Arquivo > Abrir Registro**. Navegue até a pasta onde você salvou o registro. O registro agora está ativo e o nome da tarefa é exibido na barra de título.

Para ver detalhes sobre como adquirir um arquivo de registro, consulte "[Registrar código do DFM](#)" na página 72.

6. Abrir um arquivo de tarefa

Você pode definir uma tarefa de um arquivo de registro: Selecione **Arquivo > Abrir Tarefa**.

7. Importar uma tarefa do banco de dados

Você poderá recuperar uma tarefa do banco de dados da Sonda desde que a Sonda já tenha sido executada e tenha tarefas ativas em seu banco de dados interno. Você pode usar os valores de parâmetro para definir a tarefa.

- a. Selecione **Arquivo > Importar Tarefa do Banco de Dados da Sonda**.
- b. Na caixa de diálogo que for aberta, selecione a tarefa que será executada e clique em **OK**.
- c. Continue com o procedimento para definir uma tarefa. Para obter detalhes, consulte "[Salvar a tarefa e os logs](#)" abaixo.

8. Editar uma tarefa

Depois que uma tarefa for definida, o nome da tarefa (ou do arquivo) será exibido na barra de título. Agora o arquivo poderá ser editado.

- a. Selecione **Editar > Editar Tarefa**.
- b. Faça quaisquer alterações na tarefa e clique em **OK**.

9. Salvar a tarefa e os logs

Você pode salvar os parâmetros da tarefa: Selecione **Arquivo > Salvar Tarefa**.

As seguintes opções estão disponíveis somente depois que uma tarefa é executada:

- Salve um registro da tarefa. Você pode salvar os parâmetros da tarefa e os resultados da execução da tarefa: Selecione **Arquivo > Salvar Registro**.
- Salve um log da tarefa: selecione **Arquivo > Salvar Log Geral**.
- Salve os resultados: selecione **Arquivo > Salvar Resultados**.

10. Executar a tarefa

A próxima etapa no procedimento é executar a tarefa criada.

- a. Importe o arquivo de configuração de credenciais/intervalos. Para obter detalhes, consulte "[Configurações de importação](#)" na página seguinte.
- b. Para executar a tarefa somente com base em um destino remoto, clique no botão **Executar Tarefa**.

O Analisador de Descoberta executa o trabalho e exibe informações nos três arquivos de log: **Geral**, **Comunicação** e **Resultados**.

- c. É possível salvar os arquivos de log, sejam juntos ou separadamente. selecione **Arquivo > Salvar Log Geral**, **Salvar Registro**, **Salvar Resultados** ou **Salvar Todos os Logs**. Para ver detalhes sobre os arquivos de log, consulte "[Logs](#)" na página 57.

- d. Se uma tarefa for recuperada de um arquivo de registro, a execução que está documentada nesse arquivo poderá ser reproduzida clicando no botão **Reproduzir**. O mesmo log de Comunicação será exibido, mas o tempo de execução será atualizado.

11. Enviar um resultado de tarefa ao servidor

Se a execução de uma tarefa for encerrada com os resultados (ou seja, a guia Log de Resultados exibir uma lista de ECs descobertos), você poderá enviar os resultados ao Servidor UCMDB. Isso é útil se, por exemplo, anteriormente você estava testando um script quando o servidor estava inoperante.

Observação: Só é possível enviar resultados para um Servidor UCMDB que receba tarefas da Sonda instalada na mesma máquina do Analisador de Descoberta.

12. Configurações de importação

Para executar tarefas no arquivo de registro de reprodução, é necessário importar o arquivo **domainScopeDocument.bin**. Durante a importação, digite uma senha.

- a. Inicie um navegador da Web e insira a seguinte URL: **http://localhost:8080/jmx-console**. Você pode precisar fazer logon com um nome de usuário e senha.
- b. Clique em **UCMDB:service=DiscoveryManager** para abrir a página Visualização do JMX MBEAN.
- c. Localize a operação **exportCredentialsAndRangesInformation**. Siga este procedimento:
 - o Insira a ID do cliente (o padrão é **1**).
 - o Insira um nome para o arquivo exportado.
 - o Insira a senha.
 - o Defina **isEncrypted** como **False**.
- d. Clique em **Invoke** para exportar o arquivo **domainScopeDocument.bin**.

Quando o processo de exportação for concluído com êxito, o arquivo será salvo no seguinte local: **C:\hp\UCMDB\UCMDBServer\conf\discovery\<customer_dir>**.
- e. Copie o arquivo **domainScopeDocument.bin** para o sistema de arquivos da Sonda de Fluxo de Dados e importe-o selecionando: **Configurações > Importar domainScopeDocument**.

Observação: Durante a importação do arquivo **domainScopeDocument**, você será solicitado a fornecer uma senha. Essa solicitação também é exibida depois de cada reinício do Analisador de Descoberta e antes da execução da primeira tarefa ou registro.

13. Pontos de interrupção

Se você executar o Analisador de Descoberta do script Python, poderá adicionar pontos de interrupção ao script.

14. Configurar o Eclipse

Para ver detalhes sobre como executar scripts Jython em modo de depuração, consulte "Executar o Analisador de Descoberta a partir do Eclipse" abaixo.

Executar o Analisador de Descoberta a partir do Eclipse

Esta tarefa explica como configurar o Eclipse para permitir a execução de scripts Jython em modo de depuração, proporcionando assim maior visibilidade dos threads de trabalho, ECs acionadores e resultados.

Esta seção inclui as seguintes etapas:

- "Pré-requisitos" abaixo
- "Descompactar o Eclipse e iniciá-lo" abaixo
- "Configurar o espaço de trabalho padrão" abaixo
- "Configurar as Extensões PyDev" na página seguinte
- "Configurar o espaço de trabalho do Analisador de Descoberta" na página 65
- "Configurar o caminho de classe e o interpretador" na página 69
- "Executar o Analisador de Descoberta" na página 71

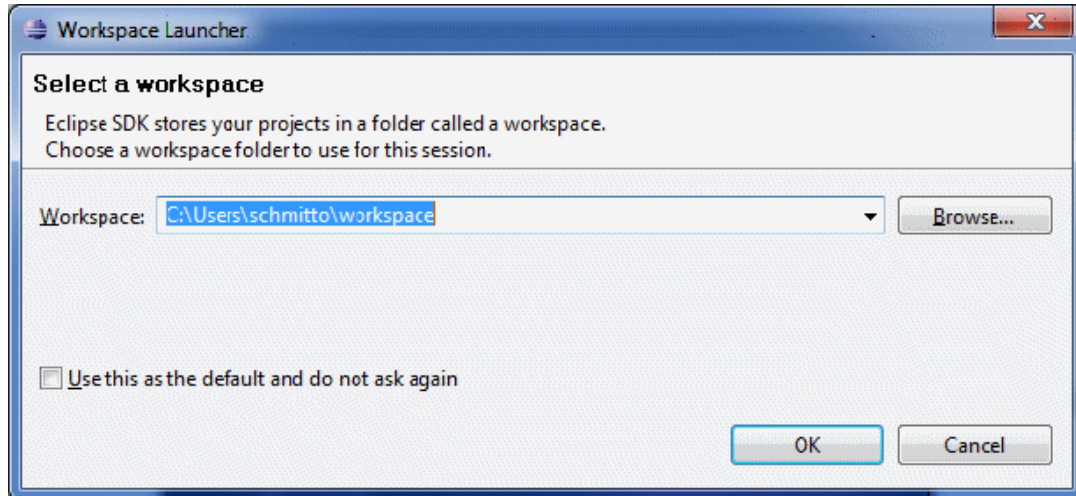
1. Pré-requisitos

- Instale a versão mais recente do Eclipse no computador. O aplicativo está disponível em www.eclipse.org.
- Verifique se a Sonda de Fluxo de Dados está instalada no mesmo computador.
- Verifique se o parâmetro `appilog.agent.local.discoveryAnalyzerFromEclipse` no arquivo `DataFlowProbe.properties` está definido como `verdadeiro`.

2. Descompactar o Eclipse e iniciá-lo

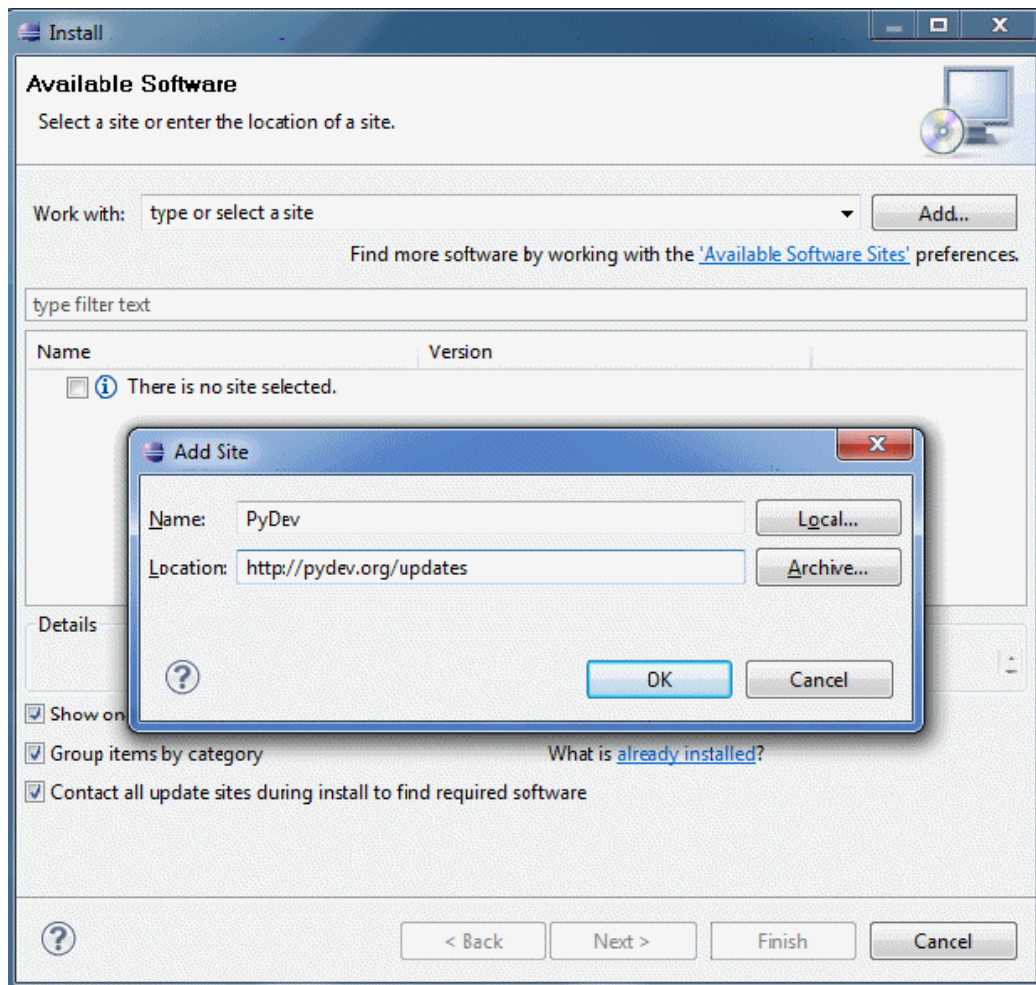
3. Configurar o espaço de trabalho padrão

Configure o espaço de trabalho padrão em que o Eclipse salva e armazena todos os projetos e dados relacionados.



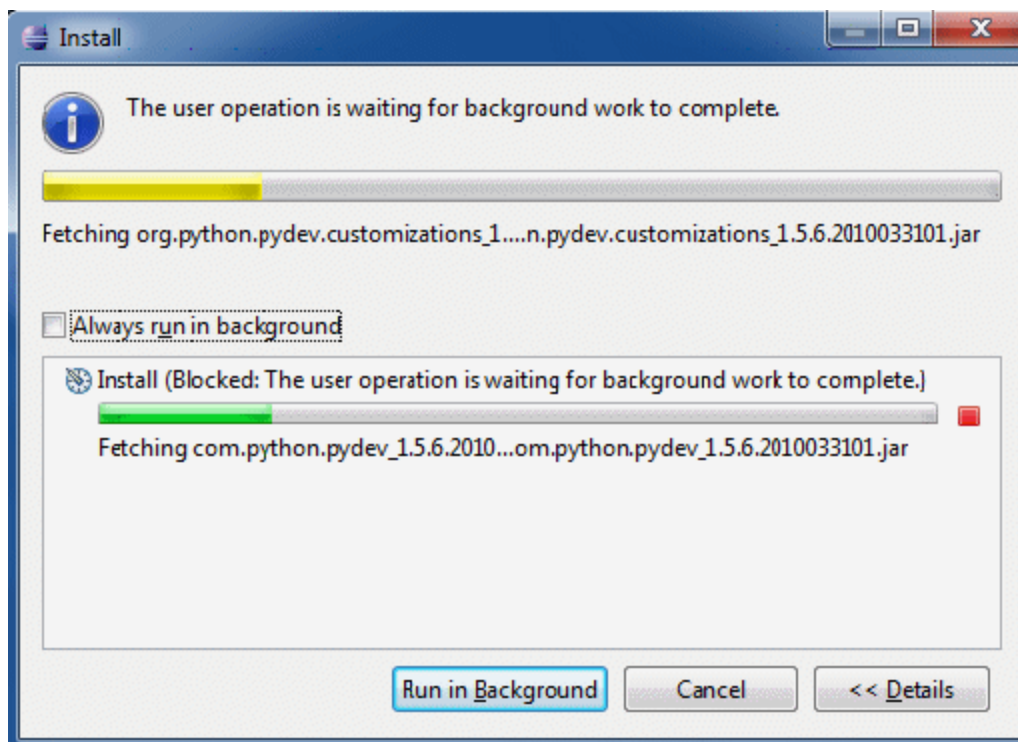
4. Configurar as Extensões PyDev

- a. Acesse **Ajuda > Instalar Novo Software**, clique em **Adicionar**, digite um nome para o plugin PyDev e, no campo Local, adicione a URL do site em que **pydev** pode ser baixado: **http://pydev.org/updates**. Clique em **OK**.



Observação: As Extensões PyDev e PyDev agora são mescladas em um único plugin porque as Extensões PyDev agora têm código-fonte aberto. Para ver informações adicionais, visite <http://pydev.org>.

- b. Na janela que for aberta, selecione **Pydev**. O segundo plugin serve para UI voltada a tarefas. Clique em **Avançar**, verifique os detalhes da instalação e clique em **Avançar** novamente.
- c. Aceite o contrato de licença e clique em **Avançar**.
- d. O Pydev está instalado. Se você for solicitado a instalar conteúdo não assinado, confirme clicando em **OK**.

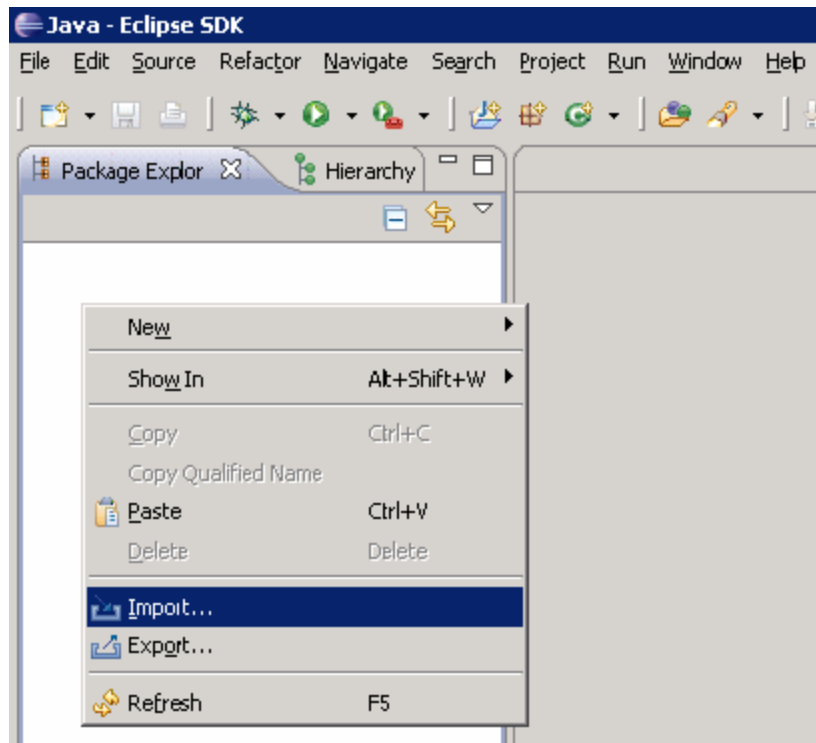


- e. Reinicie o Eclipse.

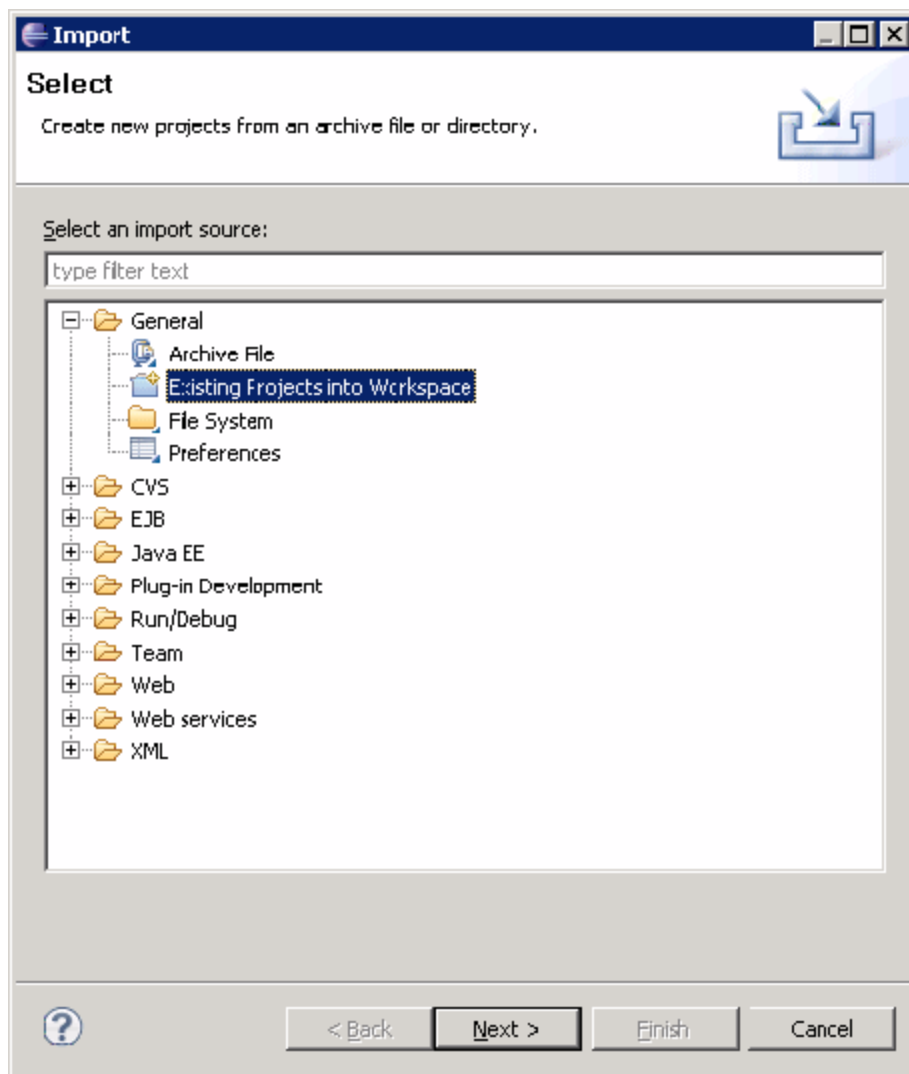
O PyDev agora está instalado na IDE Eclipse. Você tem novas perspectivas no Eclipse e a IDE pode interpretar scripts Python (realce de texto, opções de configuração adicionais e assim por diante).

5. Configurar o espaço de trabalho do Analisador de Descoberta

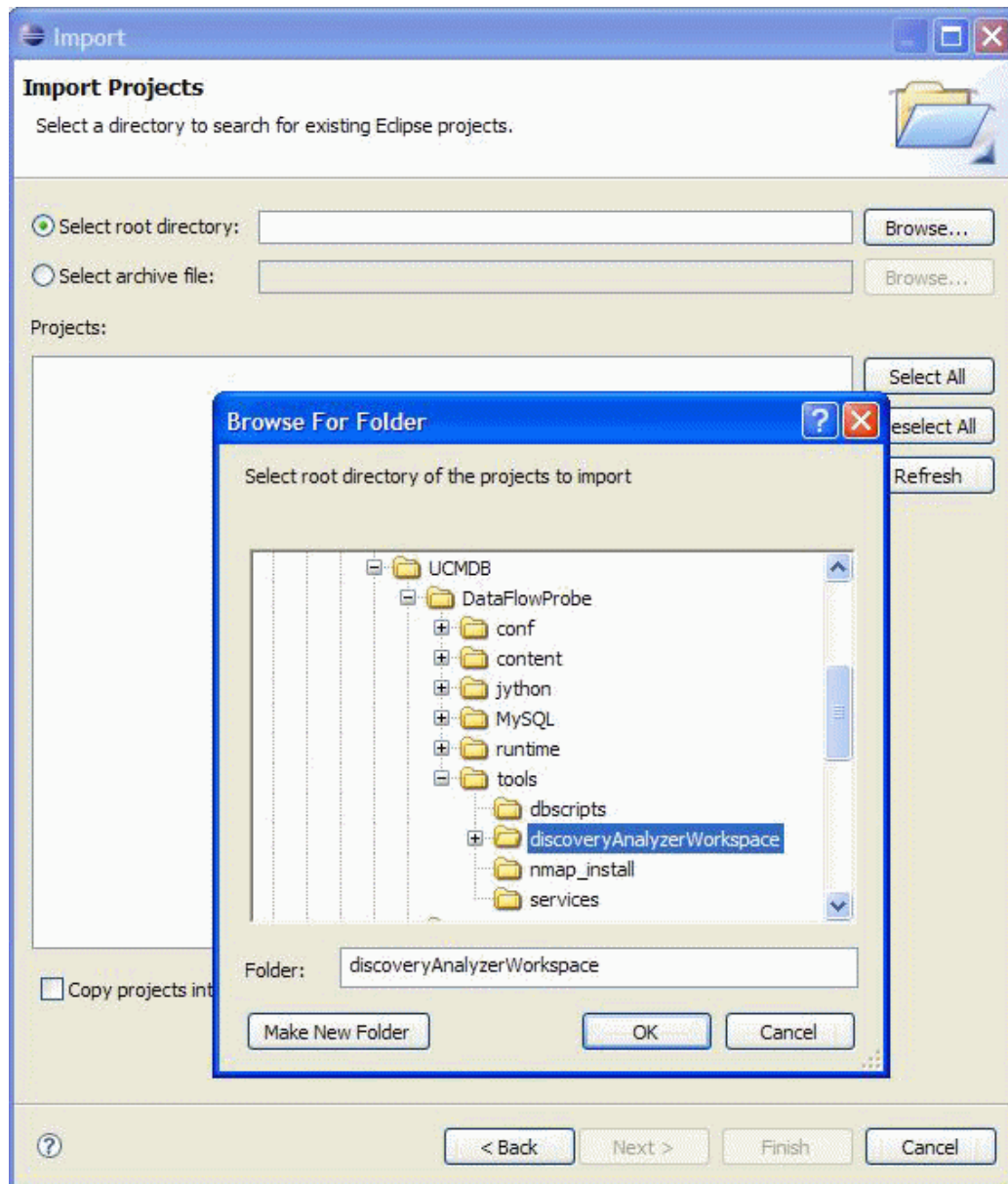
- a. Importe os arquivos necessários: Clique com o botão direito do mouse na área em branco no Explorador de Pacotes e clique em **Importar** para importar o **discoveryAnalyzerWorkspace** pré-configurado, incluído com a instalação da Sonda.



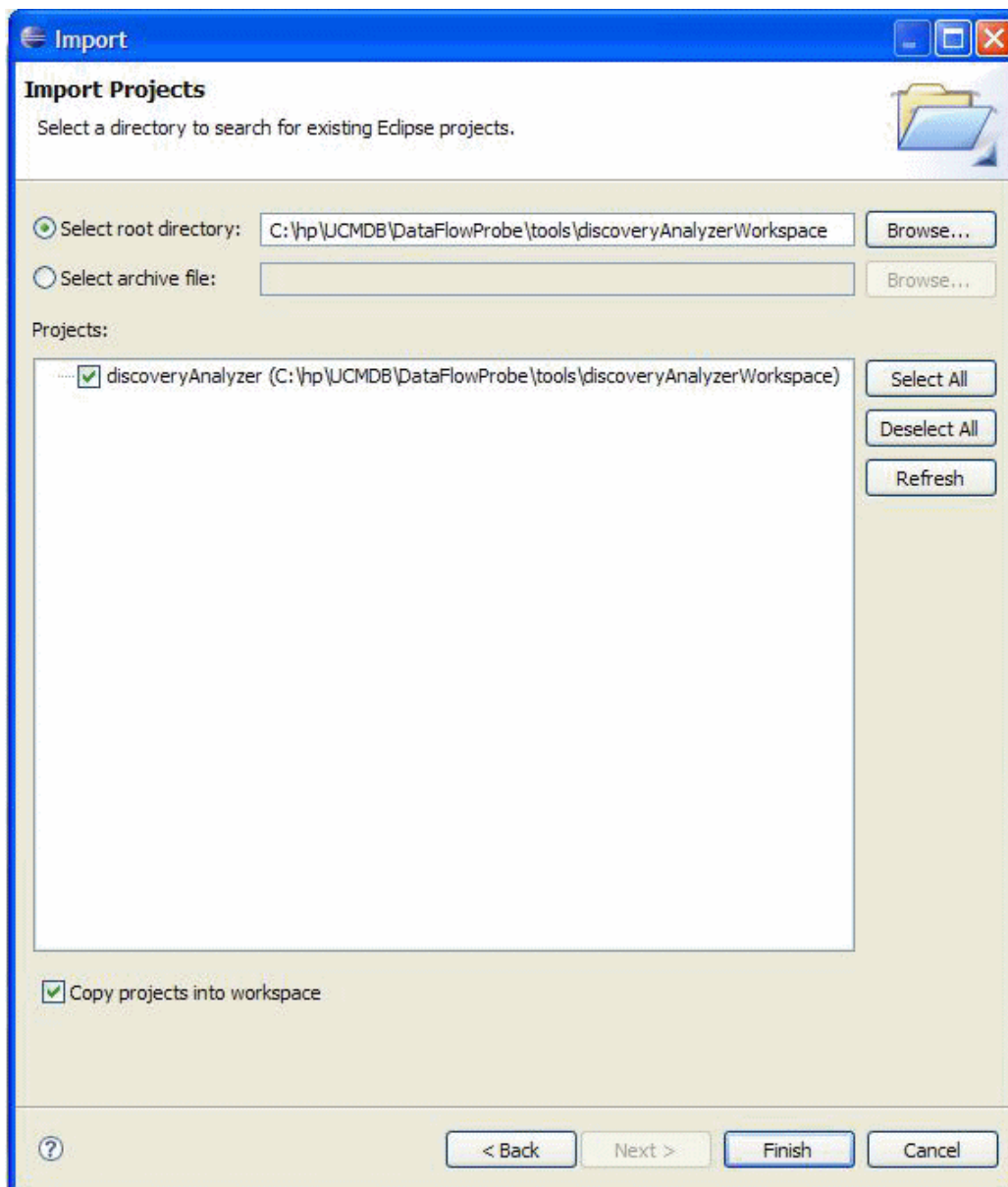
- b. Em **Geral**, selecione **Projetos existentes no espaço de trabalho** para importar o projeto no espaço de trabalho do Eclipse.



- c. Em **Selecionar diretório raiz**, selecione o espaço de trabalho do Analisador, normalmente localizado em:
C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyserWorkspace.
- d. Selecione **Copiar projetos no espaço de trabalho** para criar uma cópia real do espaço de trabalho existente. Esta é uma etapa importante: Em caso de falha, você pode importar novamente o **discoveryAnalyserWorkspace** original.



- e. Clique em **Concluir** para iniciar a Importação.

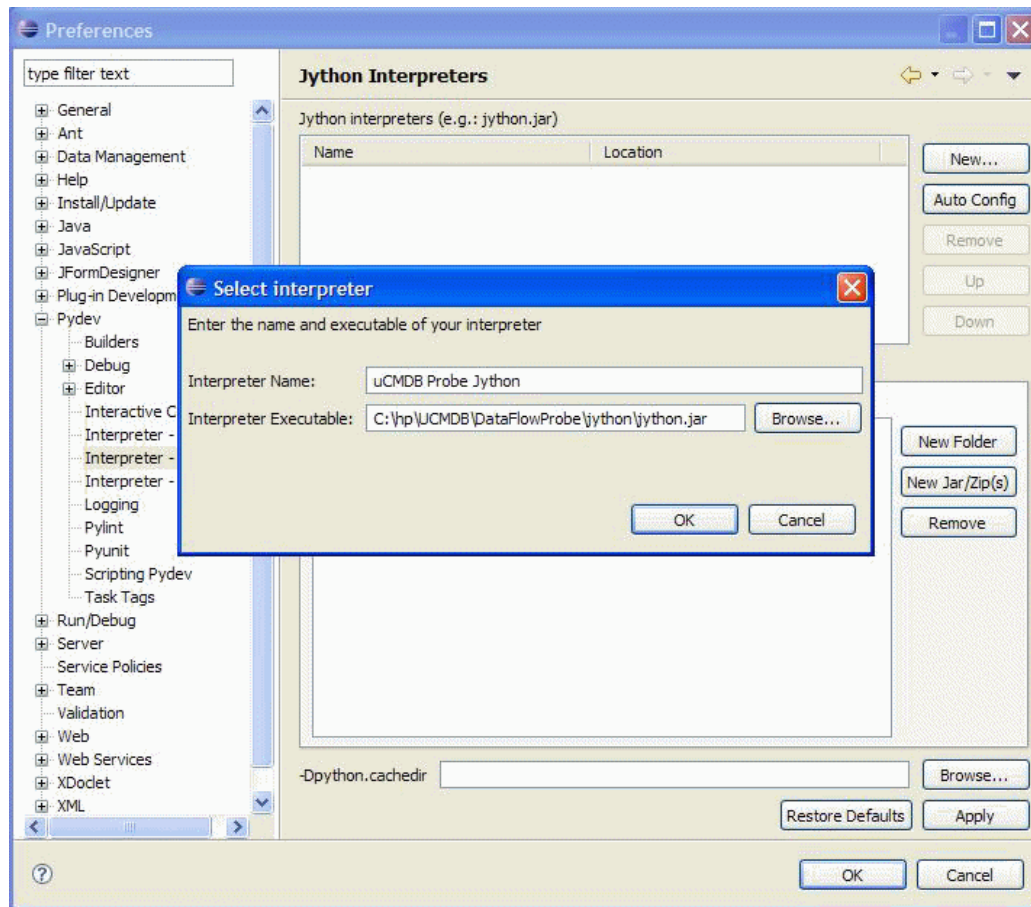


6. Configurar o caminho de classe e o interpretador

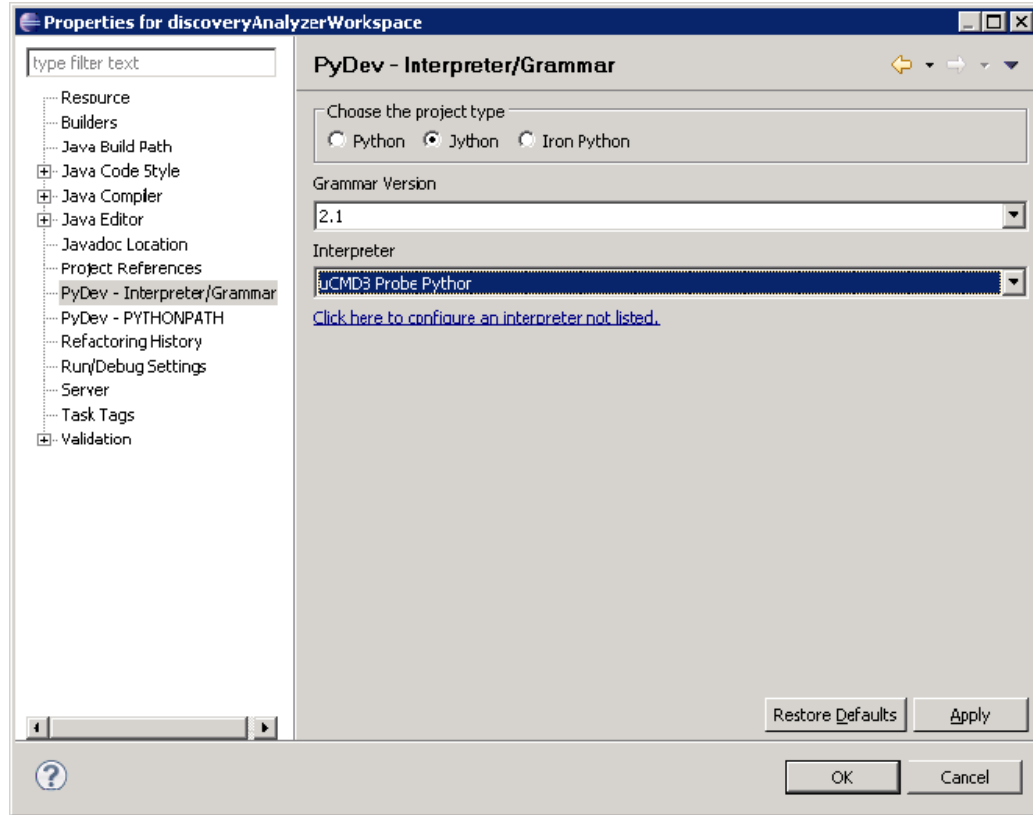
- a. Clique com o botão direito do mouse em **discoveryAnalyzerWorkspace** e selecione **Propriedades** para exibir as configurações específicas do Projeto.
- b. Vá para **Pydev > Interpreter/Grammar** e clique em **Configurar um interpretador nas preferências relacionadas antes de prosseguir**.

Essa etapa configura o mesmo interpretador Jython que a Sonda está usando para garantir que os scripts não sejam interpretados por uma versão Jython diferente.

- c. Clique em **Novo**, digite um nome para o interpretador e selecione o arquivo na seguinte pasta: **C:\hp\UCMDB\DataFlowProbe\jython\jython.jar**.



- d. Clique em **OK**. Se uma janela for exibida solicitando para você selecionar as pastas que devem ser importadas para o caminho de sistema Python, não altere nada (deve ser **C:\hp\UCMDB\DataFlowProbe\jython** e **C:\hp\UCMDB\DataFlowProbe\jython\lib**) e clique em **OK**.
- e. Clique em **Aplicar** e em **OK**.
- f. Clique em **Interpretador** e selecione o interpretador recém-criado.

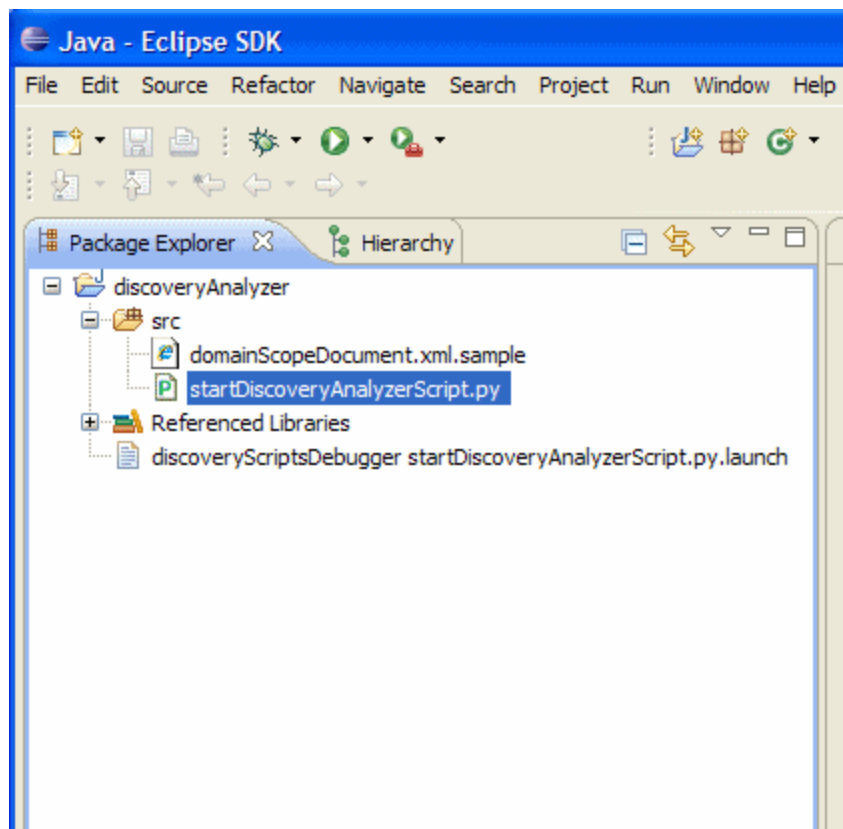


- g. Clique em **Aplicar** e em **OK**.

O interpretador Jython agora é o mesmo que a Sonda está usando.

7. Executar o Analisador de Descoberta

- a. Adicione um ponto de interrupção ao script Jython para ser depurado.
- b. Para iniciar o Analisador de Descoberta, selecione **startDiscoveryAnalyzerScript.py** no projeto **discoveryAnalyzerWorkspace\src** project. Clique com o botão direito do mouse no arquivo e escolha **Depurar como > Execução de Jython**.

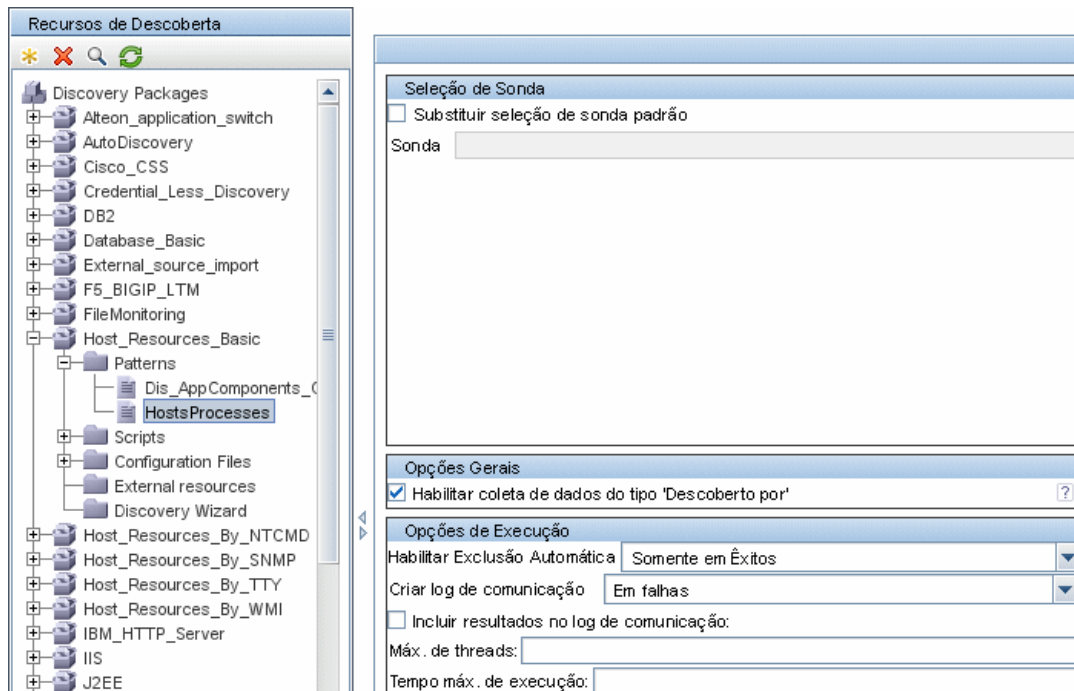


Registrar código do DFM

Pode ser muito útil registrar uma execução inteira, incluindo todos os parâmetros, por exemplo, ao depurar e testar código. Essa tarefa descreve como registrar uma execução inteira com todas as variáveis relevantes. Além disso, você pode visualizar informações de depuração adicionais que normalmente não são impressas nos logs de arquivo mesmo no nível de depuração.

Para registrar um código do DFM:

1. Acesse **Gerenciamento de Fluxo de Dados > Painel de Controle de Descoberta**. Clique com o botão direito do mouse no trabalho cuja execução precisa ser registrada em log e selecione **Ir para Adaptador** para abrir o aplicativo Gerenciamento do Adaptador.
2. Localize o painel **Opções de Execução** na guia **Configuração do Adaptador**, conforme exibido abaixo.



3. Altere a caixa **Criar logs de comunicação** para **Sempre**. Para ver detalhes sobre como configurar opções de registro em log, consulte "[Painel Opções de Execução](#)" na página 1 no Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB.

O exemplo a seguir é o arquivo de log XML que é criado quando o trabalho `Conexão de Host por Shell` é executado e a caixa **Criar logs de comunicação** está definida como **Sempre** ou **Em Falhas**:

Nome do trabalho	Acionar dados de EC
------------------	---------------------

```

- <execution jobId="Host Connection by Shell" destinationid="0e9787433d65e4a68839bfa8b224c92d">
- <destination>
  <destinationData name="ip_domain">DefaultDomain</destinationData>
  <destinationData name="hostId" />
  <destinationData name="ip_address">16.59.63.34</destinationData>
  <destinationData name="id">0e9787433d65e4a68839bfa8b224c92d</destinationData>
</destination>
    
```

O exemplo a seguir mostra a mensagem e os parâmetros de rastreamento de pilha:

Rastreamento de pilha

```

- <exec start="18:41:55" duration="2062" type="ssh" credentialsId="f464999bdf5a1e1407b479b6f730d5b">
  <cmd>[CDATA: client_connect]</cmd>
  <result IS_NULL="Y" />
- <error class="com.hp.ucmdb.discovery.probe.services.dynamic.agents.SSHAgentException">
  <message>[CDATA: Failed to connect: Error connecting: Connection refused: connect]</message>
  <stacktrace>
    <frame class="com.hp.ucmdb.discovery.probe.services.dynamic.agents.SSHAgent" method="connect" file="SSHAgent.java">
    <frame class="com.hp.ucmdb.discovery.probe.clients.shell.SSHClient" method="createWrapper" file="SSHClient.java">
    <frame class="com.hp.ucmdb.discovery.probe.clients.BaseClient" method="initPrivate" file="BaseClient.java">
    
```

Bibliotecas e utilitários Jython

Vários scripts de utilitários são usados amplamente nos adaptadores. Esses scripts fazem parte do pacote `AutoDiscovery` e estão localizados em:

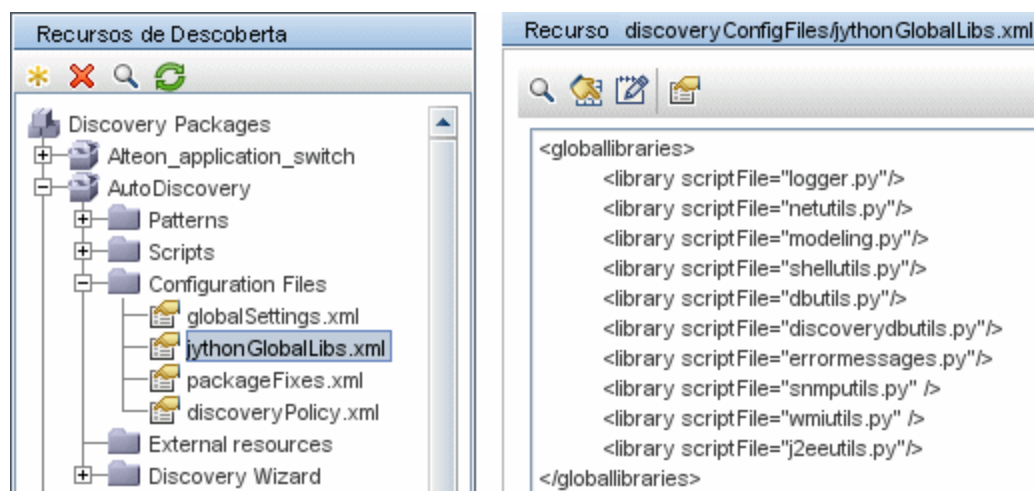
C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts com os outros scripts baixados para o Sonda.

Observação: A pasta `discoveryScript` é criada dinamicamente quando a Sonda inicia o trabalho.

Para usar um dos scripts de utilitário, adicione a seguinte linha de importação à seção de importação do script:

```
import <nome_do_script>
```

A biblioteca Python AutoDiscovery contém scripts de utilitário Jython. Esses scripts de biblioteca são considerados biblioteca externa do DFM. Eles são definidos no arquivo `jythonGlobalLibs.xml` (localizado na pasta **Arquivos de Configuração**).



Cada script que aparece no arquivo `jythonGlobalLibs.xml` é carregado por padrão na inicialização da Sonda, por isso não é necessário usá-los explicitamente na definição do adaptador.

Esta seção inclui os seguintes tópicos:

- "logger.py" abaixo
- "modeling.py" na página seguinte
- "netutils.py" na página seguinte
- "shellutils.py" na página seguinte

logger.py

O script **logger.py** contém utilitários de log e funções auxiliares para gerar relatórios de erros. Você pode chamar sua depuração, informações e APIs de erros para gravar nos arquivos de log. As mensagens de log são registradas em **C:\hp\UCMDB\DataFlowProbe\runtime\log**.

As mensagens são inseridas no arquivo de log de acordo com o nível de depuração definido para o anexador PATTERNS_DEBUG no arquivo

C:\hp\UCMDB\DataFlowProbe\conf\log\probeMgrLog4j.properties. Por padrão, o nível é DEBUG.) Para obter detalhes, consulte "Níveis de gravidade de erro" na página 80.

```
#####
##### 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\probeMgr-
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
```

As mensagens de informações e de erros também aparecem no console Prompt de Comando.

Há dois conjuntos de APIs:

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

O primeiro conjunto emite a concatenação de todos os seus argumentos de cadeia de caracteres no nível de log apropriado e o segundo conjunto emite a concatenação, além de emitir o rastreamento de pilha da exceção de lançamento mais recente, para fornecer mais informações, por exemplo:

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

modeling.py

O script **modeling.py** contém APIs para criar hosts, IPs, ECs de processo e assim por diante. Essas APIs permitem a criação de objetos comuns e tornam o código mais legível. Por exemplo:

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

netutils.py

A biblioteca **netutils.py** é usada para recuperar informações de rede e TCP, por exemplo, recuperar nomes de sistemas operacionais, verificar se um endereço MAC é válido, verificar se um endereço IP é válido e assim por diante. Por exemplo:

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

shellutils.py

A biblioteca **shellutils.py** fornece uma API para executar comandos de shell e recuperar o status

final de um comando executado, além de permitir a execução de vários comandos com base nesse status final. A biblioteca é inicializada com um Cliente Shell e usa o cliente para executar comandos e recuperar resultados. Por exemplo:

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

Capítulo 3

Mensagem de erro

Este capítulo inclui:

Visão geral de mensagens de erro	77
Convenções da criação de mensagens	77
Níveis de gravidade de erro	80

Visão geral de mensagens de erro

Durante a descoberta, muitos erros podem ser revelados, por exemplo, falhas de conexão, problemas de hardware, exceções, tempo limite esgotado e assim por diante. O DFM exibe esses erros no Painel de Controle de Descoberta, tanto em Modo Básico quanto em Modo Avançado, sempre que o fluxo de descoberta regular não tiver êxito. You can drill down from the Trigger CI that caused the problem to view the error message itself.

O DFM diferencia entre erros que podem ser ignorados (por exemplo, um host fora de alcance) e erros que precisam ser corrigidos (por exemplo, problemas de credenciais ou arquivos DLL ou de configuração ausentes). Além disso, o DFM relata erros uma vez, mesmo que o erro ocorra em execuções sucessivas, e relata um erro inclusive se ele ocorrer somente uma vez.

Ao criar um pacote, você pode adicionar ao pacote mensagens apropriadas como recursos. Durante a implementação do pacote, as mensagens também são implementadas no local correto. As mensagens devem obedecer a convenções, conforme descrito em "[Convenções da criação de mensagens](#)" abaixo.

O DFM dá suporte a mensagens de erro multilíngues. Você pode traduzir as mensagens que escreve de modo que sejam exibidas no idioma local.

Para ver detalhes sobre como procurar erros, consulte "[Painel Status de Descoberta](#)" na página 1 no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Para ver detalhes sobre os log de comunicação, consulte "[Painel Opções de Execução](#)" na página 1 no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Convenções da criação de mensagens

- Cada erro é identificado por um código de mensagem de erro e diversos argumentos (**int**, **String** []). Um erro específico é definido pelo seu código de mensagem e seus vários argumentos. A matriz de parâmetros pode ser nula.
- Cada código de erro é associado a uma **mensagem curta**, que é uma cadeia fixa, e uma **mensagem detalhada**, que é uma cadeia de modelo com zero ou mais argumentos. Assuma-se a correspondência entre o número de argumentos no modelo e o número real de parâmetros.

Exemplo de código de mensagem de erro:

10234 pode representar um erro com a mensagem curta:

```
Erro de conexão
```

e a mensagem detalhada:

```
Impossível conectar via o protocolo {0} devido ao tempo limite de {1} ms
```

em que

{0} = primeiro argumento: nome do protocolo

{1} = segundo argumento: duração do tempo limite em ms

Esta seção também inclui os seguintes tópicos:

- ["Conteúdo de arquivo de propriedade" abaixo](#)
- ["Arquivo de propriedade de mensagens de erro" abaixo](#)
- ["Convenções de nomenclatura de local" abaixo](#)
- ["Códigos de mensagem de erro" na página seguinte](#)
- ["Erros de conteúdo não classificado" na página seguinte](#)
- ["Mudanças na metodologia" na página 80](#)

Conteúdo de arquivo de propriedade

Um arquivo de propriedade deve conter duas chaves para cada código de mensagem de erro. Por exemplo, para o erro 45:

- **DDM_ERROR_MESSAGE_SHORT_45**. Descrição curta do erro.
- **DDM_ERROR_MESSAGE_LONG_45**. Descrição longa do erro (pode conter parâmetros, por exemplo: **{0},{1}**).

Arquivo de propriedade de mensagens de erro

Um arquivo de propriedade contém o mapeamento entre o código da mensagem de erro e duas mensagens (curta e detalhada).

Após implementar um arquivo de propriedade, seus dados são mesclados com os dados existentes, ou seja, novos códigos de mensagem de erro são adicionados e os códigos antigos são substituídos.

Arquivos de propriedade de infraestrutura são parte do pacote **AutoDiscoveryInfra**.

Convenções de nomenclatura de local

- Para o local padrão: **<nome do arquivo>.properties.errors**
- Para um local específico: **<nome do arquivo>_xx.properties.errors**
em que **xx** é o local (por exemplo, **infraerr_fr.properties.errors** ou **infraerr_en_us.properties.errors**).

Códigos de mensagem de erro

Os códigos de erro a seguir estão incluídos por padrão no HP Universal CMDB. Você pode adicionar a essa lista suas próprias mensagens de erro.

Nome do Erro	Código do Erro	Descrição
Interno	100-199	Geralmente resolvidos de exceções resultantes da execução de scripts Jython
Conexão	200-299	Falha da conexão; agente ausente na máquina-alvo; destino fora de alcance e assim por diante
Relacionados a credenciais	300-399	Permissão negada; tentativa de conexão bloqueada devido à falta de credenciais
Tempo limite	400-499	Tempo limite alcançado durante conexão/comando
Comportamento inesperado ou inválido	500-599	Arquivos de configuração ausentes; interrupções inesperadas e assim por diante
Recuperação de informações	600-699	Informações ausentes nas máquinas-alvo; falha ao solicitar informações ao agente e assim por diante
Relacionados a recursos	700-799	Erros relacionados à falta de memória ou clientes que não foram liberados corretamente
Análise	800-899	Erro ao analisar texto
Codificação	900	Erro na entrada de dados; codificação sem suporte
Relacionados a SQL	901-903, 924	Erros recebidos durante operações SQL
Relacionados a HTTP	904-909	Erros gerados durante conexões HTTP; derivados de códigos de erro HTTP.
Específicos a aplicativos	910-923	Erro resultante de problemas específicos de aplicativos, por exemplo, versão LSOF incorreta, gerenciadores de fila ausentes e assim por diante

Erros de conteúdo não classificado

Para dar suporte a conteúdo antigo sem causar regressões, os métodos relevantes do SDK e do aplicativo lidam de modo diferente com os erros com código de mensagem 100 (erro de script não classificado).

Esses erros não são agrupados (ou seja, não são considerados erros do mesmo tipo) de acordo com seu código de mensagem, e sim são agrupados pelo conteúdo da mensagem. Isso significa

que se um script retorna um erro via métodos antigos e obsoletos (com texto de mensagem mas sem um código de erro), essas mensagens recebem o mesmo código de erro, porém de acordo com os métodos relevantes do SDK ou do aplicativo, mensagens diferentes são exibidas como erros diferentes.

Mudanças na metodologia

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

Os métodos a seguir foram adicionados à interface:

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

Os métodos antigos a seguir ainda têm suporte com o propósito de compatibilidade com versões anteriores, porém são marcados como preteridos:

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

Níveis de gravidade de erro

Quando um adaptador termina de ser executado em um EC acionador, ele retorna um status. Se nenhum erro ou aviso for retornado, o status será **Sucesso**.

Os níveis de gravidade são listados abaixo, do escopo mais estreito ao mais amplo:

Erros Fatais

Esse nível inclui erros graves, como um problema com a infraestrutura, arquivos DLL ausentes ou exceções:

- Falha ao gerar a tarefa (a sonda não foi encontrada, variáveis não foram localizadas etc.)
- Não é possível executar o script
- O processamento dos resultados não ocorre no servidor e os dados não são gravados no CMDB

Erros

Esse nível identifica problemas que impedem que o DFM recupere dados. Analise esses erros, uma vez que exigem que medidas sejam tomadas (por exemplo, aumentar o tempo limite, mudar um intervalo, mudar um parâmetro, adicionar outra credencial de usuário etc).

- Em casos em que a intervenção do usuário pode ajudar, um erro é retornado, o qual pode envolver um problema de credenciais ou rede que talvez exija investigação adicional. (Esses não são erros de descoberta, e sim de configuração.)
- Falha interna, geralmente causada por comportamento inesperado da máquina ou aplicativo descoberto, por exemplo, arquivos de configuração ausentes, etc.

Avisos

Quando uma execução foi bem-sucedida, mas produziu erros que não foram graves mas que

devem ser relatados, o DFM define a gravidade como **Aviso**. Esses ECs devem ser verificados quanto a dados ausentes, antes do início de uma sessão de depuração mais detalhada. Um **Aviso** pode incluir mensagens sobre a falta de um agente instalado no host remoto, ou sobre dados inválidos que fazem com que um atributo não seja calculado corretamente.

- Agente de conexão ausente (SNMP, WMI)
- A descoberta foi bem-sucedida, porém nem todas as informações disponíveis foram descobertas

Capítulo 4

Desenvolvimento de adaptadores de banco de dados genéricos

Este capítulo inclui:

Visão geral dos adaptadores de banco de dados genéricos	83
Consultas TQL para o adaptador de banco de dados genérico	83
Reconciliação	84
Hibernar como provedor de JPA	84
Preparar criação do adaptador	87
Preparar o pacote de adaptadores	91
Configurar o adaptador - Método mínimo	94
Configurar o adaptador - Método avançado	99
Implementar um plugin	103
Implantar o adaptador	105
Editar o adaptador	106
Criar um ponto de integração	106
Criar uma visualização	106
Calcular os resultados	107
Visualizar os resultados	107
Visualizar relatórios	108
Habilitar arquivos de log	108
Usar o Eclipse para mapear entre atributos de TEC e tabelas de banco de dados	108
Arquivos de configuração do adaptador	115
Conversores prontos	137
Plug-ins	141
Exemplos de Configuração	141
Arquivos de log do adaptador	150
Referências externas	151
Solução de problemas e limitações	151

Visão geral dos adaptadores de banco de dados genéricos

A finalidade da plataforma de adaptador de banco de dados genérico é criar adaptadores que podem se integrar com RDBMSs (sistemas de gerenciamento de bancos de dados relacionais) e executar consultas TQL e trabalhos de população com base no banco de dados. Os sistemas RDBMS aceitos pelo adaptador de banco de dados genérico são Oracle, Microsoft SQL Server e MySQL.

Esta versão da implementação do adaptador de banco de dados se baseia em um padrão JPA (Java Persistence API) tendo a biblioteca ORM do Hibernate como provedor de persistência.

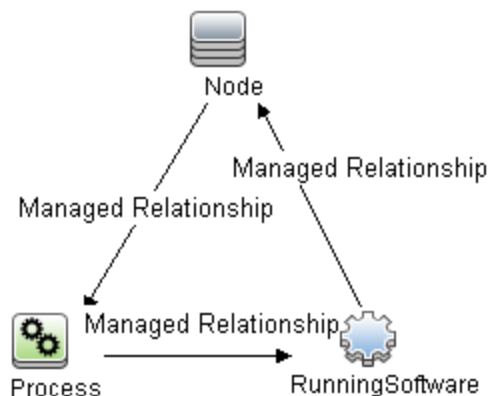
Consultas TQL para o adaptador de banco de dados genérico

Em trabalhos de população, cada layout obrigatório de um EC deve ser verificado na caixa de diálogo Configurações de Layout no Modeling Studio. Para obter detalhes, consulte "[Caixa de diálogo Propriedades do Nó de Consulta/Relacionamento](#)" na página 1 no *Guia de Modelagem do HP Universal CMDB*. É importante observar que um EC pode exigir que um atributo seja identificado e, sem esses atributos, o EC falhará em ser adicionado ao UCMDB.

As seguintes limitações existem nas consultas TQL calculadas somente pelo Adaptador de Banco de Dados Genérico:

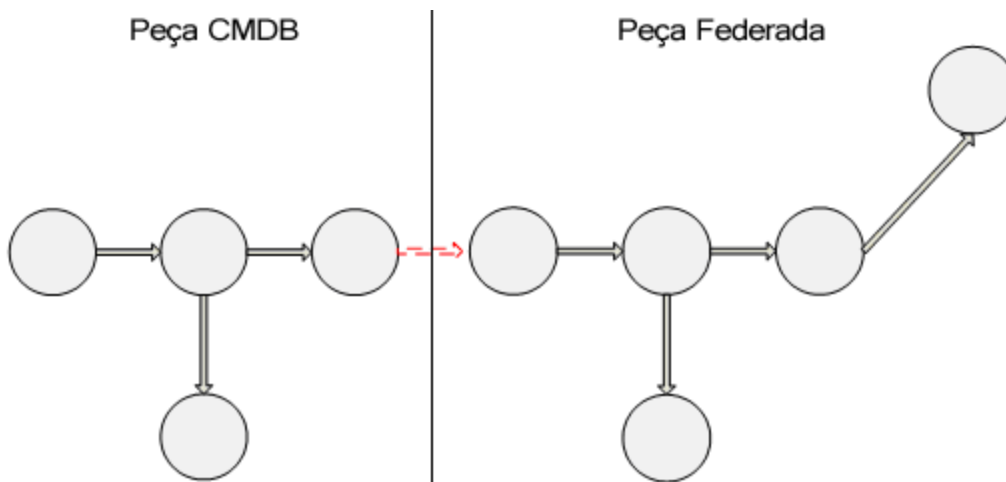
- não há suporte a subgráficos
- não há suporte a relacionamentos compostos
- não há suporte a ciclos ou partes de ciclo

A consulta TQL a seguir é um exemplo de ciclo:



- O layout de função não é aceito.
- Cardinalidade 0..0 não é aceita.
- O relacionamento `Join` não é aceito.

- Condições de qualificador não são aceitas.
- Para se conectar entre dois ECs, um relacionamento na forma de uma tabela ou chave estrangeira precisa existir na fonte de dados de dados externa.



Reconciliação

A reconciliação é executada como parte do cálculo TQL no lado do adaptador. Para que a reconciliação ocorra, o lado do CMDB é mapeado para uma entidade federada denominada TEC de reconciliação.

Mapeamento. Cada atributo no CMDB é mapeado para uma coluna na fonte de dados.

Apesar de o mapeamento ser executado diretamente, as funções de transformação nos dados de mapeamento também são aceitas. Você pode adicionar novas funções através de código Java (por exemplo, lowercase, uppercase). A finalidade dessas funções é permitir conversões de valor (valores armazenados no CMDB em um formato e no banco de dados federado em outro formato).

Observação:

- Para conectar o CMDB e a fonte de banco de dados externa, é necessário haver uma associação apropriada no banco de dados. Para obter detalhes, consulte "[Pré-requisitos](#)" na página 87.
- A reconciliação com a ID do CMDB também é aceita.
- A reconciliação com a ID Global também é aceita.

Hibernar como provedor de JPA

O Hibernate é uma ferramenta de mapeamento relacional a objeto, que permite mapear classes Java em tabelas em vários tipos de bancos de dados relacionais (por exemplo, Oracle e Microsoft SQL Server). Para obter detalhes, consulte "[Limitações funcionais](#)" na página 152.

Em um mapeamento elementar, cada classe Java é mapeada para uma única tabela. Um mapeamento mais avançado permite o mapeamento de herança (como pode ocorrer no banco de dados do CMDB).

Outros recursos aceitos incluem mapeamento de uma classe para várias tabelas, suporte a coleções e associações de tipos um-para-um, um-para-muitos e muitos-para-um. Para obter detalhes, consulte "Associações" na página seguinte a seguir.

Para as nossas finalidades, não é necessário criar classes Java. O mapeamento é definido dos TECs de modelo de classe do CMDB às tabelas de banco de dados.

Esta seção também inclui os seguintes tópicos:

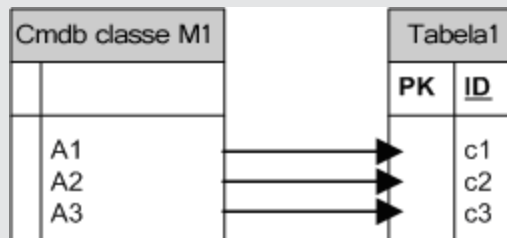
- "Exemplos de mapeamento relacional de objeto" abaixo
- "Associações" na página seguinte
- "Facilidade de uso" na página seguinte

Exemplos de mapeamento relacional de objeto

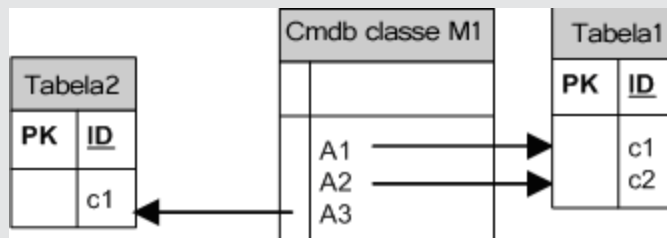
Os exemplos a seguir descrevem o mapeamento relacional de objeto:

Exemplo de uma classe do CMDB mapeada para uma tabela do banco de dados:

A classe M1, com atributos A1, A2 e A3, é mapeada para as colunas c1, c2 e c3 da tabela 1. Isso significa que qualquer instância de M1 tem uma linha de correspondência na tabela 1.

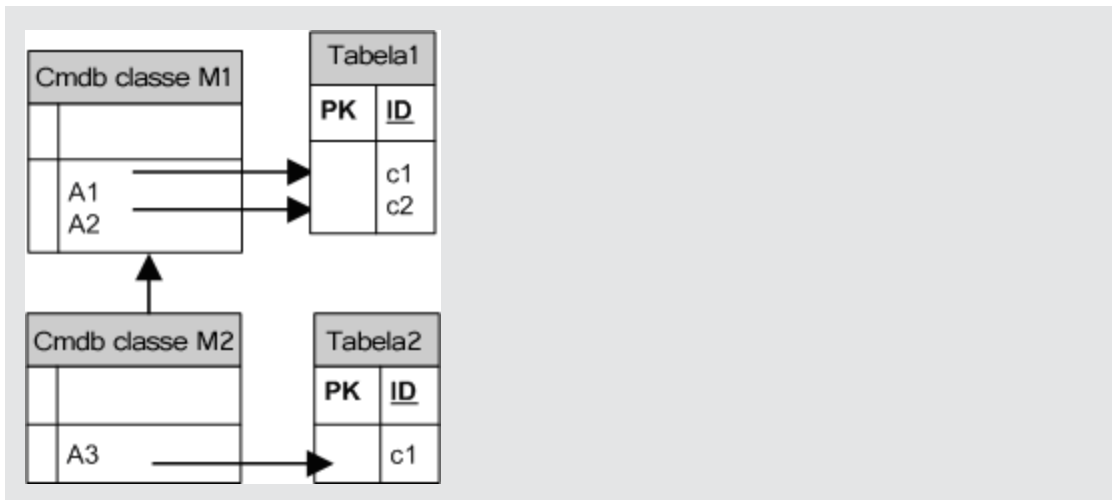


Exemplo de uma classe do CMDB mapeada para duas tabelas do banco de dados:



Exemplo de herança:

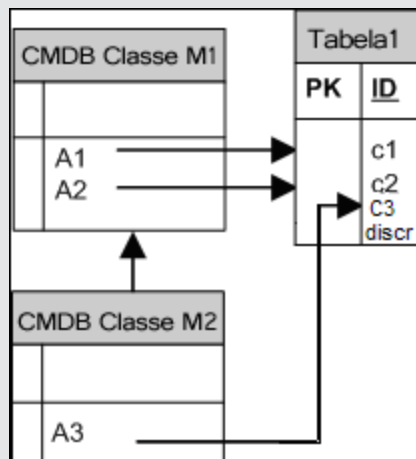
Este caso é usado no CMDB, em que cada classe tem a própria tabela de banco de dados.



Exemplo de herança de uma única tabela com discriminador:

Uma hierarquia inteira de classes é mapeada para uma única tabela de banco de dados, cujas colunas consistem em um superconjunto de todos os atributos das classes mapeadas. A tabela também contém uma coluna adicional (*Discriminator*), cujo valor indica qual classe específica deve ser mapeada para essa entrada.

Ao usar os recursos de discriminador, você não pode ignorar uma classe na hierarquia; ou seja, como C3 herda de C2, que por sua vez herda de C1, não é possível simplesmente definir C1 e C3, mas sim é necessário definir as três classes.



Associações

Existem três tipos de associações: um-para-muitos, muitos-para-um e muitos-para-muitos. Para se conectar entre os diferentes objetos de banco de dados, uma dessas associações precisa ser definida usando uma coluna de chave estrangeira (para o caso um-para-muitos) ou uma tabela de mapeamento (para o caso de muitos-para-muitos).

Facilidade de uso

Como o esquema JPA é muito amplo, um arquivo XML simplificado é fornecido para facilitar as definições.

O caso de uso deste arquivo XML é assim: os dados federados são modelados em uma classe federada. Essa classe tem bastantes relacionamentos muitos-para-um com uma classe do CMDB não-federada. Além disso, só há um tipo de relacionamento possível entre a classe federada e a classe não-federada.

Preparar criação do adaptador

Esta tarefa descreve os preparativos necessários para criar um adaptador.

Observação: Você pode visualizar exemplos do adaptador de banco de dados genérico na API do UCMDB. Especificamente, o exemplo de Adaptador DDMi contém um arquivo **orm.xml** complicado, além das implementações para algumas interfaces de plugin.

Esta tarefa inclui as seguintes etapas:

- "Pré-requisitos" abaixo
- "Criar um tipo de EC" na página 89
- "Criar um relacionamento" na página 89

1. Pré-requisitos

Para validar se é possível usar o adaptador de banco de dados com o seu banco de dados, verifique estes itens:

- Se as classes de reconciliação e seus atributos (também conhecidos como multinós) existem no banco de dados. Por exemplo, se a reconciliação for executada por nome de nó, verifique se há uma tabela que contém uma coluna com nomes de nó. Se a reconciliação estiver sendo executada de acordo com o nó `cmdb_id`, verifique se há uma coluna com IDs do CMDB que corresponda aos IDs do CMDB dos nós no CMDB. Para ver detalhes sobre reconciliação, consulte "Reconciliação" na página 84.

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

- Para correlacionar dois TECs com um relacionamento, deve haver dados de correlação entre as tabelas de TEC. A correlação pode ser mediante uma coluna de chave estrangeira ou mediante uma tabela de mapeamento. Por exemplo, para correlacionar entre nó e ticket,

deve haver uma coluna na tabela de tickets que contém a ID de nó, uma coluna na tabela de nó com a ID de ticket que está conectada a ela ou uma tabela de mapeamento cujo `end1` é a ID de nó e `end2` é a ID de ticket. Para ver detalhes sobre dados de correlação, consulte "Hibernar como provedor de JPA" na página 84.

A tabela a seguir mostra a coluna `NODE_ID` da chave estrangeira:

<code>NODE_ID</code>	<code>CARD_ID</code>	<code>CARD_TYPE</code>	<code>CARD_NAME</code>
2015	1	Controlador de barramento serial	Intel 82801EB USB Universal Host Controller
3581	2	Sistema	Intel 631xESB/6321ESB/3100 Chipset LPC
3581	3	Vídeo	ATI ES1000
3581	4	Periférico de sistema base	HP ProLiant iLO 2 Legacy Support Function

- Cada TEC pode ser mapeado para uma ou mais tabelas. Para mapear um TEC para mais de uma tabela, verifique se há uma tabela primária cuja chave primária exista nas outras tabelas e seja uma coluna de valor exclusivo.

Por exemplo, um ticket é mapeado para duas tabelas: `ticket1` e `ticket2`. A primeira tabela tem as colunas `c1` e `c2` e a segunda tabela tem as colunas `c3` e `c4`. Para permitir que elas sejam consideradas como uma só tabela, ambas precisam ter a mesma chave primária. Como alternativa, a primeira chave primária da tabela pode ser uma coluna na segunda tabela.

No exemplo a seguir, as tabelas compartilham a mesma chave primária denominada `CARD_ID`:

<code>CARD_ID</code>	<code>CARD_TYPE</code>	<code>CARD_NAME</code>
1	Controlador de barramento serial	Intel 82801EB USB Universal Host Controller
2	Sistema	Intel 631xESB/6321ESB/3100 Chipset LPC
3	Vídeo	ATI ES1000
4	Periférico de sistema base	HP ProLiant iLO 2 Legacy Support Function

<code>CARD_ID</code>	<code>CARD_VENDOR</code>
1	Hewlett-Packard Company
2	(Controlador de host USB padrão)

CARD_ID	CARD_VENDOR
3	Hewlett-Packard Company
4	(Dispositivos de sistema padrão)
5	Hewlett-Packard Company

2. Criar um tipo de EC

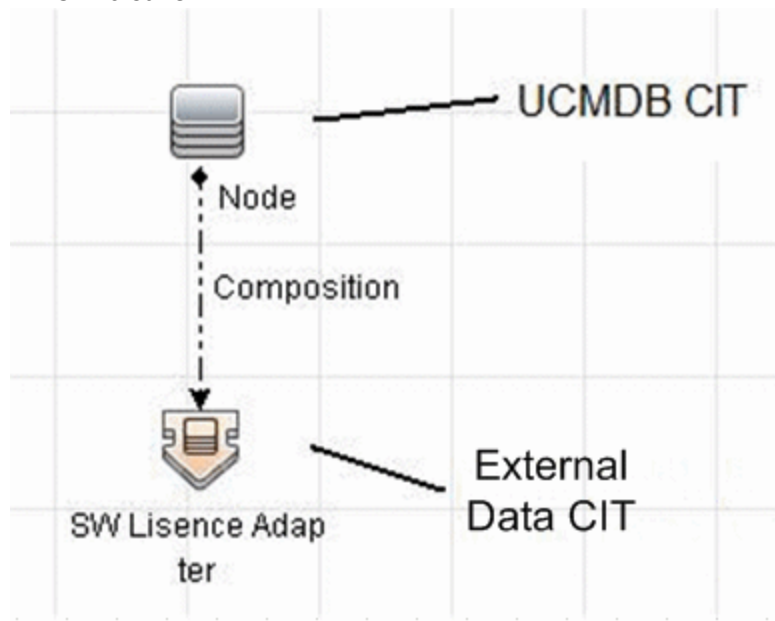
Nesta etapa, você cria um TEC que representa os dados no RDBMS (a fonte de dados externa).

- No UCMDB, acesse o Gerenciador de Tipo de EC e crie um novo Tipo de EC. Para obter detalhes, consulte ["Criar um tipo de EC"](#) na página 1 no *Guia de Modelagem do HP Universal CMDB*.
- Adicione os atributos necessários ao TEC, como o último horário de acesso, o fornecedor e assim por diante. Esses são os atributos que o adaptador recuperará da fonte de dados externa e importará para as visualizações do CMDB.

3. Criar um relacionamento

Nesta etapa, você adiciona um relacionamento entre o TEC do UCMDB e o novo TEC que representa os dados da fonte de dados externa.

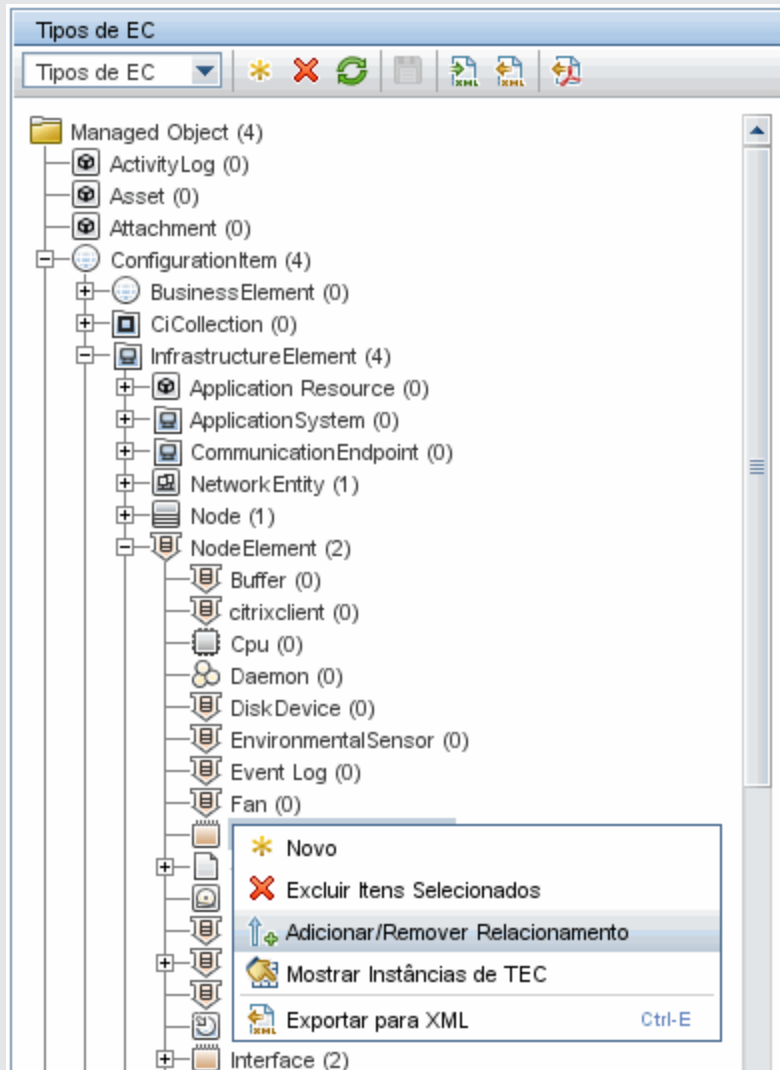
Adicione relacionamentos válidos apropriados ao novo TEC. Para obter detalhes, consulte ["Caixa de diálogo Adicionar/Remover Relacionamento"](#) na página 1 no *Guia de Modelagem do HP Universal CMDB*.



Observação: Nesta etapa, você ainda não pode visualizar os dados federados ou popular os dados externos, já que ainda não definiu o método para importar os dados.

Exemplo de criação de um relacionamento Containment:

a. No Gerenciador de TEC, selecione os dois TECs:



- b. Crie um relacionamento **Containment** entre dois TECs:



Preparar o pacote de adaptadores

Nesta etapa, você localiza e configura o pacote de adaptadores de banco de dados genérico.

1. Localize o pacote **db-adapter.zip** na pasta **C:\hp\UCMDB\UCMDBServer\content\adapters**.
2. Extraia o pacote para um diretório temporário local.
3. Edite o arquivo XML do adaptador:
 - Abra o arquivo **discoveryPatterns\db_adapter.xml** em um editor de texto.
 - Localize o atributo **adapter id** e substitua o nome:

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

Se o adaptador oferecer suporte a dados de replicação, o recurso a seguir deverá ser adicionado ao elemento **<adapter-capabilities>**:

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

O ID ou rótulo de exibição aparece na lista de adaptadores no painel Pontos de Integração do HP Universal CMDB.

Ao criar um Adaptador de BD Genérico, não é necessário editar a marcação **changes-source** na marcação **support-replicatiion-data**. Se o plug-in **FcmdbPluginForSyncGetChangesTopology** estiver implementado, a topologia alterada da última execução será retomada. Se o plug-in não estiver implementado, toda a topologia será retomada e a exclusão automática será realizada de acordo com os ECs retomados.

Para ver detalhes sobre como popular os dados do CMDB, consulte "[Página Integration Studio](#)" na [página 1](#) no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

- Se o adaptador está usando o mecanismo de mapeamento da versão 8.x (significando que ele não está usando o novo mecanismo de mapeamento de reconciliação), substitua o elemento a seguir:

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

por:

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

Para reverter para o novo mecanismo de mapeamento, retorne o elemento para o seguinte valor:

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

- Localize a definição **category**:

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

Altere o nome da categoria **Generic** para a categoria de sua preferência.

Observação: Os adaptadores cujas categorias são especificadas como **Generic** não estão listados no Integration Studio ao criar um novo ponto de integração.

- A conexão com o banco de dados pode ser descrita usando um nome de usuário (esquema), senha, tipo de banco de dados, nome da máquina de host do banco de dados e nome do banco de dados ou SID.

Para esse tipo de conexão, os parâmetros Têm os seguintes elementos na seção **parameter** do arquivo XML do adaptador:

```
<parameters>  
  <!--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.-->  
  <!--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" mandatory="false" order-index="10" />
  <parameter name="port" display-name="Port" type="integer"
description="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;MySQL;BO" mandatory="false" order-
index="13">Oracle</parameter>
  <parameter name="dbname" display-name="DB Name/SID"
type="string" description="The name of the database or its SID
(in case of Oracle)" mandatory="false" order-index="13" />
  <parameter name="credentialsId" display-name="Credentials
ID" type="integer" description="The credentials to be used"
mandatory="true" order-index="12" />
</parameters>
```

Observação: Essa é a configuração padrão. Portanto, o arquivo **db_adapter.xml** já contém essa definição.

Há situações nas quais a conexão com o banco de dados não pode ser configurada dessa maneira. Por exemplo, conectar a um Oracle RAC ou conectar usando um driver do banco de dados que não seja o fornecido com o CMDB.

Para essas situações, você pode descrever a conexão usando nome de usuário (esquema), senha e uma cadeia de URL de conexão.

Para definir isso, edite a seção de parâmetros do XML do adaptador como a seguir:

```
<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
database" mandatory="true" order-index="10" />
  <parameter name="credentialsId" display-name="Credentials
ID" type="integer" description="The credentials to be used"
mandatory="true" order-index="12" />
</parameters>
```

Um exemplo de uma URL que conecta a um Oracle RAC usando o driver pronto para o uso Data Direct é:

jdbc:mercury:oracle://labm3amdb17:1521;ServiceName=RACQA;AlternateServers=(labm3amdb18:1521);LoadBalancing=true.

4. No diretório temporário, abra a pasta **adapterCode** e renomeie **GenericDBAdapter** para o valor de **adapter id** que foi usado na etapa anterior.

Essa pasta contém a configuração do adaptador, por exemplo, o nome do adaptador, as consultas e as classes no CMDB, além dos campos no RDBMS que o adaptador aceita.
5. Configure o adaptador conforme necessário. Para obter detalhes, consulte "[Configurar o adaptador - Método mínimo](#)" abaixo.
6. Crie um arquivo *.zip com o mesmo nome dado ao atributo **adapter id**, conforme descrito na etapa "Edite o arquivo XML do adaptador:" na página 91.

Observação: O arquivo **descriptor.xml** é um arquivo padrão que existe em cada pacote.

7. Salve o novo pacote que você criou na etapa anterior. O diretório padrão dos adaptadores é: **C:\hp\UCMDB\UCMDBServer\content\adapters**.

Configurar o adaptador - Método mínimo

O procedimento a seguir descreve um método de mapear o modelo de classe no CMDB para um RDBMS.

Estes arquivos de configuração estão localizados no pacote **db-adapter.zip** da pasta **C:\hp\UCMDB\UCMDBServer\content\adapters** que você extraiu na etapa "[Extraia o pacote para um diretório temporário local.](#)" na página 91 em "[Preparar o pacote de adaptadores](#)" na página 91.

Observação: O arquivo **orm.xml** que é gerado automaticamente como resultado da execução desse método é um ótimo exemplo que pode ser usado ao trabalhar com o método avançado.

Você usará este método mínimo quando precisar:

- Federar/popular um único nó, como um atributo de nó.
- Demonstrar os recursos do adaptador de banco de dados genérico

Este método:

- Suporta somente federação/população de um nó
- Suporta somente relacionamentos virtuais de muitos-para-um

Esta tarefa inclui as seguintes etapas:

- "[Configurar o arquivo adapter.conf](#)" abaixo
- "[Configurar o arquivo simplifiedConfiguration.xml](#)" na página seguinte

Configurar o arquivo adapter.conf

Nesta etapa, você altera as configurações no arquivo `adapter.conf` para que o adaptador use o método de configuração simplificado.

1. Abra o arquivo **adapter.conf** em um editor de texto.
2. Localize a seguinte linha: **use.simplified.xml.config=<true/false>**.

3. Altere-a para `use.simplified.xml.config=true`.

Configurar o arquivo `simplifiedConfiguration.xml`

Nesta etapa, você configura o arquivo `simplifiedConfiguration.xml` mapeando o TEC no CMDB para os campos na tabela RDBMS.

1. Abra o arquivo `simplifiedConfiguration.xml` em um editor de texto.

Esse arquivo inclui um modelo usado para que cada entidade seja mapeada.

Observação: Não edite o arquivo `simplifiedConfiguration.xml` em nenhuma versão do Bloco de Notas da Microsoft Corporation. Use o Notepad++, o UltraEdit ou algum outro editor de texto de terceiros.

2. Faça alterações nos seguintes atributos:

- O nome do TEC no UCMDb (`cmdb-class-name`) e o nome da tabela correspondente no RDBMS (`default-table-name`):

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

O atributo `cmdb-class-name` é obtido no TEC de nó:



O atributo `default-table-name` é obtido na tabela Device:

	Column Name	Data Type	Length	Allow Nulls
1	Device_ID	int		<input type="checkbox"/>
2	Device_Discovered	enum		<input type="checkbox"/>
3	Device_ManagedCategory	enum		<input checked="" type="checkbox"/>
4	Device_PREFERREDMACAddress	varchar	12	<input checked="" type="checkbox"/>
5	Device_PREFERREDIPAddress	varchar	15	<input checked="" type="checkbox"/>
6	Device_LogicalSubNet	varchar	50	<input checked="" type="checkbox"/>
7	Device_Tag	text		<input checked="" type="checkbox"/>
8	Device_Label	varchar	255	<input checked="" type="checkbox"/>
9	DeviceCategory_ID	int		<input checked="" type="checkbox"/>
10	DeviceIcon_ID	int		<input checked="" type="checkbox"/>
11	Device_Description	text		<input checked="" type="checkbox"/>
12	Device_ObjectID	text		<input checked="" type="checkbox"/>
13	Device_Contact	text		<input checked="" type="checkbox"/>
14	Device_Name	text		<input checked="" type="checkbox"/>
15	Device_Location	text		<input checked="" type="checkbox"/>
16	Device_NetBIO5	varchar	255	<input checked="" type="checkbox"/>

- O identificador exclusivo no RDBMS:

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

Observação: A chave principal é equivalente à id de entidade no arquivo orm.xml.

- A regra de reconciliação (reconciliation-by-two-nodes):


```
<reconciliation-by-two-nodes connected-node-cmdb-class-name="ip_address" cmdb-link-type="containment">
```
- O atributo de reconciliação no UCMDb (cmdb-attribute-name) e no RDBMS (column-name):


```
<connected-node-attribute cmdb-attribute-name="name" column-name="[column_name]"/>
```
- O nome do TEC (cmdb-class-name) e o nome da tabela correspondente no RDBMS (default-table-name). Além disso, o relacionamento do UCMDb (connected-cmdb-class-name) e o relacionamento do TEC (link-class-name):


```
<class cmdb-class-name="sw_sub_component" default-table-name="SWSubComponent" connected-cmdb-class-name="node" link-class-name="composition">
```
- A chave primária e a chave estrangeira:


```
<foreign-primary-key column-name="Device_ID" cmdb-class-primary-key-column="Device_ID"/>
```
- O identificador exclusivo no RDBMS:


```
<primary-key column-name="Device_ID"/>
```
- O mapeamento entre o atributo UCMDb (cmdb-attribute-name) e o nome da coluna no RDBMS (column-name):



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

3. Salve o arquivo.

Exemplo: Preenchendo um nó e endereço IP usando o método simplificado

Esse exemplo demonstra o preenchimento de um **Nó** relacionado por um link de contenção para **Endereço IP** no UCMDb. O The RDBMS tem uma tabela chamada **simpleNode** que contém dados no nome do computador, nó do computador e o endereço IP do computador.

O conteúdo da tabela **simpleNode** é mostrado abaixo:

	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

A população é realizada em três estágios, como a seguir:

- ["Criar o arquivo simplifiedConfiguration.xml" abaixo](#)
- [" Crie o TQL" na página seguinte](#)
- ["Criar um ponto de integração" na página 99](#)

Criar o arquivo **simplifiedConfiguration.xml**

Criar o arquivo **simplifiedConfiguration.xml** como indicado a seguir:

1. Criar uma entidade **cmdb-class** como indicado a seguir:

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

O Tipo de EC é **node** e o nome da tabela RDBMS é **simpleNode**.

2. Definir o primary-key da tabela como indicado a seguir:

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

Essa chave principal é equivalente à id de entidade no arquivo **orm.xml**.

3. Defina a regra **reconciliation-by-two-nodes** como a seguir:

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

Essa tag define a relação entre os tipos de EC **Node** e **IpAddress**. O tipo de relação é o link Containment. A reconciliação é executada pelos dois Tipos de EC conectados. O mapeamento de atributos do nó conectado (nesse caso IpAddress) é definido no atributo **connected-node**.

4. Adicione a condição **or** entre os atributos de reconciliação como indicado a seguir:

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

Essa marcação define um relacionamento OR entre os atributos de reconciliação, significando que o primeiro atributo de reconciliação que é **verdadeiro** define toda a regra de reconciliação como **verdadeira**.

5. Adicione os seguintes atributos:

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

Essa marcação define um mapeamento entre **node.name** no UCMDb para a coluna **host_name** na tabela **simpleNode**.

Faça o mesmo com o atributo **data_note**:

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

Adicione o atributo de nó conectado:

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

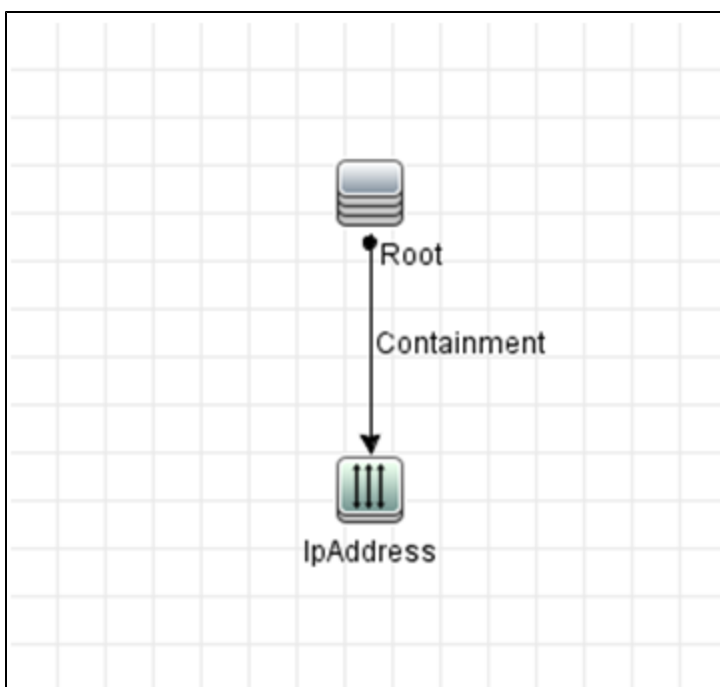
Essa marcação define um mapeamento entre **ip_address.name** para a coluna **ip_address** na tabela **simpleNode**.

6. Feche a marcação aberta pela ordem:

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

Crie o TQL

O TQL será um **nó** conectado por um link de contenção a **ip_address**. O nó deve ser marcado como **root**, conforme mostrado abaixo.



Para criar o TQL:

1. Vá até Modelagem > Modeling Studio.
2. Clique no botão Novo e crie uma nova consulta.
3. Vá até a guia Tipos de EC e arraste o Tipo de EC Node e o Tipo de EC IpAddress para a tela do TQL.
4. Conecte **Node** e **IpAddress** com o link Containment.
5. Clique com o botão direito do mouse no elemento **Node** e escolha Propriedades do Nó de Consulta.
6. Altere o **Nome do elemento** para **Raiz**.
7. Vá até a guia **Layout de Elemento**. Na **Condição do atributo**, selecione **Atributos Específicos**. Escolha **Nome** e **Nota** a partir da janela de Atributos Disponíveis e mova-os para a janela Atributos Específicos.
8. Clique com o botão direito do mouse no elemento **IpAddress** e escolha Propriedades do Nó de Consulta.

9. Vá até a guia **Layout de Elemento**. Na **Condição do atributo**, selecione **Atributos Específicos**. Escolha **Nome** a partir da janela de Atributos Disponíveis e mova-os para a janela Atributos Específicos.
10. Salve o TQL.

Criar um ponto de integração

Crie um ponto de integração como indicado a seguir:

1. Vá até Gerenciamento de Fluxo de Dados > Integration Studio e clique no botão **Novo Ponto de Integração**.
2. Insira os detalhes do Ponto de Integração e clique em OK.
3. Na guia População, selecione o botão **Novo Trabalho de Integração** e adicione o TQL criado anteriormente.
4. Salve o Ponto de Integração e clique no botão Executar Sincronização Completa.

Configurar o adaptador - Método avançado

Estes arquivos de configuração estão localizados no pacote **db-adapter.zip** da pasta **C:\hp\UCMDB\UCMDBServer\content\adapters** que você extraiu na etapa "Extraia o pacote para um diretório temporário local." na página 91 em "Preparar o pacote de adaptadores" na página 91.

Esta tarefa inclui as seguintes etapas:

- "Configurar o arquivo **orm.xml**" abaixo
- "Configurar o arquivo **reconciliation_types.txt**" na página 102
- "Configurar o arquivo **reconciliation_rules.txt**" na página 103

Configurar o arquivo **orm.xml**

Nesta etapa, você mapeia os TECs e relacionamentos no CMDB para as tabelas no RDBMS.

1. Abra o arquivo **orm.xml** em um editor de texto.

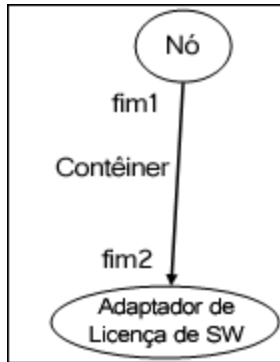
Esse arquivo, por padrão, contém um modelo que é usado para mapear quantos TECs e relacionamentos forem necessários.

Observação: Não edite o arquivo **orm.xml** em nenhuma versão do Bloco de Notas da Microsoft Corporation. Use o Notepad++, o UltraEdit ou algum outro editor de texto de terceiros.

2. Faça alterações no arquivo de acordo com as entidades de dados que serão mapeadas. Para ver detalhes, consulte os exemplos a seguir.

Os seguintes tipos de relacionamentos podem ser mapeados no arquivo **orm.xml**:

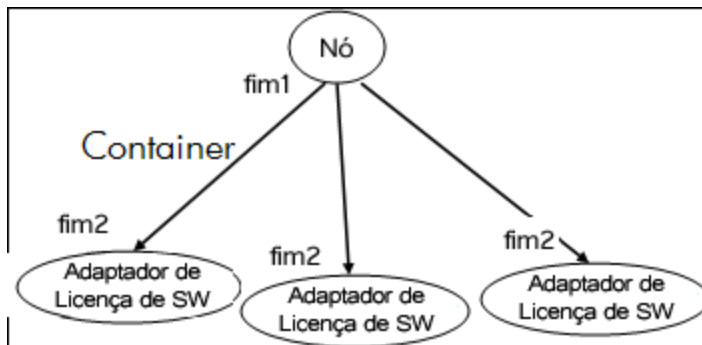
■ Um-para-um:



O código desse tipo de relacionamento é:

```
<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>
```

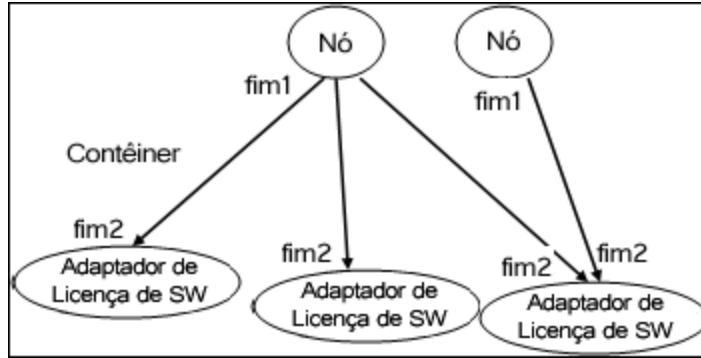
■ Muitos-para-um:



O código desse tipo de relacionamento é:

```
<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>
```

■ Muitos-para-muitos:



O código desse tipo de relacionamento é:

```
<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>
```

Para ver detalhes sobre convenções de nomenclatura, consulte "[Convenções de nomenclatura](#)" na página 123.

Exemplo de mapeamento de entidades entre o modelo de dados e o RDBMS:

Observação: Atributos que não precisam ser configurados são omitidos dos exemplos a seguir.

- A classe do relacionamento do CMDB:

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

- O nome da tabela no RDBMS:

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

- O nome da coluna do identificador exclusivo na tabela RDBMS:

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

- O nome do atributo no TEC do CMDB:

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

- O nome do campo de tabela na fonte de dados externa:

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

- O nome do novo TEC criado no "[Criar um tipo de EC](#)" na página 89:

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

- O nome da tabela correspondente no RDBMS:

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

- A identidade exclusiva no RDBMS:
- O nome do atributo no TEC do CMDB e o nome do atributo correspondente no RDBMS:

Exemplo de mapeamento de relacionamentos entre o modelo de dados e o RDBMS:

- A classe do relacionamento do CMDB:

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

- O nome da tabela RDBMS em que o relacionamento é executado:

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

- O ID exclusivo no 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>
```

- O tipo de relacionamento e o TEC do CMDB:

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

- Os campos de chave primária e chave estrangeira no RDBMS:

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

Configurar o arquivo `reconciliation_types.txt`

Abra o arquivo `reconciliation_types.txt` em um editor de texto.

Para obter detalhes, consulte ["O arquivo `reconciliation_types.txt`" na página 132](#).

Configurar o arquivo `reconciliation_rules.txt`

Nesta etapa, você define as regras pelas quais o adaptador reconcilia o CMDB e o RDBMS (somente se o Mecanismo de Mapeamento for usado, para permitir compatibilidade com a versão 8.x):

1. Abra o arquivo `META-INF\reconciliation_rules.txt` em um editor de texto.
2. Faça alterações no arquivo de acordo com o TEC que você está mapeando. Por exemplo, para mapear um TEC de nó, use a seguinte expressão:

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

Observação:

- Se os dados no banco de dados diferenciarem maiúsculas de minúsculas, não exclua o caractere de controle (^).
- Verifique se cada colchete de abertura tem um colchete de fechamento correspondente.

Para obter detalhes, consulte "[O arquivo `reconciliation_rules.txt` \(para compatibilidade com versões anteriores\)](#)" na página 132.

Implementar um plugin

Esta tarefa descreve como implementar e implantar um adaptador de banco de dados genérico com plugins.

Observação: Antes de escrever um plugin para um adaptador, verifique se concluiu todas as etapas necessárias em "[Preparar o pacote de adaptadores](#)" na página 91.

1. Copie os seguintes arquivos jar do diretório de instalação do servidor UCMDB para o caminho de classe de desenvolvimento:
 - Copie o arquivo `db-interfaces.jar` e o arquivo `db-interfaces-javadoc.jar` da pasta `tools\adapter-dev-kit\db-adapter-framework`.
 - Copie o arquivo `federation-api.jar` e o arquivo `federation-api-javadoc.jar` da pasta `tools\adapter-dev-kit\SampleAdapters\production-lib`.

Observação: Para ver mais informações sobre como desenvolver um plugin, consulte os arquivos `db-interfaces-javadoc.jar` e `federation-api-javadoc.jar` e na documentação online em:

- `C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docsleng\APIs\DBAdapterFramework_JavaAPI\index.html`
- `C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docsleng\APIs\Federation_JavaAPI\index.html`

2. Escreva uma classe Java implementando a interface Java do plugin. As interfaces são

definidas no arquivo **db-interfaces.jar**. A tabela a seguir especifica a interface que precisa ser implementada para cada plugin:

Tipo de plugin	Nome da interface	Método
Sincronizar topologia completa	FcmdbPluginForSyncGetFullTopology	getFullTopology
Sincronizar alterações	FcmdbPluginForSyncGetChangesTopology	getChangesTopology
Sincronizar layout	FcmdbPluginForSyncGetLayout	getLayout
Recuperar consultas aceitas	FcmdbPluginForSyncGetSupportedQueries	getSupportedQueries
Alterar definições e resultados de consulta TQL	FcmdbPluginGetTopologyCmdbFormat	getTopologyCmdbFormat
Alterar solicitação de layout para ECs	FcmdbPluginGetCIsLayout	getCisLayout
Alterar solicitação de layout para links	FcmdbPluginGetRelationsLayout	getRelationsLayout
Devolver IDs	FcmdbPluginPushBackIds	getPushBackIdsSQL

A classe do plugin precisa ter um construtor padrão público. Além disso, todas as interfaces expõem um método denominado `initPlugin`. Esse método tem a garantia de ser chamado antes de qualquer outro método e é usado para inicializar o adaptador com o objeto de ambiente do adaptador recipiente.

Se **FcmdbPluginForSyncGetChangesTopology** for implementado, há dois modos diferentes de reportar as mudanças:

- Reporte toda a topologia raiz todas as vezes.** De acordo com essa topologia, a função de exclusão automática descobre quais ECs devem ser removidos. Nesse caso, a função de exclusão automática deve ser habilitada usando o seguinte:

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

- Reporte cada instância de EC removida/atualizada.** Nesse caso, o mecanismo de exclusão automática deve ser habilitado usando o seguinte:

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



```
</autoDeleteCITs>
```

3. Verifique se tem o JAR do Federation SDK e os JARs do adaptador de banco de dados genérico em seu caminho de classe antes de compilar seu código Java. O Federation SDK é o arquivo **federation_api.jar**, que pode ser encontrado no diretório **C:\hp\UCMDB\UCMDBServer\lib**.
4. Compacte sua classe em um arquivo jar e na pasta `adapterCode\<Seu_Nome_de_Adaptador>` do pacote de adaptadores, antes de implantá-lo.

Os plugins são configurados usando o arquivo **plugins.txt**, localizado na pasta **\META-INF** do adaptador.

Veja a seguir um exemplo do arquivo do adaptador 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 [getTopologyCmdFormat] #
internal not mandatory plugin to change layout request and CIs
result [getCisLayout] # internal not mandatory plugin to change
layout request and relations result [getRelationsLayout] # internal
not mandatory plugin to change action on pushBackIds [pushBackIds]
```

Legenda:



- Uma linha de comentário.

[<Tipo de Adaptador>] – Inicie a seção de definição de um tipo de adaptador específico.

Poderá haver uma linha vazia sob cada [<Tipo de Adaptador>], significando que não há uma classe de plugin associada ou poderá estar listado o nome totalmente qualificado de sua classe de plugin.

5. Compacte o adaptador com o novo arquivo jar e o arquivo **plugins.xml** atualizado. O restante dos arquivos no pacote deverá ser o mesmo de qualquer adaptador baseado no adaptador de banco de dados genérico.

Implantar o adaptador

1. No UCMDB, acesse o Gerenciador de Pacotes. Para obter detalhes, consulte "[Página Gerenciador de Pacotes](#)" na página 1 no *Guia de Administração do HP Universal CMDB*.
2. Clique no ícone **Implantar pacotes no servidor (a partir do disco local)**  e navegue até o pacote de adaptadores. Selecione o pacote e clique em **Abrir**, clique em **Implantar** para exibir o pacote no Gerenciador de Pacotes.
3. Selecione o pacote na lista e clique no ícone **Exibir recursos do pacote**  para verificar se o conteúdo do pacote é reconhecido pelo Gerenciador de Pacotes.

Editar o adaptador

Depois de criar e implantar o adaptador, você poderá editá-lo no UCMDB. Para obter detalhes, consulte "[Configuração do Adaptador](#)" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Criar um ponto de integração

Nesta etapa, você verifica se a federação está funcionando. Ou seja, se a conexão e o arquivo XML são válidos. Entretanto, essa verificação não confere se o XML está mapeando para os campos corretos no RDBMS.

1. No UCMDB, acesse o Integration Studio (**Gerenciamento de Fluxo de Dados > Integration Studio**).
2. Crie um ponto de integração. Para obter detalhes, consulte "[Caixa de diálogo Novo Ponto de Integração/Editar Ponto de Integração](#)" na [página 1](#) no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

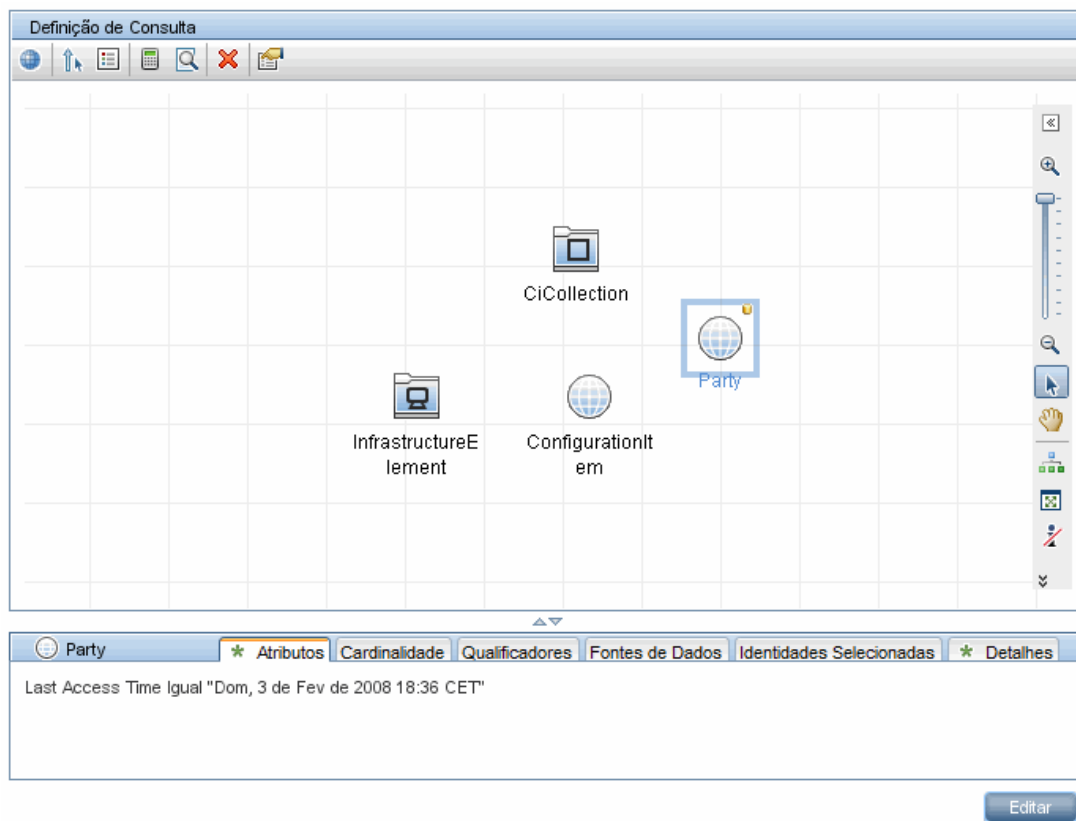
A guia Federação exibe todos os TECs que podem ser federados usando esse ponto de integração. Para obter detalhes, consulte "[Guia Federação](#)" na [página 1](#) no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Criar uma visualização

Nesta etapa, você cria uma visualização que permite ver instâncias do TEC.

1. No UCMDB, acesse o Modeling Studio (**Modelagem > Modeling Studio**).
2. Crie uma visualização. Para obter detalhes, consulte "[Criar uma visualização do padrão](#)" na [página 1](#) no *Guia de Modelagem do HP Universal CMDB*.
3. Você pode adicionar condições ao TQL, por exemplo, o último horário de acesso é superior a

seis meses:



Calcular os resultados

Nesta etapa, você verifica os resultados.

1. No UCMDB, acesse o Modeling Studio (**Modelagem > Modeling Studio**).
2. Abra uma visualização.
3. Calcule os resultados clicando no botão **Calcular Contagem de Resultados de Consulta**.



4. Clique no botão **Visualização** para exibir os ECs na visualização.

Visualizar os resultados

Nesta etapa, você exibe os resultados e depura os problemas no procedimento. Por exemplo, se nada for mostrado na visualização, verifique as definições no arquivo **orm.xml**; remova os atributos de relacionamento e recarregue o adaptador.

1. No UCMDB, acesse o Gerenciador de Universo de IT (**Modelagem > Gerenciador de Universo de TI**).
2. Selecione um EC. A guia Propriedades exibe os resultados da federação.

Visualizar relatórios

Nesta etapa, você exibe os relatórios de Topologia. Para obter detalhes, consulte "Visão geral de Relatórios de Topologia" na página 1 no *Guia de Modelagem do HP Universal CMDB*.

Habilitar arquivos de log

Para entender os fluxos de cálculo e o ciclo de vida do adaptador, bem como visualizar informações de depuração, você pode consultar os arquivos de log. Para obter detalhes, consulte "Arquivos de log do adaptador" na página 150.

Usar o Eclipse para mapear entre atributos de TEC e tabelas de banco de dados

Cuidado: Este procedimento é voltado aos usuários com conhecimento avançado em desenvolvimento de conteúdo. Em caso de dúvidas, entre em contato com Suporte da HP Software.

Esta tarefa descreve como instalar e usar o plugin JPA, fornecido com a edição J2EE do Eclipse, para:

- Habilitar o mapeamento gráfico entre os atributos de classe do CMDB e as colunas de tabela de banco de dados.
- Habilitar a edição manual do arquivo de mapeamento (`orm.xml`), ao mesmo tempo garantindo exatidão. A verificação de exatidão inclui uma verificação de sintaxe, além da verificação de se os atributos de classe e as colunas de tabela de banco de dados mapeadas estão declaradas corretamente.
- Habilitar a implantação do arquivo de mapeamento no servidor CMDB e para visualizar os erros, como uma verificação de exatidão adicional.
- Definir uma consulta de exemplo no servidor CMDB e executá-la diretamente do Eclipse, para testar o arquivo de mapeamento.

A versão 1.1 do plugin é compatível com a versão 9.01 do UCMDb ou posterior e Eclipse IDE for Java EE Developers, versão 1.2.2.20100217-2310 ou posterior.

Esta tarefa inclui as seguintes etapas:

- "Pré-requisitos" na página seguinte
- "Instalação" na página seguinte
- "Preparar o ambiente de trabalho" na página seguinte
- "Criar um adaptador" na página 110
- "Configure o plugin do CMDB" na página 110
- "Importar o modelo de classe do UCMDb" na página 111

- "Criar o arquivo ORM – mapear classes do UCMDB para tabelas de banco de dados" na página 111
- "IDs de Mapa" na página 111
- "Atributos de mapa" na página 112
- "Mapear um link válido" na página 112
- "Criar o arquivo ORM – usar tabelas secundárias" na página 113
- "Definir uma tabela secundária" na página 113
- "Mapear um atributo para uma tabela secundária" na página 113
- "Usar um arquivo ORM existente como base" na página 113
- "Importando um arquivo ORM existente de um adaptador" na página 114
- "Verificar a exatidão do arquivo orm.xml – verificação de exatidão interna" na página 114
- "Criar um novo ponto de integração" na página 114
- "Implantar o arquivo ORM no CMDB" na página 115
- "Executar uma consulta TQL de exemplo" na página 115

1. Pré-requisitos

Instalar a atualização mais recente do **Java Runtime Environment (JRE) 6** na máquina em que você executará o Eclipse a partir do seguinte site:
<http://java.sun.com/javase/downloads/index.jsp>.

2. Instalação

- a. Baixe e extraia **Eclipse IDE for Java EE Developers** em <http://www.eclipse.org/downloads> para uma pasta local, por exemplo, **C:\Arquivos de Programas\eclipse**.
- b. Copie **com.hp.plugin.import_cmdb_model_1.0.jar** de **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\bin** para **C:\Arquivos de Programas\Eclipse\plugins**.
- c. Inicie **C:\Program Files\Eclipse\eclipse.exe**. Se uma mensagem for exibida informando que a máquina virtual Java não foi encontrada, inicie o **eclipse.exe** com a seguinte linha de comando:

```
"C:\Arquivos de Programas\eclipse\eclipse.exe" -vm "<pasta de  
instalação JRE>\bin"
```

3. Preparar o ambiente de trabalho

Nesta etapa, você vai configurar o espaço de trabalho, o banco de dados, as conexões e as propriedades de driver.

- a. Extraia o arquivo **workspaces_gdb.zip** de **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** into **C:\Documents and Settings\All Users**.

Observação: Você precisa usar o caminho de pastas exato. Se descompactar o arquivo para o caminho errado ou deixar o arquivo descompactado, o procedimento não funcionará.

- b. No Eclipse, escolha **File (Arquivo) > Switch Workspace (Trocar espaço de trabalho) > Other (Outro)**:
Se você estiver trabalhando com:
 - o SQL Server, selecione a seguinte pasta: **C:\Documents and Settings\All Users\workspace_gdb_sqlserver**.
 - o MySQL, selecione a seguinte pasta: **C:\Documents and Settings\All Users\workspace_gdb_mysql**.
 - o Oracle, selecione a seguinte pasta: **C:\Documents and Settings\All Users\workspace_gdb_oracle**.
- c. Clique em **OK**.
- d. No Eclipse, exiba a visualização Project Explorer e selecione **<Projeto ativo> > JPA Content > persistence.xml > <nome do projeto ativo> > orm.xml**.
- e. Na visualização Data Source Explorer (painel inferior esquerdo), clique com o botão direito do mouse na conexão de banco de dados e selecione o menu **Properties (Propriedades)**.
- f. Na caixa de diálogo **Properties for (Propriedades para) <Nome da conexão>**, selecione **Common (Comum)** e marque a caixa de seleção **Connect every time the workbench is started (Conectar sempre que o workbench for iniciado)**. Selecione **Driver Properties (Propriedades do driver)** e preencha as propriedades de conexão. Clique em **Test Connection (Testar conexão)** e verifique se a conexão está funcionando. Clique em **OK**.
- g. Na visualização Data Source Explorer, clique com o botão direito do mouse na conexão de banco de dados e clique em **Connect (Conectar)**. Um árvore que contém os esquemas e tabelas de banco de dados é exibido no ícone de conexão de banco de dados.

4. Criar um adaptador

Crie um adaptador usando as diretrizes em ["Etapa 1: Criar um adaptador"](#) na página 27.

5. Configure o plugin do CMDB

- a. No Eclipse, clique em **UCMDB > Settings (Configurações)** para abrir a caixa de diálogo **CMDB Settings (Configurações do CMDB)**.
- b. Se ainda não estiver selecionado, selecione o projeto JPA recém-criado como projeto ativo.
- c. Insira o nome de host do CMDB, por exemplo, **localhost** ou **labm1.itdep1**. Não é necessário incluir o número da porta ou o prefixo `http://` no endereço.
- d. Preencha o nome de usuário e senha para acessar a API do CMDB, normalmente **admin/admin**.
- e. Verifique se a pasta **C:\hp** no servidor CMDB está mapeada como uma unidade de rede.

- f. Selecione a pasta base do adaptador relevante em **C:\hp**. A pasta base é aquela que contém o arquivo **dbAdapter.jar** e a subpasta **META-INF**. Seu caminho deve ser **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\<nome do adaptador>**.
. Verifique se não há barra invertida (\) no final.

6. Importar o modelo de classe do UCMDB

Nesta etapa, você seleciona os TECs que serão mapeados como entradas JPA.

- a. Clique em **UCMDB > Import CMDDB Class Model (Importar Modelo de Classe do CMDDB)** para abrir a caixa de diálogo **CI Type Selection (Seleção de Tipo de EC)**.
- b. Selecione os tipos de EC que você pretende mapear como entidades JPA. Clique em **OK**. Os tipos de EC são importados como classes Java. Verifique se eles aparecem sob a pasta **src** do projeto ativo.

7. Criar o arquivo ORM – mapear classes do UCMDB para tabelas de banco de dados

Nesta etapa, você mapeia as classes Java (importadas na etapa anterior) para as tabelas de banco de dados.

- a. Verifica se há conexão do banco de dados. Clique com o botão direito do mouse no projeto ativo (denominado myProject por padrão) no Project Explorer. Selecione a visualização JPA, marque a caixa de seleção **Override default schema from connection (Substituir esquema padrão da conexão)** e selecione o esquema de banco de dados relevante. Clique em **OK**.
- b. Mapeie um TEC: Na visualização Estrutura de JPA, clique com o botão direito do mouse na ramificação **Entity Mappings (Mapeamentos de Entidade)** e selecione **Add Class (Adicionar Classe)**. A caixa de diálogo **Add Persistent Class (Adicionar Classe Persistente)** será aberta. Não altere o campo **Map as (Mapear como) (Entity)**.
- c. Clique em **Procurar** e selecione a classe do UCMDB que será mapeada (todas as classes do UCMDB pertencem ao pacote **generic_db_adapter**).
- d. Clique em **OK** em ambas as caixas de diálogo. A classe selecionada é exibida na ramificação **Entity Mappings (Mapeamentos de Entidade)** na visualização JPA Structure (Estrutura de JPA).

Observação: Se a entidade aparecer sem uma árvore de atributo, clique com o botão direito do mouse no projeto ativo na visualização Project Explorer. Escolha **Fechar e Abrir**.

- e. Na visualização Detalhes de JPA, selecione a tabela de banco de dados primária para a qual a classe do UCMDB deve ser mapeada. Deixe todos os outros campos inalterados.

8. IDs de Mapa

De acordo com os padrões JPA, cada classe persistente precisa ter pelo menos um atributo de ID. Para as classes do UCMDB, você pode mapear até três atributos como IDs. Os possíveis atributos de ID são denominados **id1**, **id2** e **id3**.

Para mapear um atributo de ID:

- a. Expanda a classe correspondente na ramificação **Mapeamentos de Entidade** na visualização Estrutura de JPA, clique com o botão direito do mouse no atributo relevante (por exemplo, **id1**) e selecione **Adicionar atributo como XML e mapear...**
- b. A caixa de diálogo **Add Persistent Attribute (Adicionar Atributo Persistente)** será aberta. Selecione **Id** no campo **Map as (Mapear como)** e clique em **OK**.
- c. Na visualização Detalhes de JPA, selecione a coluna de tabela de banco de dados para a qual o campo de ID deve ser mapeado.

9. Atributos de mapa

Nesta etapa, você mapeia os atributos para as colunas de banco de dados.

- a. Expanda a classe correspondente na ramificação **Mapeamentos de Entidade** na visualização Estrutura de JPA, clique com o botão direito do mouse no atributo relevante (por exemplo, **host_nomedohost**) e selecione **Adicionar atributo como XML e mapear...**
- b. A caixa de diálogo **Add Persistent Attribute (Adicionar Atributo Persistente)** será aberta. Selecione **Basic (Básico)** no campo **Map as (Mapear como)** e clique em **OK**.
- c. Na visualização JPA Details (Detalhes de JPA), selecione a coluna de banco de dados para a qual o campo de atributo deve ser mapeado.

10. Mapear um link válido

Execute estas fases descritas acima na etapa "[Criar o arquivo ORM – mapear classes do UCMDB para tabelas de banco de dados](#)" na página anterior para mapear uma classe do UCMDB que indica um link válido. O nome de cada uma dessas classes tem a seguinte estrutura: **<end1 entity name>_<link name>_<end 2 entity name>**. Por exemplo, um link **Contains** entre um host e um local é indicado por uma classe Java cujo nome é **generic_db_adapter.host_contains_location**. Para obter detalhes, consulte "[O arquivo reconciliation_rules.txt \(para compatibilidade com versões anteriores\)](#)" na página 132.

- a. Mapeie os atributos de ID da classe de link conforme descrito em "[IDs de Mapa](#)" na página anterior. Para cada atributo de ID, expanda o grupo de caixas de seleção **Details (Detalhes)** na visualização Detalhes de JPA e desmarque as caixas de seleção **Insertable (Inserível)** e **Updateabl (Atualizável)**.
- b. Mapeie os atributos **end1** and **end2** da classe de link da seguinte maneira: Para cada um dos atributos **end1** and **end2** da classe de link:
 - o. Expanda a classe correspondente na ramificação **Entity Mappings (Mapeamentos de Entidade)** na visualização JPA Structure, clique com o botão direito do mouse no atributo relevante (por exemplo, **end1**) e selecione **Add Attribute to XML and Map... (Adicionar atributo como XML e mapear...)**.
 - o. Na caixa de diálogo **Add Persistent Attribute (Adicionar Atributo Persistente)**, selecione **Many to One (Muitos-para-Um)** ou **One to One (Um-para-Um)** no campo **Map as (Mapear como)**.
 - o. Selecione **Many to One** se o EC **end1** ou **end2** especificado puder ter vários links desse tipo. Caso contrário, selecione **One to One**. Por exemplo, para um link **host_contains_ip** a extremidade de **host** deve ser mapeada como **Many to One (Muitos-para-Um)**, porque um host pode ter vários IPs, e a extremidade **ip** deveria ser mapeada

como **One to One (Um-para-Um)**, porque um IP pode ter somente um host.

- Na visualização JPA Details (Detalhes de JPA), selecione **Target entity (Entidade de destino)**, por exemplo, **generic_db_adapter.host**.
- Na seção **Join Columns (Colunas de Junção)** da visualização Detalhes de JPA, marque **Override Default (Substituir Padrão)**. Clique em **Edit (Editar)**. Na caixa de diálogo **Edit Join Column (Editar Coluna de Junção)**, selecione a coluna de chave estrangeira da tabela de banco de dados do link que aponta para uma entrada na tabela da entidade de destino **end1/end2**. Se o nome da coluna referenciado na tabela da entidade de destino **end1/end2** for mapeado para seu atributo de ID, deixe o **Referenced Column Name (Nome de Coluna Referenciado)** inalterado. Caso contrário, selecione o nome da coluna para a qual a coluna de chave estrangeira aponta. Desmarque as caixas de seleção **Insertable (Inserível)** e **Updatable (Atualizável)** e clique em **OK**.
- Se a entidade de destino **end1/end2** tiver mais de uma ID, clique no botão **Add (Adicionar)** para adicionar colunas de junção e mapeá-las da mesma maneira descrita na etapa anterior.

11. Criar o arquivo ORM – usar tabelas secundárias

O JPA permite que uma classe Java seja mapeada para mais de uma tabela de banco de dados. Por exemplo, **Host** pode ser mapeado para a tabela **Device** para habilitar a persistência da maioria de seus atributos e para a tabela **NetworkNames** para habilitar a persistência de **host_hostName**. Nesse caso, **Device** é a tabela primária e **NetworkNames** é a tabela secundária. Qualquer número de tabelas secundárias pode ser definido. A única condição é que haja um relacionamento de um-para-um entre as entradas das tabelas primárias e secundárias.

12. Definir uma tabela secundária

Selecione a classe apropriada na visualização Estrutura de JPA. Na visualização **JPA Details (Detalhes de JPA)**, acesse a seção **Secondary Tables (Tabelas Secundárias)** e clique em **Add (Adicionar)**. Na caixa de diálogo **Add Secondary Table (Adicionar Tabela Secundária)**, selecione a tabela secundária apropriada. Deixe os outros campos inalterados.

Se a tabela primária e a secundária não tiverem as mesmas chaves primárias, configure as colunas de junção na seção **Primary Key Join Columns (Colunas de Junção de Chave Primária)** da visualização **JPA Details (Detalhes de JPA)**.

13. Mapear um atributo para uma tabela secundária

Você mapeia um atributo de classe para um campo de uma tabela secundária da seguinte maneira:

- a. Mapeie o atributo conforme descrito em "[Atributos de mapa](#)" na página anterior.
- b. Na seção **Column** da visualização JPA Details, selecione o nome da tabela secundária no campo **Table** para substituir o valor padrão.

14. Usar um arquivo ORM existente como base

Para usar um arquivo **orm.xml** existente como base para aquele que você está desenvolvendo, execute as seguintes etapas:

- a. Verifique se todos os TECs mapeados no arquivo **orm.xml** existente são importados para o projeto do Eclipse ativo.
- b. Selecione e copie total ou parcialmente os mapeamentos de entidade do arquivo existente.
- c. Selecione a guia **Source (Fonte)** do arquivo **orm.xml** na perspectiva JPA do Eclipse.
- d. Cole todos os mapeamentos de entidades copiados abaixo da marca **<entity-mappings>** do arquivo **orm.xml** editado, abaixo da marca **<schema>**. Verifique se a marca do esquema está configurada conforme descrito na etapa "[Criar o arquivo ORM – mapear classes do UCMDDB para tabelas de banco de dados](#)" na página 111. Todas as entidades coladas agora aparecem na visualização Estrutura de JPA. De agora em diante, os mapeamentos poderão ser editados tanto gráfica quanto manualmente através do código XML do arquivo **orm.xml**.
- e. Clique em **Salvar**.

15. Importando um arquivo ORM existente de um adaptador

Se já existir um adaptador, o Plugin do Eclipse poderá ser usado para editar seu arquivo ORM graficamente. Importe o arquivo **orm.xml** para o Eclipse, edite-o usando o plugin e implante-o de volta na máquina do UCMDDB. Para importar o arquivo ORM, pressione o botão na barra de ferramentas do Eclipse. Uma caixa de diálogo de confirmação será exibida. Clique em **OK**. O arquivo ORM é copiado da máquina do UCMDDB para o projeto do Eclipse ativo e todas as classes relevantes são importadas do modelo de classe do UCMDDB.

Se as classes relevantes não aparecerem na visualização Estrutura de JPA, clique com o botão direito do mouse no projeto ativo na visualização Project Explorer, escolha **Close (Fechar)** e **Open (Abrir)**.

De agora em diante, o arquivo ORM pode ser editado graficamente usando o Eclipse e implantado de volta na máquina do UCMDDB conforme descrito em "[Implantar o arquivo ORM no CMDB](#)" na página seguinte.

16. Verificar a exatidão do arquivo orm.xml – verificação de exatidão interna

O plugin de JPA do Eclipse verifica se há erros e marca-os no arquivo **orm.xml**. Tanto erros de sintaxe (por exemplo, nome de marca errado, marca não-fechada, ID ausente) quanto de mapeamento (por exemplo, nome de atributo ou nome de campo de tabela de banco de dados errado) são verificados. Se houver erros, sua descrição aparecerá na visualização **Problemas**.

17. Criar um novo ponto de integração

Se não houver nenhum ponto de integração no CMDB para esse adaptador, você poderá criá-lo no Integration Studio. Para obter detalhes, consulte "[Integration Studio](#)" na página 1 no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Preenche o nome do ponto de integração na caixa de diálogo que for aberta. O arquivo **orm.xml** será copiado para a pasta do adaptador. Um ponto de integração será criado com todos os tipos de EC importados como sendo suas classes aceitas, com exceção de TECs de multinós, se eles estiverem configurados no arquivo **reconciliation_rules.txt**. Para obter detalhes, consulte "[O arquivo reconciliation_rules.txt \(para compatibilidade com versões anteriores\)](#)" na página 132.

18. Implantar o arquivo ORM no CMDB

Salve o arquivo **orm.xml** e implante-o no servidor UCMDB clicando em **UCMDB > Implantar ORM**. O arquivo **orm.xml** será copiado para a pasta do adaptador, e o adaptador será recarregado. O resultado da operação é mostrado em uma caixa de diálogo **Operation Result (Resultado da Operação)**. Se algum erro ocorrer durante o processo de recarregamento, o rastreamento de pilha da exceção Java será exibido na caixa de diálogo. Se nenhum ponto de integração ainda não tiver sido definido usando o adaptador, nenhum erro de mapeamento será detectado após a implantação.

19. Executar uma consulta TQL de exemplo

- a. Definir uma consulta (não uma visualização) no Modeling Studio Para obter detalhes, consulte "[Modeling Studio](#)" na página 1 no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.
- b. Crie um ponto de integração usando o adaptador criado na etapa "[Criar um novo ponto de integração](#)" na página anterior. Para obter detalhes, consulte "[Caixa de diálogo Novo Ponto de Integração/Editar Ponto de Integração](#)" na página 1 no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.
- c. Durante a criação do adaptador, verifique se os tipos de EC que devem participar da consulta são aceitos por esse ponto de integração.
- d. Ao configurar o plugin do CMDB, use este nome de consulta de exemplo na caixa de diálogo Settings (Configurações). Para obter detalhes, consulte a etapa acima "[Configure o plugin do CMDB](#)" na página 110.
- e. Clique no botão **Run TWL (Executar TWL)** para executar um TQL de exemplo e verificar se ele retorna os resultados necessários usando o arquivo **orm.xml** recém-criado.

Arquivos de configuração do adaptador

Os arquivos tratados nesta seção estão localizados no pacote **db-adapter.zip** da pasta **C:\hp\UCMDB\UCMDBServer\content\adapters**

Esta seção descreve os seguintes arquivos de configuração:

- "O arquivo **adapter.conf**" na página seguinte
- "O arquivo **simplifiedConfiguration.xml**" na página 117
- "O arquivo **orm.xml**" na página 119
- "O arquivo **reconciliation_types.txt**" na página 132
- "O arquivo **reconciliation_rules.txt** (para compatibilidade com versões anteriores)" na página 132
- "O arquivo **transformations.txt**" na página 134
- "O arquivo **discriminator.properties**" na página 134
- "O arquivo **replication_config.txt**" na página 136
- "O arquivo **fixed_values.txt**" na página 136
- "O arquivo **persistence.xml**" na página 136

Configuração geral

- **adapter.conf.** O arquivo de configuração do adaptador. Para obter detalhes, consulte "[O arquivo adapter.conf](#)" abaixo.

Configuração simples

- **simplifiedConfiguration.xml.** Arquivo de configuração que substitui os arquivos **orm.xml**, **transformations.txt** e **reconciliation_rules.txt** com menos recursos. Para obter detalhes, consulte "[O arquivo simplifiedConfiguration.xml](#)" na página seguinte.

Configuração avançada

- **orm.xml.** O arquivo de mapeamento relacional a objeto em que você mapeia entre os TECs e as tabelas de banco de dados do CMDDB. Para obter detalhes, consulte "[O arquivo orm.xml](#)" na página 119.
- **reconciliation_types.txt.** Contém as regras usadas para configurar os tipos de reconciliação. Para obter detalhes, consulte "[O arquivo reconciliation_types.txt](#)" na página 132.
- **reconciliation_rules.txt.** Contém as regras de reconciliação. Para obter detalhes, consulte "[O arquivo reconciliation_rules.txt \(para compatibilidade com versões anteriores\)](#)" na página 132.
- **transformations.txt.** Arquivo de transformações em que você especifica os conversores que serão aplicados para converter do valor do CMDDB para o valor do banco de dados e vice-versa. Para obter detalhes, consulte "[O arquivo transformations.txt](#)" na página 134.
- **Discriminator.properties.** Este arquivo mapeia cada tipo de EC aceito para uma lista separada por vírgulas de possíveis valores correspondentes. Para obter detalhes, consulte "[O arquivo discriminator.properties](#)" na página 134.
- **Replication_config.txt.** Este arquivo contém uma lista separada por vírgulas de tipos de EC e de relacionamento cujas condições de propriedade são aceitas pelo plugin de replicação. Para obter detalhes, consulte "[O arquivo replication_config.txt](#)" na página 136.
- **Fixed_values.txt.** Este arquivo permite configurar os valores fixos para atributos específicos de determinados TECs. Para obter detalhes, consulte "[O arquivo fixed_values.txt](#)" na página 136.

Configuração do Hibernate

- **persistence.xml.** Usado para substituir configurações prontas do Hibernate. Para obter detalhes, consulte "[O arquivo persistence.xml](#)" na página 136.

O arquivo adapter.conf

Este arquivo contém as seguintes configurações:

- **use.simplified.xml.config=false.true:** uses `simplifiedConfiguration.xml`.

Observação: O uso deste arquivo significa que os arquivos `orm.xml`, `transformations.txt` e `reconciliation_rules.txt` são substituídos por menos recursos.

- **dal.ids.chunk.size=300.** Não altere esse valor.
- **dal.use.persistence.xml=false.true:** o adaptador lê a configuração do Hibernate de

```
persistence.xml.
```

Observação: Não é recomendável substituir a configuração do Hibernate.

- **performance.memory.id.filtering=true.** Quando o GDBA executa TQLS, em alguns casos, um grande número de IDs pode ser recuperado e enviado de volta ao banco de dados usando o SQL. Para evitar esse trabalho excessivo e melhorar o desempenho, o GDBA tenta ler toda a visualização/tabela e filtra os resultados na memória.
- **id.reconciliation.cmdb.id.type=string/bytes.** Ao mapear o adaptador do banco de dados genérico usando a Reconciliação de ID (para informações, consulte a etapa "[Configure o arquivo reconciliation_types.txt](#) (para o mecanismo de mapeamento padrão do UCMDB 9.0x)" em "[Implementar o mecanismo de mapeamento](#)" na página 179, você pode mapear **cmdb_id** para uma **cadeia** ou tipo de coluna **bytes/raw** alterando a propriedade **META-INF/adapter.conf**.
- **performance.enable.single.sql=true.** Esse é um parâmetro opcional. Se ele não for exibido no arquivo, seu valor padrão é **verdadeiro**. Quando **verdadeiro**, o adaptador de banco de dados genérico tenta gerar uma única instrução SQL para cada consulta que está sendo executada (para população ou uma consulta federada). Usar uma única instrução SQL melhora o desempenho e o consumo de memória do adaptador de banco de dados genérico. Quando **falso**, o adaptador de banco de dados genérico gera várias instruções SQL, que podem levar mais tempo e consumir mais memória que uma única. Mesmo quando esse atributo é definido como **verdadeiro**, o adaptador não gera uma única instrução SQL nos seguintes cenários:
 - O banco de dados ao qual o adaptador se conecta não está em um servidor Oracle ou SQL.
 - O TQL em execução contém uma condição de cardinalidade diferente de 0..* e 1..* (por exemplo, se houver uma condição de cardinalidade como 2..* ou 0..2).
- **in.expression.size.limit=950** (padrão). Esse parâmetro divide a expressão 'IN' do SQL executado, quando o limite de tamanho da lista de argumentos é atingido.

O arquivo `simplifiedConfiguration.xml`

Este arquivo é usado para mapeamento simples de classes do UCMDB para tabelas de banco de dados. Para acessar o modelo para editar o arquivo, vá até **Gerenciamento do Adaptador > db-adapter > arquivos de configuração**.

Esta seção inclui os seguintes tópicos:

- "[O modelo de arquivo `simplifiedConfiguration.xml`](#)" abaixo
- "[Limitações](#)" na página 119

O modelo de arquivo `simplifiedConfiguration.xml`

A propriedade **CMDB-class-name** é de tipo de vários nós (o nó ao qual os TECs federados se conectam no TQL):

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-
```

```
CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="node" default-table-name="[table_
name]">
    <primary-key column-name="[column_name]" />
```

reconciliation-by-two-nodes. A reconciliação pode ser executada usando um ou dois nós. Neste exemplo de caso, a reconciliação usa dois nós.

connected-node-CMDB-class-name. O segundo tipo de classe necessário no TQL de reconciliação.

CMDB-link-type. O tipo de relacionamento necessário no TQL de reconciliação.

link-direction. A direção do relacionamento no TQL de reconciliação (de `node` para `ip_address` ou de `ip_address` para `node`):

```
<reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment" link-direction="main-
to-connected">
```

A expressão de reconciliação está na forma de ORs e cada OR inclui ANDs.

is-ordered. Determina se a reconciliação é executada na forma de ordem ou por uma comparação OR regular.

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

Se a propriedade de reconciliação for recuperada da classe principal (o multinó), use a marca **attribute**, caso contrário, use a marca **connected-node-attribute**.

ignore-case.true: quando os dados no modelo de classe UCMDB são comparados com os dados no RDBMS, o caso não importa:

```
<attribute CMDB-attribute-name="name" column-name="[
column_name]" ignore-case="true"/>
```

O nome da coluna é o nome da coluna de chave estrangeira (a coluna com valores que apontam para a coluna de chave primária de multinó).

Se a coluna de chave primária de multinó for composta de várias colunas, será necessário haver diversas colunas de chave estrangeira, sendo uma para cada coluna de chave primária.

```
<foreign-primary-key column-name="[column_name]" CMDB-class-
primary-key-column="[column_name]"/>
```

Se houver poucas colunas de chave primária, duplique essa coluna.

```
<primary-key column-name="[column_name]"/>
```

As propriedades **from-CMDB-converter** e **to-CMDB-converter** são classes Java que implementam as seguintes interfaces:

- `com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.FcldbDalTransformerFromExternalDB`
- `com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.FcldbDalTransformerToExternalDB`

Use esses conversores se o valor no CMDB e no banco de dados não for o mesmo.

Neste exemplo, `GenericEnumTransformer` é usado para converter o enumerador de acordo com o arquivo XML escrito entre parênteses (**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.im-
pl.
GenericEnumTransformer(generic-enum-transformer-example.xml)" to-CMDB-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.im-
pl.
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>
```

Limitações

- Pode ser usado para mapear somente consultas TQL com um nó (na fonte do banco de dados). Por exemplo, você pode executar um `node > ticket` e uma consulta TQL `ticket`. Para importar a hierarquia de nós do banco de dados, você precisa usar o arquivo **orm.xml** avançado.
- Somente relacionamentos um-para-muitos são aceitos. Por exemplo, você pode importar um ou mais tickets para cada nó. Você não pode importar tickets que pertencem a mais de um nó.
- Não pode conectar a mesma classe a diferentes tipos de TECs do CMDB. Por exemplo, se você definir que `ticket` está conectado a `node`, ele não poderá ser conectado a `application` também.

O arquivo orm.xml

Este arquivo é usado para mapeamento dos TECs do CMDB para tabelas de banco de dados.

Um modelo para ser usado na criação de um novo arquivo está localizado no diretório **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INF**.

Para editar o arquivo XML para um adaptador implantado, navegue para **Gerenciamento do Adaptador > db-adapter > arquivos de configuração**.

Esta seção inclui os seguintes tópicos:

- " O modelo de arquivo orm.xml" na página seguinte
- " Vários arquivos ORM" na página 123
- " Convenções de nomenclatura" na página 123
- " Usando instruções SQL em linha em vez de nomes de tabela" na página 123
- " O esquema orm.xml" na página 124
- " Exemplo de criação do arquivo orm.xml" na página 128

O modelo de arquivo orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd">
  <description>Generic DB adapter orm</description>
```

Não altere o nome do pacote.

```
<package>generic_db_adapter</package>
```

entity. O nome do TEC do CMDB. Esta é a entidade de vários nós.

Verifique se **class** inclui um prefixo **generic_db_adapter..**

```
<entity class="generic_db_adapter.node">
  <table name="[table_name]" />
```

Use uma tabela secundária se a entidade for mapeada para mais de uma tabela.

```
<secondary-table name="" />
<attributes>
```

Para uma herança de tabela única com discriminador, use o código a seguir:

```
<inheritance strategy="SINGLE_TABLE" />
<discriminator-value>node</discriminator-value>
<discriminator-column name="[column_name]" />
```

Os atributos com marca **id** são as colunas de chave primária. Verifique se a convenção de nomenclatura para essas colunas de chave primária são **idX** (id1, id2 e assim por diante) em que **X** é o índice de coluna na chave primária.

```
<id name="id1">
```

Altere somente o nome da coluna da chave primária.

```
      <column updatable="false" insertable="false" name="
[column_name]" />
      <generated-value strategy="TABLE" />
    </id>
```

basic. Usado para declarar os atributos do CMDB. Verifique se editou somente as propriedades **name** e **column_name**.


```

        <basic name="name">
            <column updatable="false" insertable="false" name="
[column_name]" />
        </basic>

```

Para uma herança de tabela única com discriminador, mapeie as classes estendidas da seguinte maneira:

```

    <entity name="[cmdb_class_name]" class="generic_db_adapter.nt"
name="nt">
        <discriminator-value>nt</discriminator-value>
        <attributes>
    </entity>
    <entity class="generic_db_adapter.unix" name="unix">
        <discriminator-value>unix</discriminator-value>
        <attributes>
    </entity>
    <entity name="[CMDB_class_name]" class="generic_db_adapter.
[CMDB[cmdb_class_name]">
        <table name="[default_table_name]" />
        <secondary-table name="" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
            <id name="id2">
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
            <id name="id3">
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>

```

O exemplo a seguir mostra um nome de atributo do CMDB sem prefixo:

```

        <basic name="[CMDB_attribute_name]">
            <column updatable="false" insertable="false" name="
[column_name]" />
        </basic>
        <basic name="[CMDB_attribute_name]">
            <column updatable="false" insertable="false" name="

```

```
[column_name]" />
    </basic>
    <basic name="[CMDB_attribute_name]">
        <column updatable="false" insertable="false" name="
[column_name]" />
    </basic>
</attributes>
</entity>
```

Esta é uma entidade de relacionamento. A convenção de nomenclatura é **end1Type_linkType_end2Type**. Neste exemplo, **end1Type** é **node** e **linkType** é **composition**.

```
<entity name="node_composition_[CMDB_class_name]"
class="generic_db_adapter.node_composition_[CMDB_class_name]">
    <table name="[default_table_name]" />
    <attributes>
        <id name="id1">
            <column updatable="false" insertable="false" name="
[column_name]" />
            <generated-value strategy="TABLE" />
        </id>
```

A entidade de destino é aquela para a qual esta propriedade está apontando. Neste exemplo, **end1** é mapeado para a entidade **node**.

many-to-one. Muitos relacionamentos podem ser conectados a um nó.

join-column. A coluna que contém IDs **end1** (os IDs de entidade de destino).

referenced-column-name. O nome de coluna na entidade de destino (**node**) que contém os IDs que são usados na coluna de junção.

```
<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. Um relacionamento pode ser conectado a um **[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. Esse é um exemplo de como adicionar um atributo de nó.

```

<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>

```

Vários arquivos ORM

Vários arquivos de mapeamento são aceitos. Cada nome de arquivo de mapeamento deve terminar com **orm.xml**. Todos os arquivos de mapeamento devem ser colocados na pasta META-INF do adaptador.

Convenções de nomenclatura

- Em cada entidade, a propriedade classe precisa corresponder à propriedade de nome com o prefixo de `generic_db_adapter`.
- As colunas de chave primária precisam ter nomes com a forma **idX** em que **X = 1, 2, ...**, de acordo com o número de chaves primárias na tabela.
- Os nomes de atributo precisam corresponder aos nomes de atributo de classe inclusive em relação a maiúsculas e minúsculas.
- O nome do relacionamento tem a forma `end1Type_linkType_end2Type`.
- Os TECs do CMDB, que também são nomes reservados em Java, devem ter como prefixo **gdba_**. Por exemplo, para o TEC do CMDB **goto**, a entidade ORM deve ter o nome **gdba_goto**.

Usando instruções SQL em linha em vez de nomes de tabela

Você pode mapear entidades para cláusulas `select` em linha em vez de mapear para tabelas de banco de dados. Isso equivale a definir uma visualização no banco de dados e mapear uma entidade para essa visualização. Por exemplo:

```

<entity class="generic_db_adapter.node">
  <table name="(select d.id as id1, d.name as name , d.os as
host_os from
Device d)" />

```

Nesse exemplo, os atributos de nó devem ser mapeados para as colunas id1, name e host_os, em vez de id, name e os.

As seguintes limitações se aplicam:

- A instrução SQL em linha está disponível somente ao usar o Hibernate como provedor de JPA.
- Parênteses ao redor da cláusula de seleção SQL em linha são obrigatórias.
- O elemento **<schema>** não deve estar presente no arquivo **orm.xml**. No caso do Microsoft SQL Server 2005, isso significa que todos os nomes de tabela devem ter como prefixo `dbo.`, em vez de defini-los globalmente por `<schema>dbo</schema>`.

O esquema orm.xml

A tabela a seguir explica os elementos comuns do arquivo **orm.xml**. O esquema completo pode ser encontrado em http://java.sun.com/xml/ns/persistence/orm_1_0.xsd. A lista não está completa e explica principalmente o comportamento específico da JPA (API de Persistência Java) padrão para o adaptador de banco de dados genérico.

Nome e caminho do elemento	Descrição	Atributos
entity-mappings	O elemento raiz do documento de mapeamento de entidade. Esse elemento deve ser exatamente o mesmo daquele fornecido nos arquivos de exemplo de adaptador de banco de dados genérico.	
descrição (entity-mappings)	Uma descrição de texto livre do documento de mapeamento de entidade. (Opcional)	
pacote (entity-mappings)	O nome do pacote Java que conterá as classes de mapeamento. Deve sempre conter o texto <code>generic_db_adapter</code> .	<ol style="list-style-type: none"> 1. Nome: nome Descrição: O nome do tipo de EC do UCMDB ao qual essa entidade está mapeada. Se essa entidade estiver mapeada para um link no CMDB, o nome da entidade deverá estar no formato <code><end_1>_<link_name>_<end_2></code>. Por exemplo, <code>node_composition_cpu</code> define uma entidade que será mapeada para o link de composição entre um nó e uma CPU. Se o nome do tipo de EC for o mesmo do nome

Nome e caminho do elemento	Descrição	Atributos
		<p>da classe Java sem o prefixo do pacote, esse campo poderá ser omitido.</p> <p>É necessário?: Opcional Tipo: Cadeia</p> <p>2. Nome: classe Descrição: O nome totalmente qualificado da classe Java que será criado para essa entidade de banco de dados. O nome do pacote da classe Java deve ser o mesmo do nome fornecido no elemento <code>package</code>. Você não pode usar palavras reservadas Java, como uma interface ou <code>switch</code>, como o nome da classe. Em vez disso, adicione o prefixo <code>gdba_</code> ao nome (para que a interface seja <code>generic_db_adapter.gdba_interface</code>).</p> <p>É necessário?: Obrigatório Tipo: Cadeia</p>
<p>table (entity-mappings>entity)</p>	<p>Esse elemento define a tabela primária da entidade de banco de dados. Pode aparecer somente uma vez. Obrigatório.</p>	<p>Nome: nome Descrição: O nome da tabela primária. Se o nome da tabela não contiver o esquema ao qual pertence, a tabela será pesquisada somente no esquema do usuário que foi utilizado para criar o ponto de integração. Isso também pode ser qualquer instrução <code>SELECT</code> válida. Se essa for uma instrução <code>SELECT</code>, será necessário encapsulá-la com parênteses.</p> <p>É necessário?: Obrigatório Tipo: Cadeia</p>
<p>secondary-table (entity-mappings>entity)</p>	<p>Esse elemento pode ser usado para definir uma tabela secundária da entidade de banco de dados. Essa tabela precisa ser conectada à tabela primária com um relacionamento 1-para-1. Você pode definir mais de uma tabela secundária.</p>	<p>Nome: nome Descrição: O nome da tabela secundária. Se o nome da tabela não contiver o esquema ao qual pertence, a tabela será pesquisada somente no esquema do usuário que foi utilizado para criar o ponto de integração. Isso também pode ser qualquer instrução</p>

Nome e caminho do elemento	Descrição	Atributos
	Opcional.	SELECT válida. Se essa for uma instrução SELECT, será necessário encapsulá-la com parênteses. É necessário?: Obrigatório Tipo: Cadeia
primary-key-join-column (entity-mappings > entity > secondary-table)	Se as tabelas secundária e primária não estiverem conectadas usando campos com o mesmo nome, esse elemento definirá o nome do campo de chave primária na tabela secundária que precisa estar conectada ao campo de chave primária da tabela primária.	Nome: nome Descrição: O nome do campo de chave primária na tabela secundária. Se esse elemento não existe, pressupõe-se que o campo de chave primária tenha o mesmo nome do campo de chave primária da tabela primária. É necessário?: Opcional Tipo: Cadeia
herança (entity-mappings>entity)	Se a entidade atual for a entidade pai de uma família de entidades de banco de dados, use esse elemento para marcá-la como tal. Opcional.	Nome: estratégia Descrição: Define a maneira como a herança é implementada no banco de dados. É necessário?: Obrigatório Tipo: Um dos seguintes valores: <ul style="list-style-type: none"> • SINGLE_TABLE: Esta entidade e todas as entidades filhos existem na mesma tabela. • JOINED: As entidades filho estão em tabelas juntadas. • TABLE_PER_CLASS: Cada entidade é definida completamente por uma tabela separada.
discriminator-column (entity-mappings>entity)	Se a herança for do tipo SINGLE_TABLE, este elemento será usado para definir o nome do campo usado para determinar o tipo de entidade para cada linha.	Nome: nome Descrição: O nome da coluna do discriminador. É necessário?: Obrigatório Tipo: Cadeia
discriminator-value (entity-mappings>entity)	Este elemento define o tipo da entidade específica na árvore de herança. Este nome precisa ser o mesmo do nome definido no arquivo	

Nome e caminho do elemento	Descrição	Atributos
	discriminator.properties para o grupo de valores deste tipo de entidade específico.	
attributes (entity-mappings>entity)	O elemento raiz de todos os mapeamentos de atributo de uma entidade.	
id (entity-mappings>entity attributes)	Este elemento define o campo de chave da entidade. Deve haver pelo menos um campo id definido. Se existir mais de um elemento id, seus campos criarão uma chave composta para a entidade. Você deve testar e evitar chaves compostas para entidades de EC (não para links).	Nome: nome Descrição: Uma cadeia de caracteres do tipo idX, em que X é um número entre 1 e 9. O primeiro id deve ser marcado como id1, o segundo como id2 e assim por diante. Este NÃO é o nome do atributo-chave do UCMDB. É necessário?: Obrigatório Tipo: Cadeia
basic (entity-mappings>entity attributes)	Este elemento define um mapeamento entre um campo na tabela, que não faz parte da chave primária da tabela, e um atributo do UCMDB.	Nome: nome Descrição: O nome do atributo do UCMDB para o qual esse campo está mapeado. Este atributo precisa existir no tipo de EC do UCMDB para o qual essa entidade atual está mapeada. É necessário?: Obrigatório Tipo: Cadeia
column (entity-mappings>entity> attributes> id -OU- (entity-mappings>entity> attributes> basic)	Define o nome da coluna na tabela para mapeamento básico ou um campo id.	1. Nome: nome Descrição: O nome do campo. É necessário?: Obrigatório Tipo: Cadeia 2. Nome: tabela Descrição: O nome da tabela à qual o campo pertence. Esse precisa ser a tabela primária ou uma das tabelas secundárias definidas para a entidade. Se esse atributo for omitido, será pressuposto que o campo pertence à tabela principal. É necessário: Opcional Tipo: Cadeia
one-to-one	Define uma coluna cujo valor	1. Nome: nome

Nome e caminho do elemento	Descrição	Atributos
(entity-mappings>entity>attributes)	está em outra tabela, e as duas tabelas estão conectadas usando um relacionamento um-para-um. Este elemento só é aceito para mapeamentos de entidade de link e não para outros tipos de EC. Essa é a única maneira de definir um mapeamento entre uma tabela e um link UCMDB.	<p>Descrição: Qual das duas extremidades este campo representa. É necessário?: Obrigatório Tipo: end1 ou end2</p> <p>2. Nome: target-entity Descrição: O nome da entidade à qual a extremidade se refere. É necessário?: Obrigatório Tipo: Um dos nomes de entidade definidos no documento de mapeamento de entidade.</p>
join-column (entity-mappings>entity attributes>one-to-one)	Define a maneira de juntar o target-entity definido no elemento um-para-um pai e a entidade atual.	<p>1. Nome: nome Descrição: O nome do campo na tabela atual que será usado para executar a junção um-para-um. É necessário?: Obrigatório Tipo: Cadeia</p> <p>2. Nome: nome Descrição: O nome de um campo na entidade de junção pela qual será executada a junção. Se esse atributo for omitido, pressupõe-se que a tabela de junção tenha uma coluna com o mesmo nome do campo definido no atributo de nome. É necessário?: Opcional Tipo: Cadeia</p>

Exemplo de criação do arquivo orm.xml

O exemplo apresentado aqui mostra como criar o arquivo **orm.xml**. Nesse exemplo, tabelas SQL em um banco de dados remoto são mapeadas para tipos de EC no UCMDB.

Dadas as tabelas com o seguinte formato no banco de dados remoto, preencha a tabela **Hosts** com nós, a tabela **IP_Addresses** com endereços IP e crie links entre os nós e endereços IP como a seguir:

Tabela de hosts

host_name	host_id
Teste1	1
Teste2	2
Teste3	3

Tabela 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

Tabela Host_IP_Link (links entre Nós e Endereços IP)

host_id	ip_id
1	1
2	2
2	3
3	4

A chave principal para a tabela **Hosts** é o campo **host_id** e a chave primária na tabela **IP_Addresses** é o campo **ip_id**. Na tabela **Host_IP_Link**, **host_id** e **ip_id** são chaves externas da tabela **Hosts** e **IP_Addresses**.

De acordo com as tabelas acima, crie o arquivo **orm.xml** como as etapas a seguir. As entidades usadas nesse exemplo são **node**, **ip_address** e **node_containment_ip_address**.

1. Crie a entidade **node** mapeando **host_id** a partir da tabela **Hosts** como a seguir:

```
<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>

```

A **classe** da entidade deve ser um tipo de EC que já existe no UCMDB. O **nome** da tabela é a tabela dentro do banco de dados que contém uma ID e as informações do host. O atributo ID é necessário para identificar hosts específicos e é usado posteriormente no mapeamento. Nesse exemplo, o atributo **name** dessa entidade será preenchido com a coluna **hostname** da tabela de hosts.

2. Para a próxima entidade, mapeie os endereços IP a partir da tabela interfaces:

```

<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. Em seguida, o link entre o Nó e o Endereço IP devem ser criados pela tabela de mapeamento e fazer referência ao campo **ip_id** (embora possa fazer referência aos campos **host_id** e **ip_id**, se desejado).

```

<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>

```

O nome da entidade para o contêiner tem o formato: [end1 CIT]_[link CIT]_[end2 CIT]. Então, para esse exemplo, como o Tipo de EC de link é **containment**, o nome da entidade para o contêiner é: **node_containment_ip_address** e a classe de entidade é **generic_db_adapter.node_containment_ip_address**. A ID é necessária nesse bloco de código e, embora esse exemplo funcione com uma única ID da Interface, as duas colunas devem usar as referências id1 e id2. O código para isso seria:

```

<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>

```

As duas extremidades desse link são 'vários para um' e 'um para um', significando que cada endereço IP será vinculado a 1 nó, mas um nó pode ser vinculado a vários endereços IP. As colunas incluídas são da tabela Links e fazem referência às tabelas Hosts e Interfaces.

O arquivo `reconciliation_types.txt`

Este arquivo é usado para configurar os tipos de reconciliação.

Cada linha no arquivo representa um TEC do CMDB que está conectado a um TEC de banco de dados federado na consulta TQL.

O arquivo `reconciliation_rules.txt` (para compatibilidade com versões anteriores)

Este arquivo é usado para configurar as regras de reconciliação se você quiser executar a reconciliação quando o DBMappingEngine estiver configurado no adaptador. Se você não usar o DBMappingEngine, o mecanismo de reconciliação genérico do UCMDB será usado e não haverá necessidade de configurar esse arquivo.

Cada linha no arquivo representa uma regra. Por exemplo:

```
multinode[node] expression[^node.name OR ip_address.name] end1_type  
[node] end2_type[ip_address] link_type[containment]
```

O multinó é preenchido com o nome de multinó (o TEC do CMDB que está conectado a um TEC de banco de dados federado na consulta TQL).

Esta expressão inclui a lógica que decide se dois multinós são iguais (um multinó no CMDB e o outro na fonte do banco de dados).

A expressão consiste em ORs ou ANDs.

A convenção em relação a nomes de atributos na parte da expressão é `[className].[attributeName]`. Por exemplo, `attributeName` na classe `ip_address` é escrito `ip_address.name`.

Para ver uma correspondência ordenada (se a primeira subexpressão OR retornar uma resposta de que os multinós não são iguais, a segunda subexpressão OR não será comparada), use `ordered expression` em vez de `expression`.

Para ignorar a diferenciação de maiúsculas e minúsculas durante uma comparação, use o sinal de controle (^).

Os parâmetros `end1_type`, `end2_type` e `link_type` são usados somente se a consulta TQL de reconciliação contém dois nós e não só um multinó. Nesse caso, a consulta TQL de reconciliação é `end1_type > (link_type) > end2_type`.

Não é necessário adicionar o layout relevante porque ele é obtido da expressão.

Tipos de regras de reconciliação

As regras de reconciliação têm a forma de condições OR e AND. Você pode definir essas regras em vários nós diferentes (por exemplo, o nó é identificado por `name from node` AND/OR `name from ip_address`).

As seguintes opções localizam uma correspondência:

- **Correspondência ordenada.** A expressão de reconciliação é lida da esquerda para a direita. Duas subexpressões `OR` são consideradas iguais se têm valores e estes são iguais. Duas subexpressões `OR` não são consideradas iguais se ambas têm valores e estes não são iguais. Para qualquer outro caso, não há decisão, e a subexpressão `OR` seguinte é testada para fins de igualdade.

name from node OR from ip_address. Se tanto o CMDB quanto a fonte de dados incluir `name` e eles forem iguais, os nós serão considerados iguais. Se ambos tiverem `name` mas não forem iguais, os nós não serão considerados iguais sem testar o `name` do `ip_address`. Se tanto o CMDB quanto a fonte de dados não tiver o `name of node`, o `name of ip_address` será verificado.

- **Correspondência regular.** Se houver igualdade em uma das duas subexpressões `OR`, o CMDB e a fonte de dados serão considerados iguais.

name from node OR from ip_address. Se não houver correspondência no `name of node`, o `name of ip_address` será verificado para ver se há igualdade.

Para reconciliações complexas, em que a entidade de reconciliação é modelada no modelo de classe como vários TECs e relacionamentos (como `node`), o mapeamento de um nó de incluirá todos os atributos relevantes de todos os TECs modelados.

Observação: Consequentemente, haverá uma limitação de que todos os atributos de reconciliação na fonte de dados deverá residir em tabelas que compartilham a mesma chave primária.

Outra limitação informa que a consulta TQL de reconciliação deve ter no máximo dois nós. Por exemplo, a consulta TQL `node > ticket` tem um nó no CMDB e um a `ticket` na fonte de dados.

Para reconciliar os resultados, o `name` precisa ser recuperado do nó e/ou `ip_address`.

Se o `name` no CMDB estiver no formato de `*.m.com`, um conversor poderá ser usado do CMDB para o banco de dados federado, e vice-versa, para converter esses valores.

A coluna `node_id` na tabela de `ticket` de banco de dados será usada para conectar entre as entidades (a associação definida também pode ser feita em uma tabela de nó):

Nó de BD		Endereço_IP do BD	
PK	<code>node_id</code>	PK	<code>ip_id</code>
	<code>nome</code>		<code>nome</code>

Ticket do BD	
PK	<code>ticket_id</code>
	<code>node_id</code>

Observação: As três tabelas precisam fazer parte da fonte do RDBMS federada e não do banco de dados do CMDB.

O arquivo `transformations.txt`

Este arquivo contém todas as definições do conversor.

O formato é que cada linha contém uma nova definição.

O modelo de arquivo `transformations.txt`

```
entity[[CMDB_class_name]] attribute[[CMDB_attribute_name]] to_DB_class
[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-
example.xml)]
from_DB_class
[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)]
```

entity. O nome da entidade conforme aparece no arquivo `orm.xml`.

attribute. O nome do atributo conforme aparece no arquivo `orm.xml`.

to_DB_class. O nome completo qualificado de uma classe que implementa a interface **`com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB`**. Os elementos entre parênteses são fornecidos para esse construtor de classe. Use este conversor para transformar valores do CMDB em valores de banco de dados, por exemplo, para acrescentar o sufixo de **`.com`** a cada nome de nó.

from_DB_class. O nome completo qualificado de uma classe que implementa a interface **`com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB`**. Os elementos entre parênteses são fornecidos para esse construtor de classe. Use este conversor para transformar valores de banco de dados em valores do CMDB, por exemplo, para acrescentar o sufixo de **`.com`** a cada nome de nó.

Para obter detalhes, consulte "[Conversores prontos](#)" na página 137.

O arquivo `discriminator.properties`

Este arquivo mapeia cada tipo de EC aceito (que também é usado como valor de discriminador em `orm.xml`) para uma lista separada por vírgulas de possíveis valores correspondentes da coluna do discriminador ou uma condição que corresponda a possíveis valores da coluna do discriminador.

Se uma condição for usada, use a sintaxe: `like (condition)`, onde `condition` é uma cadeia que pode conter os seguintes curingas:

- `%` (sinal de percentual) - permite que você corresponda qualquer cadeia de qualquer tamanho (incluindo uma cadeia de tamanho zero)
- `_` (sublinhado) - permite que você corresponda um único caractere

Por exemplo, `like (%unix%)` corresponderá a `unix`, `linux`, `unix-aix` e assim por diante. As condições `like` só podem ser aplicadas a colunas de cadeias.

Você também pode ter um valor de discriminador mapeado para qualquer valor que não pertença a outro discriminador informando 'all-other'.

Se o adaptador que você está criando usar recursos de discriminador, será necessário definir todos os valores de discriminador no arquivo **discriminator.properties**.

Exemplo de mapeamento de discriminador:

Por exemplo, o adaptador suporta os tipos de EC `node`, `nt`, e `unix`, e o banco de dados contém uma única tabela chamada `t_nodes` que contém uma coluna chamada **type**. Se o tipo for 10001, a linha representa um nó; se o tipo for 10004, ele representa uma máquina unix e assim por diante. O arquivo **discriminator.properties** pode ser semelhante a este:

```
node=10001, 10005
nt=10002,10003
unix=2%
mainframe=all-other
```

O arquivo **orm.xml** inclui o seguinte código:

```
<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>
```

O atributo `discriminator_column` é calculado da seguinte maneira:

- Se **type** contém 10002 ou 10003 para uma certa entrada, a entrada será mapeada para o TEC **nt**.
- Se **type** contém 10001 ou 10005 para uma certa entrada, a entrada será mapeada para o TEC **node**.
- Se **type** começa com 2 para uma certa entrada, a entrada será mapeada para o TEC **unix**.
- Qualquer outro valor na coluna **type** será mapeado para o TEC **mainframe**.

Observação: O TEC **node** também é o pai de **nt** e **unix**.

O arquivo replication_config.txt

Este arquivo contém uma lista separada por vírgulas de tipos de EC e de relacionamento cujas condições de propriedade são aceitas pelo plugin de replicação. Para obter detalhes, consulte "Plug-ins" na página 141.

O arquivo fixed_values.txt

Este arquivo permite configurar os valores fixos para atributos específicos de determinados TECs. Dessa maneira, cada um desses atributos pode receber um valor fixo que não está armazenado no banco de dados.

O arquivo deve conter zero ou mais entradas do seguinte formato:

```
entity[<entityName>] attribute[<attributeName>] value[<value>]
```

Por exemplo:

```
entity[ip_address] attribute[ip_domain] value[DefaultDomain]
```

O arquivo também suporta uma lista de constantes. Para definir uma lista de constantes, use a seguinte sintaxe:

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

O arquivo persistence.xml

Este arquivo é usado para substituir as configurações do Hibernate padrão e adicionar suporte a tipos de banco de dados que não são prontos (os tipos de banco de dados OOB são Oracle Server, Microsoft SQL Server e MySQL).

Se você precisar oferecer suporte a um novo tipo de banco de dados, verifique se forneceu um provedor de pool de conexão (o padrão é c3p0) e um driver JDBC para o banco de dados (coloque os arquivos *.jar na pasta do adaptador).

Para ver todos os valores do Hibernate disponíveis que podem ser alterados, verifique a classe **org.hibernate.cfg.Environment** (para obter detalhes, consulte <http://www.hibernate.org>).

Exemplo do arquivo 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.mercury.
```



```

jdbc.MercOracleDriver" />          <!--The connection url/-->
    <property name="hibernate.connection.url"
value="jdbc:mercury:oracle://artist:1521;sid=cmdb2" />
    <!--DB login credentials/-->          <property
name="hibernate.connection.username" value="CMDB" />
    <property name="hibernate.connection.password"
value="CMDB" />          <!--connection pool properties/-->
    <property name="hibernate.c3p0.min_size" value="5" />
    <property name="hibernate.c3p0.max_size" value="20" />
    <property name="hibernate.c3p0.timeout" value="300" />
    <property name="hibernate.c3p0.max_statements" value="50"
/>
    <property name="hibernate.c3p0.idle_test_period"
value="3000" />          <!--The dialect to use-->
    <property name="hibernate.dialect"
value="org.hibernate.dialect.OracleDialect" />          </properties>
</persistence-unit> </persistence>

```

Conversores prontos

Você pode usar os seguintes conversores (transformadores) para converter consultas federadas e trabalhos de replicação em dados de banco de dados e vice-versa.

Esta seção inclui os seguintes tópicos:

- "Conversores prontos" acima
- "O conversor SuffixTransformer" na página 140
- "O conversor PrefixTransformer" na página 140
- "O conversor BytesToStringTransformer" na página 141
- "O conversor StringDelimitedListTransformer" na página 141

O conversor enum-transformer

Este conversor usa um arquivo XML fornecido como um parâmetro de entrada.

O arquivo XML mapeia entre valores do CMDB embutidos no código-fonte e valores de banco de dados (enums). Se um dos valores não existir, você poderá optar por retornar o mesmo valor, retornar nulo ou lançar uma exceção.

O transformador realiza uma comparação entre duas cadeias usando um método de distinção entre maiúsculas e minúsculas ou sem distinção entre maiúsculas e minúsculas. O comportamento padrão é diferenciar maiúsculas de minúsculas. Para definir como distinção entre maiúsculas e minúsculas, use: `case-sensitive="false"` no elemento `enum-transformer`.

Use um arquivo de mapeamento XML para cada atributo de entidade.

Observação: Este conversor pode ser usado para os campos `to_DB_class` e `from_DB_class` no arquivo `transformations.txt`.

XSD de arquivo de entrada:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="enum-transformer">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="value" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="db-type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="integer"/>
            <xs:enumeration value="long"/>
            <xs:enumeration value="float"/>
            <xs:enumeration value="double"/>
            <xs:enumeration value="boolean"/>
            <xs:enumeration value="string"/>
            <xs:enumeration value="date"/>
            <xs:enumeration value="xml"/>
            <xs:enumeration value="bytes"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="cmdb-type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="integer"/>
            <xs:enumeration value="long"/>
            <xs:enumeration value="float"/>
            <xs:enumeration value="double"/>
            <xs:enumeration value="boolean"/>
            <xs:enumeration value="string"/>
            <xs:enumeration value="date"/>
            <xs:enumeration value="xml"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
        <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="required"/>
        <xs:attribute name="is-cmdb-value-null" type="xs:boolean"
use="optional"/>
        <xs:attribute name="is-db-value-null" type="xs:boolean"
use="optional"/>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Exemplo de conversão do valor 'sys' no valor 'System':

Neste exemplo, o valor `sys` no CMDB é transformado no valor `System` no banco de dados federado, enquanto o valor `System` no banco de dados federado é transformado no valor `sys` no CMDB.

Se o valor não existir no arquivo XML (por exemplo, a cadeia de caracteres `demo`), o conversor retornará o mesmo valor de entrada recebido.

```
<enum-transformer CMDB-type="string" DB-type="string" non-existing-  
value-action="return-original"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:noNamespaceSchemaLocation=" ../META-CONF/generic-enum-  
transformer.xsd">  
    <value CMDB-value="sys" external-DB-value="System" />  
</enum-transformer>
```

Exemplo de conversão de um valor externo ou do CMDB para um valor nulo:

Neste exemplo, um valor de `NNN` no banco de dados remoto é transformado em um valor nulo no banco de dados do CMDB.

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

Neste exemplo, o valor `OOO` no CMDB é transformado em um valor nulo no banco de dados.

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

O conversor SuffixTransformer

Este conversor é usado para adicionar ou remover sufixos do valor do CMDB ou do valor da fonte de banco de dados federado.

Há duas implementações:

- **com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddSuffixTransformer.** Adiciona o sufixo (fornecido como entrada) ao converter do valor de banco de dados federado para o valor CMDB e remove o sufixo ao converter do valor CMDB para o valor de banco de dados federado.
- **com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemoveSuffixTransformer.** Remove o sufixo (fornecido como entrada) ao converter do valor de banco de dados federado para o valor CMDB e adiciona o sufixo ao converter do valor CMDB para o valor de banco de dados federado.

O conversor PrefixTransformer

Este conversor é usado para adicionar ou remover um prefixo do valor do CMDB ou do valor da fonte de banco de dados federado.

Há duas implementações:

- **com.mercury.topaz.fcmbd.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddPrefixTransformer.** Adiciona o prefixo (fornecido como entrada) ao converter do valor de banco de dados federado para o valor CMDB e remove o prefixo ao converter do valor CMDB

para o valor de banco de dados federado.

- **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemovePrefixTransformer.** Remove o prefixo (fornecido como entrada) ao converter do valor de banco de dados federado para o valor CMDB e adiciona o prefixo ao converter do valor CMDB para o valor de banco de dados federado.

O conversor **BytesToStringTransformer**

Este conversor é usado para converter matrizes de bytes do CMDB em sua representação de cadeia de caracteres da fonte de banco de dados federado.

O conversor é:

com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer.

O conversor **StringDelimitedListTransformer**

Esse conversor é usado para transformar uma única lista de cadeias em uma lista de inteiros/cadeias no CMDB.

O conversor é: **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.StringDelimitedListTransformer.**

Plug-ins

O adaptador de banco de dados genérico oferece suporte aos seguintes plugins:

- Um plugin opcional para sincronização de topologia completa.
- Um plugin opcional para sincronização de alterações na topologia. Se não estiver implementado nenhum plugin para sincronizar alterações, será possível executar uma sincronização diferencial, mas ela na verdade será total.
- Um plugin opcional para layout de sincronização.
- Um plugin opcional para recuperar consultas aceitas para sincronização. Se esse plugin não estiver definido, todos os nomes TQL serão retomados.
- Um plugin opcional interno para alterar a definição e o resultado de TQL.
- Um plugin opcional interno para alterar uma solicitação de layout e o resultado de ECs.
- Um plugin opcional interno para alterar uma solicitação de layout e o resultado de relacionamentos.
- Um plugin opcional interno para alterar a ação de fazer push back de IDs.

Para ver detalhes sobre como implementar e implantar plugins, consulte "[Implementar um plugin](#)" na página 103.

Exemplos de Configuração

Esta seção fornece exemplos de configurações.

Esta seção inclui os seguintes tópicos:

- "Caso de uso" abaixo
- "Reconciliação de nó único" abaixo
- "Reconciliação de nó duplo" na página 144
- "Usando uma chave primária que contém mais de uma coluna" na página 147
- "Usando transformações" na página 149

Caso de uso

Caso de uso. Uma consulta TQL é:

node > (composition) > card

onde:

- **node** é a entidade do CMDB
- **card** é a entidade da fonte de banco de dados federado
- **composition** é o relacionamento entre eles

O exemplo é executado com base no banco de dados ED. ED nodes são armazenados na tabela Device e card é armazenado na tabela hwCards. Nos exemplos a seguir, card sempre é mapeado da mesma maneira.

Reconciliação de nó único

Neste exemplo, a reconciliação é executada com base na propriedade name.

Definição simplificada

A reconciliação é executada por node e enfatizada pela marca especial **CMDB-class**.

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-
CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="node" default-table-name="Device">
    <primary-key column-name="Device_ID" />
    <reconciliation-by-single-node>
      <or>
        <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
      </or>
    </reconciliation-by-single-node>
  </CMDB-class>
  <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="composition">
    <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID"
    <primary-key column-name="hwCards_Seq" />
    <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
    <attribute CMDB-attribute-name="card_vendor" column-
```

```

name="hwCardVendor" />
    <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>

```

Definição avançada

O arquivo orm.xml

Preste atenção à inclusão do mapeamento de relacionamentos. Para ver detalhes, consulte a seção de definições em "[O arquivo orm.xml](#)" na página 119.

Exemplo do arquivo orm.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/
persistence/orm http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
version="1.0">
    <description>Generic DB adapter orm</description>
    <package>generic_db_adapter</package>
    <entity class="generic_db_adapter.node" >
        <table name="Device"/>
        <attributes>
            <id name="id1">
                <column name="Device_ID" insertable="false"
                    updatable="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <basic name="name">
                <column name="Device_Name"/>
            </basic>
        </attributes>
    </entity>
    <entity class="generic_db_adapter.card" >
        <table name="hwCards"/>
        <attributes>
            <id name="id1">
                <column name="hwCards_Seq" insertable="false"
                    updatable="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <basic name="card_class">
                <column name="hwCardClass" insertable="false"
                    updatable="false"/>
            </basic>
            <basic name="card_vendor">
                <column name="hwCardVendor" insertable="false"
                    updatable="false"/>
            </basic>
        </attributes>
    </entity>

```

```

        </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>

```

O arquivo reconciliation_types.txt

Para obter detalhes, consulte ["O arquivo reconciliation_types.txt" na página 132.](#)

nó

O arquivo reconciliation_rules.txt

Para obter detalhes, consulte ["O arquivo reconciliation_rules.txt \(para compatibilidade com versões anteriores\)" na página 132.](#)

```
multinode[node] expression[node.name]
```

O arquivo transformations.txt

Este arquivo continua vazio porque nenhum valor precisa ser convertido neste exemplo.

Reconciliação de nó duplo

Neste exemplo, a reconciliação é calculada de acordo com a propriedade `name` de `node` e de `ip_address` com diferentes variações.

A consulta TQL de reconciliação é **node > (containment) > ip_address**.

Definição simplificada

A reconciliação é por `name` de `node` OR de `ip_address`:


```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-
CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="node" default-table-name="Device">
    <primary-key column-name="Device_ID" />
    <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment">
      <or>
        <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
        <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PREFERREDIPAddress" />
      </or>
    </reconciliation-by-two-nodes>
  </CMDB-class>
  <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
    <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
    <primary-key column-name="hwCards_Seq" />
    <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
    <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
    <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
  </class>
</generic-DB-adapter-config>
```

A reconciliação é por name de node AND de ip_address:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-
CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="node" default-table-name="Device">
    <primary-key column-name="Device_ID" />
    <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment">
      <and>
        <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
        <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PREFERREDIPAddress" />
      </and>
    </reconciliation-by-two-nodes>
  </CMDB-class>
  <class CMDB-class-name="card" default-table-name="hwCards">
```

```
connected-CMDB-class-name="node" link-class-name="containment">
  <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
  <primary-key column-name="hwCards_Seq" />
  <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
  <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
  <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
</class>
</generic-DB-adapter-config>
```

A reconciliação é por name de 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>
```

Definição avançada

O arquivo orm.xml

Como a expressão de reconciliação não é definida neste arquivo, a mesma versão deve ser usada para qualquer expressão de reconciliação.

O arquivo reconciliation_types.txt

Para obter detalhes, consulte "O arquivo reconciliation_types.txt" na página 132.

nó

O arquivo `reconciliation_rules.txt`

Para obter detalhes, consulte "[O arquivo `reconciliation_rules.txt` \(para compatibilidade com versões anteriores\)](#)" na página 132.

```
multinode[node] expression[ip_address.name OR node.name] end1_type  
[node] end2_type[ip_address] link_type[containment]
```

```
multinode[node] expression[ip_address.name AND node.name] end1_type  
[node] end2_type[ip_address] link_type[containment]
```

```
multinode[node] expression[ip_address.name] end1_type[node] end2_type  
[ip_address] link_type[containment]
```

O arquivo `transformations.txt`

Este arquivo continua vazio porque nenhum valor precisa ser convertido neste exemplo.

Usando uma chave primária que contém mais de uma coluna

Se a chave primária for composta por mais de uma coluna, o seguinte código será adicionado às definições de XML:

Definição simplificada

Há mais de uma marca de chave primária e, para cada coluna, há uma marca.

```
<class CMDB-class-name="card" default-table-name="hwCards"  
connected-CMDB-class-name="node" link-class-name="containment">  
  <foreign-primary-key column-name="Device_ID" CMDB-class-  
primary-key-column="Device_ID" />  
  <primary-key column-name="Device_ID" />  
  <primary-key column-name="hwBusesSupported_Seq" />  
  <primary-key column-name="hwCards_Seq" />  
  <attribute CMDB-attribute-name="card_class" column-  
name="hwCardClass" />  
  <attribute CMDB-attribute-name="card_vendor" column-  
name="hwCardVendor" />  
  <attribute CMDB-attribute-name="card_name" column-  
name="hwCardName" />  
</class>
```

Definição avançada

O arquivo `orm.xml`

É adicionada uma nova entidade `id` que mapeia para as colunas de chave primária. As entidades que usam essa entidade `id` precisam adicionar uma marca especial.

Se você usar uma chave estrangeira (marca `join-column`) para essa chave primária, deverá mapear entre cada coluna na chave estrangeira para uma coluna na chave primária.

Para obter detalhes, consulte "[O arquivo `orm.xml`](#)" na página 119.

Exemplo do arquivo `orm.xml`:

```
<entity class="generic_db_adapter.card" >
  <table name="hwCards"/>
  <attributes>
    <id name="id1">
      <column name="Device_ID" insertable="false"
updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id2">
      <column name="hwBusesSupported_Seq" insertable="false"
updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id3">
      <column name="hwCards_Seq" insertable="false"
updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
  </attributes>
</entity>

<entity class="generic_db_adapter.node_containment_card" >
  <table name="hwCards"/>
  <attributes>
    <id name="id1">
      <column name="Device_ID" insertable="false"
updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id2">
      <column name="hwBusesSupported_Seq" insertable="false"
updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id3">
      <column name="hwCards_Seq" insertable="false"
updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <many-to-one name="end1" target-entity="node">
      <join-column name="Device_ID" insertable="false"
updatable="false"/>
    </many-to-one>
    <one-to-one name="end2" target-entity="card">
      <join-column name="Device_ID" referenced-column-
name="Device_ID" insertable="false" updatable="false"/>
      <join-column name="hwBusesSupported_Seq" referenced-
column-name="hwBusesSupported_Seq" insertable="false"
updatable="false"/>
      <join-column name="hwCards_Seq" referenced-column-
name="hwCards_Seq" insertable="false" updatable="false"/>
    </one-to-one>
  </attributes>
</entity>
```

```
</entity>
</entity-mappings>
```

Usando transformações

No exemplo a seguir, o transformador **enum** genérico é convertido dos valores 1, 2, 3 para os valores a, b, c respectivamente na coluna `name`.

O arquivo de mapeamento é `generic-enum-transformer-example.xml`.

```
<enum-transformer CMDB-type="string" DB-type="string" non-existing-
value-action="return-original"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/generic-enum-
transformer.xsd">
    <value CMDB-value="1" external-DB-value="a" />
    <value CMDB-value="2" external-DB-value="b" />
    <value CMDB-value="3" external-DB-value="c" />
</enum-transformer>
```

Definição simplificada

```
<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.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-
transformer-example.
xml)" />
            <connected-node-attribute CMDB-attribute-name="name"
                column-name="Device_PREFERREDIPAddress" />
        </or>
    </reconciliation-by-two-nodes>
</CMDB-class>
```

Definição avançada

Há uma alteração somente no arquivo **transformation.txt**.

O arquivo `transformations.txt`

Verifique se os nomes de atributo e de entidade são os mesmos no arquivo `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)]
```

Arquivos de log do adaptador

Para entender os fluxos de cálculo e o ciclo de vida do adaptador, bem como visualizar informações de depuração, você pode consultar os seguintes arquivos de log.

Esta seção inclui os seguintes tópicos:

- "Níveis de log" abaixo
- "Locais dos arquivos" abaixo

Níveis de log

É possível configurar o nível de log de cada um dos logs.

Em um editor de texto, abra o arquivo **C:\hp\UCMDB\UCMDBServer\conf\log\fcmdb.gdba.properties**

.

O nível de log padrão é **ERROR**:

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL  
loglevel=ERROR
```

- Para aumentar o nível de log para todos os arquivos de log, altere **loglevel=ERROR** para **loglevel=DEBUG** ou **loglevel=INFO**.
- Para alterar o nível de log para um arquivo específico, altere a linha de categoria **log4j** específica adequadamente. Por exemplo, para alterar o nível de log de `fcmdb.gdba.dal.sql.log` para **INFO**, altere:

```
log4j.category.fcmdb.gdba.dal.SQL=${loglevel},  
fcmdb.gdba.dal.SQL.appender
```

para:

```
log4j.category.fcmdb.gdba.dal.SQL=INFO,fcmdb.gdba.dal.SQL.appender
```

Locais dos arquivos

Os arquivos de log ficam localizados no diretório **C:\hp\UCMDB\UCMDBServer\runtime\log**.

- **Fcmdb.gdba.log**

O log do ciclo de vida do adaptador. Fornece detalhes sobre quando o adaptador iniciou ou parou, além de quais TECs são aceitos por esse adaptador.

Consulte sobre erros de iniciação (carregamento/descarregamento do adaptador).

- **fcmdb.log**

Consulte sobre exceções.

- **cmdb.log**

Consulte sobre exceções.

- **Fcmdb.gdba.mapping.engine.log**

O log do mecanismo de mapeamento. Fornece detalhes sobre a consulta TQL de reconciliação que o mecanismo de mapeamento usa, além das topologias de reconciliação que são comparadas com a fase de conexão.

Consulte este log quando uma consulta TQL não fornecer resultados, mesmo que você saiba que há ECs relevantes no banco de dados ou que os resultados sejam inesperados (verifique a reconciliação).

- **Fcmdb.gdba.TQL.log**

O log TQL. Fornece detalhes sobre as consultas TQL e seus resultados.

Consulte este log quando uma consulta TQL não retornar resultados e o log do mecanismo de mapeamento mostrar que não há resultados na fonte de dados federados.

- **Fcmdb.gdba.dal.log**

O log do ciclo de vida de DAL. Fornece detalhes sobre conexão de banco de dados e geração de TECs.

Consulte este log quando não puder se conectar ao banco de dados ou quando houver TECs ou atributos não aceitos pela consulta.

- **Fcmdb.gdba.dal.command.log**

O log de operações de DAL. Fornece detalhes sobre operações de DAL internas que são chamadas. (Este log é semelhante ao `cmdb.dal.command.log`).

- **Fcmdb.gdba.dal.SQL.log**

O log de consultas SQL de DAL. Fornece detalhes sobre JPAQLs chamadas (consultas SQL orientadas a objeto) e seus resultados.

Consulte este log quando não puder se conectar ao banco de dados ou quando houver TECs ou atributos não aceitos pela consulta.

- **Fcmdb.gdba.hibernate.log**

O log do Hibernate. Fornece detalhes sobre as consultas SQL que são executadas, a análise de cada JPAQL em relação a SQL, os resultados das consultas, os dados referentes ao cache do Hibernate e assim por diante. Para ver detalhes sobre o Hibernate, consulte "[Hibemar como provedor de JPA](#)" na página 84.

Referências externas

Para ver detalhes sobre a especificação JavaBeans 3.0, consulte <http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html>.

Solução de problemas e limitações

Esta seção descreve a solução de problemas e as limitações do adaptador de banco de dados genérico.

Limitações gerais

- Não há suporte à autenticação NTLM do SQL Server.
- Ao atualizar um pacote de adaptadores, use o Notepad++, o UltraEdit ou algum outro editor de texto de terceiros em vez do Bloco de Notas (qualquer versão) da Microsoft Corporation para editar os arquivos de modelo. Isso impede o uso de símbolos especiais, que causam falha na implantação do pacote preparado.

Limitações de JPA

- Todas as tabelas precisam ter uma coluna de chave primária.
- Os nomes de atributo de classe do CMDB precisam seguir a convenção de nomenclatura JavaBeans (por exemplo, os nomes precisam iniciar com letras minúsculas).
- Dois ECs conectados a um relacionamento no modelo de classe precisam ter associação direta no banco de dados (por exemplo, se `node` estiver conectado a `ticket`, deverá haver uma chave estrangeira ou tabela vinculável que se conecte a ele).
- Várias tabelas mapeadas para o mesmo TEC precisam compartilhar a mesma tabela de chave primária.

Limitações funcionais

- Você não pode criar um relacionamento manual entre o CMDB e os TECs federados. Para poder definir relacionamentos virtuais, uma lógica de relacionamento especial precisa estar definida (ela pode se basear nas propriedades da classe federada).
- Os TECs federados não podem ser TECs acionadores em uma regra de impacto, mas podem ser incluídos em uma consulta TQL de análise.
- Um TEC federado pode fazer parte de um TQL de melhoria, mas não pode ser usado como o nó em que a melhoria é executada (você não pode adicionar, atualizar ou excluir o TEC federado).
- Não há suporte ao uso de um qualificador de classe em uma condição.
- Não há suporte a subgráficos.
- Não há suporte a relacionamentos compostos.
- O `id` do CMDB de EC externo é composto por sua chave primária, e não seus atributos-chave.
- Uma coluna de tipo `bytes` não pode ser usada como coluna de chave primária no Microsoft SQL Server.
- O cálculo de consulta TQL falhará se as condições de atributo definidas em um nó federado não tiverem tido seus nomes mapeados no arquivo `orm.xml`.
- O adaptador de banco de dados genérico não aceita a Autenticação do Windows para o SQL Server.

Capítulo 5

Desenvolvimento de adaptadores Java

Este capítulo inclui:

Visão geral do Federation Framework	153
Interação de adaptador e mapeamento com o Federation Framework	158
Fluxo do Federation Framework para consultas TQL federadas	158
Interações entre o Federation Framework, servidor, adaptador e mecanismo de mapeamento	160
Fluxo do Federation Framework para população	168
Interfaces de adaptador	170
Depurar recursos do adaptador	171
Adicionar um adaptador para uma nova fonte de dados externos	171
Implementar o mecanismo de mapeamento	179
Criar um adaptador de amostra	180
Marcas e propriedades de configuração XML	181

Visão geral do Federation Framework

Observação:

- O termo **relationship** é equivalente ao termo **link**.
- O termo **CI** é equivalente ao termo **object**.
- Um gráfico é uma coleção de nós e links.

A funcionalidade do Federation Framework usa uma API para recuperar informações de fontes federadas. O Federation Framework fornece três recursos principais:

- **Federação** em tempo real. Todas as consultas são executadas sobre repositórios de dados originais e os resultados são criados em tempo real no CMDB.
- **População**. Popula os dados (dados de topologia e propriedades de EC) de uma fonte de dados externa para o CMDB.
- **Push de dados**. Faz push dos dados (dados de topologia e propriedades de EC) de uma fonte de dados remota para o CMDB local.

Todos os tipos de ação exigem um adaptador para cada repositório de dados, que pode fornecer os recursos específicos do repositório de dados e recuperar e/ou atualizar os dados necessários. Cada solicitação do repositório de dados é feita por meio de seu adaptador.

Esta seção também inclui os seguintes tópicos:

- ["Federação em tempo real" abaixo](#)
- ["Push de Dados" na página seguinte](#)
- ["População" na página 156](#)

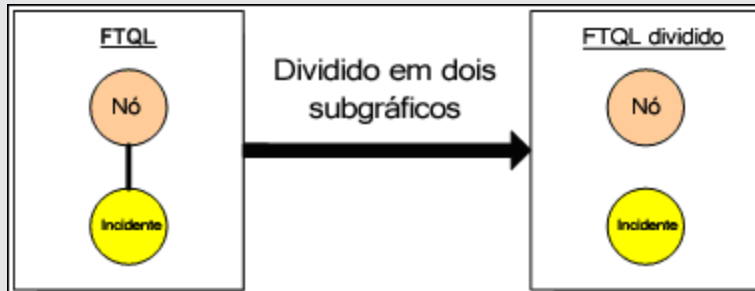
Federação em tempo real

As consultas TQL federadas permitem a recuperação de dados de qualquer repositório de dados externo sem replicar seus dados.

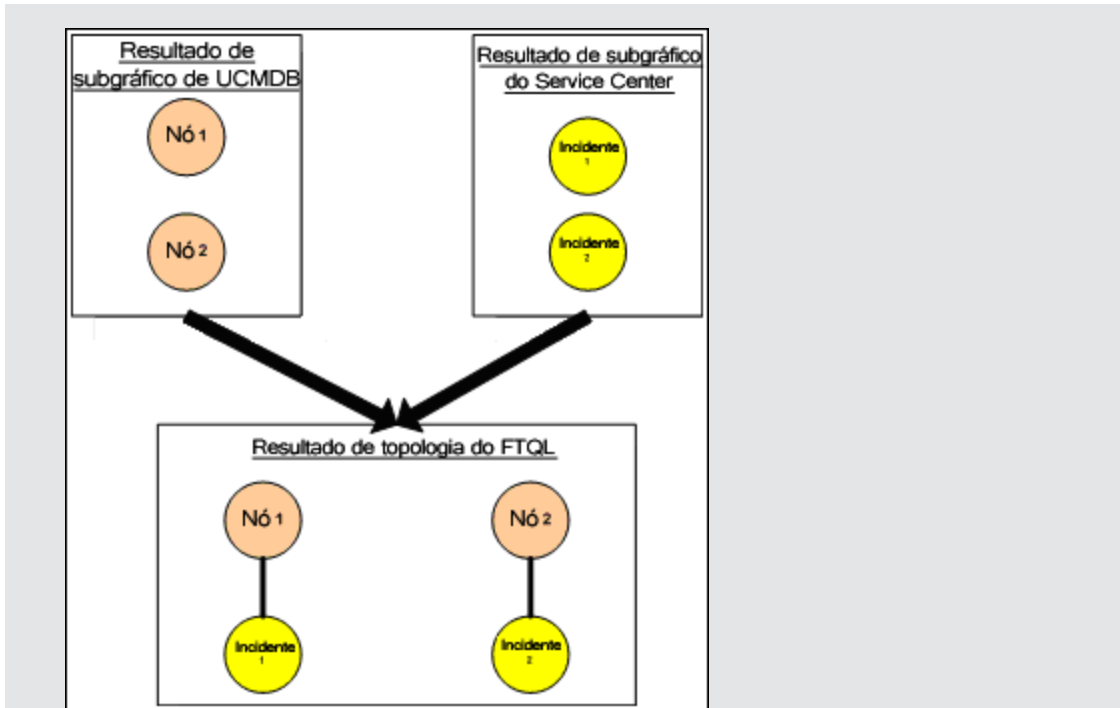
Uma consulta TQL federada usa adaptadores que representam repositórios de dados externos, para criar relacionamentos externos apropriados entre ECs de diferentes repositórios de dados externos e os ECs do UCMDB.

Exemplo de fluxo de federação em tempo real:

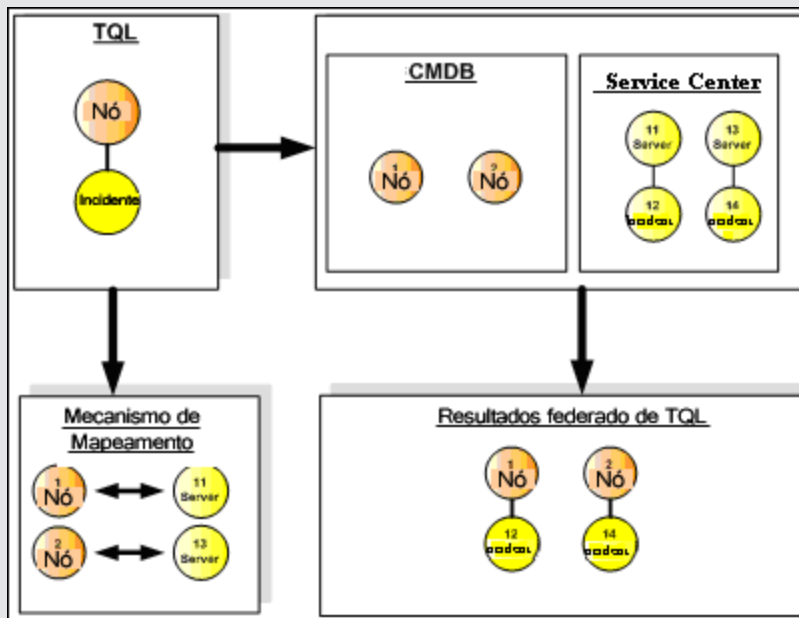
1. O Federation Framework divide uma consulta TQL federada em vários subgráficos, em que todos os nós de um subgráfico se referem ao mesmo repositório de dados. Cada subgráfico é conectado a outros subgráficos por um relacionamento virtual (mas ele próprio não contém relacionamentos virtuais).



2. Depois que a consulta TQL federada é dividida em subgráficos, o Federation Framework calcula a topologia de cada subgráfico e conecta dois subgráficos apropriados criando relacionamentos virtuais entre os nós apropriados.



3. Depois que a topologia TQL federada é calculada, o Federation Framework recupera um layout para o resultado da topologia.



Push de Dados

Use o fluxo de push de dados para sincronizar dados do CMDB local para um serviço remoto ou repositório de dados de destino.

No push de dados, os repositórios de dados são divididos em duas categorias: origem (CMDB local) e destino. Os dados são recuperados do repositório de dados de origem e atualizados no

repositório de dados de destino. O processo de push de dados se baseia em nomes de consulta, significando que os dados são sincronizados entre os repositórios de dados de origem (CMDB local) e de destino e são recuperados por um nome de consulta TQL do CMDB local.

O fluxo do processo de push de dados inclui as seguintes etapas:

1. Recuperar o resultado de topologia com assinaturas do repositório de dados de origem.
2. Comparar os novos resultados com os anteriores.
3. Recuperar um layout completo (ou seja, todas as propriedades de EC) de ECs e relacionamentos – somente para os resultados alterados.
4. Atualizar o repositório de dados de destino com o layout completo recebido de ECs e relacionamentos. Se algum EC ou relacionamento for excluído no repositório de dados de origem e a consulta for exclusiva, o processo de replicação também removerá os ECs e relacionamentos no repositório de dados de destino.

O CMDB tem 2 fontes de dados ocultas (**hiddenRMIDataSource** e **hiddenChangesDataSource**), que sempre são a fonte de dados de 'origem' nos fluxos de push de dados. Para implementar um novo adaptador para fluxos de push de dados, basta apenas implementar o adaptador de 'destino'.

População

Use o fluxo de população para popular o CMDB com dados de fontes externas.

O fluxo sempre usa um repositório de dados de 'origem' para recuperar os dados, e faz push dos dados recuperados para a Sonda em um processo semelhante ao fluxo de um trabalho de descoberta.

Para implementar um novo adaptador para fluxos de população, basta apenas implementar o adaptador de origem, já que a Sonda de Fluxo de Dados atua como destino.

O adaptador no fluxo de população é executado na Sonda. A depuração e o registro em log devem ser executados na Sonda e não no CMDB.

O fluxo de população se baseia em nomes de consulta, ou seja, os dados são sincronizados entre o repositório de dados de origem e a Sonda de Fluxo de Dados e são recuperados por um nome de consulta no repositório de dados de origem. Por exemplo, no UCMDB, o nome de consulta é o nome da consulta TQL. Entretanto, em outro repositório de dados, o nome da consulta pode ser um nome de código que retorna dados. O adaptador foi projetado para manipular corretamente o nome da consulta.

Cada trabalho pode ser definido como um trabalho exclusivo. Isso significa que os ECs e relacionamentos nos resultados do trabalho são exclusivos no CMDB local, e nenhuma consulta pode transmiti-los ao destino. O adaptador do repositório de dados de origem oferece suporte a consultas específicas e pode recuperar os dados desse repositório de dados. O adaptador do repositório de dados de destino permite a atualização dos dados recuperados nesse repositório de dados.

Fluxo SourceDataAdapter

- Recupera o resultado de topologia com assinaturas do repositório de dados de origem.
- Compara os novos resultados com os anteriores.

- Recupera um layout completo (ou seja, todas as propriedades de EC) de ECs e relacionamentos – somente para os resultados alterados.
- Atualiza o repositório de dados de destino com o layout completo recebido de ECs e relacionamentos. Se algum EC ou relacionamento for excluído no repositório de dados de origem e a consulta for exclusiva, o processo de replicação também removerá os ECs e relacionamentos no repositório de dados de destino.

Fluxo SourceChangesDataAdapter

- Recupera o resultado de topologia que ocorreu desde a última data determinada.
- Recupera um layout completo (ou seja, todas as propriedades de EC) de ECs e relacionamentos – somente para os resultados alterados.
- Atualiza o repositório de dados de destino com o layout completo recebido de ECs e relacionamentos. Se algum EC ou relacionamento for excluído no repositório de dados de origem e a consulta for exclusiva, o processo de replicação também removerá os ECs e relacionamentos no repositório de dados de destino.

Fluxo PopulateDataAdapter

- Recupera a topologia completa com o resultado de layout solicitado.
- Usa o mecanismo de partes de topologia para recuperar os dados em partes.
- A sonda filtra e desconsidera quaisquer dados que já foram transmitidos em execuções anteriores.
- Atualiza o repositório de dados de destino com o layout recebido de ECs e relacionamentos. Se algum EC ou relacionamento for excluído no repositório de dados de origem e a consulta for exclusiva, o processo de replicação também removerá os ECs e relacionamentos no repositório de dados de destino.

Fluxo PopulateChangesDataAdapter

- Recupera a topologia com o resultado de layout solicitado que tem alterações desde a última execução.
- Usa o mecanismo de partes de topologia para recuperar os dados em partes.
- A sonda filtra e desconsidera quaisquer dados que já foram transmitidos em execuções anteriores (incluindo este fluxo).
- Atualiza o repositório de dados de destino com o layout recebido de ECs e relacionamentos. Se algum EC ou relacionamento for excluído no repositório de dados de origem e a consulta for exclusiva, o processo de replicação também removerá os ECs e relacionamentos no repositório de dados de destino.

Interação de adaptador e mapeamento com o Federation Framework

Um adaptador é uma entidade no UCMDb que representa os dados externos (dados que não são salvos no UCMDb). Em fluxos federados, todas as interações com fontes de dados externos são executadas através de adaptadores. O fluxo de interação e as interfaces de adaptador do Federation Framework são diferentes para replicação e consultas TQL federadas.

Esta seção também inclui os seguintes tópicos:

- ["Ciclo de vida do adaptador" abaixo](#)
- ["Métodos assist do adaptador" abaixo](#)

Ciclo de vida do adaptador

Uma instância de adaptador é criada para cada repositório de dados externo. O adaptador começa seu ciclo de vida com a primeira ação aplicada a ele (por exemplo, `calculate TQL` ou `retrieve/update data`). Quando o método **start** é chamado, o adaptador recebe informações do ambiente, como a configuração do repositório de dados, o logger e assim por diante. O ciclo de vida do adaptador termina quando o repositório de dados é removido da configuração, e o método **shutdown** é chamado. Isso significa que o adaptador tem informações de estado e pode conter a conexão ao repositório de dados externo se for necessário.

Métodos assist do adaptador

O adaptador tem vários métodos `assist` que podem adicionar configurações de repositório de dados externo. Esses métodos não fazem parte do ciclo de vida do adaptador e criam um novo adaptador sempre que são chamados.

- O primeiro método testa a conexão ao repositório de dados externo para uma determinada configuração. `testConnection` pode ser executado no servidor UCMDb ou na Sonda de Fluxo de Dados, dependendo do tipo de adaptador.
- O segundo método é relevante apenas para o adaptador de origem e retorna as consultas válidas para replicação. (Esse método é executado somente na Sonda.)
- O terceiro método é relevante apenas para fluxos de federação e de população e retorna as classes externas aceitas pelo repositório de dados externo. (Esse método é executado somente no servidor UCMDb.)

Todos esses métodos são usados ao criar ou visualizar configurações de integração.

Fluxo do Federation Framework para consultas TQL federadas

Esta seção inclui os seguintes tópicos:

- ["Definições e termos" na página seguinte](#)
- ["Mecanismo de mapeamento" na página seguinte](#)
- ["Adaptador federado" na página seguinte](#)

Consulte "[Interações entre o Federation Framework, servidor, adaptador e mecanismo de mapeamento](#)" na página seguinte para diagramas que ilustram as interações entre o Federation Framework, o CMDB, o adaptador e o Mecanismo de Mapeamento.

Definições e termos

Dados de reconciliação. A regra para corresponder ECs do tipo especificado que são recebidos do CMDB e do repositório de dados externos. A regra de reconciliação pode ser de três tipos:

- **Reconciliação de ID.** Pode ser usada somente se o repositório de dados externo contém a ID do CMDB de objetos de reconciliação.
- **Reconciliação de propriedade.** É usada quando a correspondência pode ser feita por propriedades somente do tipo de EC de reconciliação.
- **Reconciliação de topologia.** É usada quando você precisa das propriedades de TECs adicionais (não só do TEC de reconciliação) para executar uma correspondência em ECs de reconciliação. Por exemplo, é possível executar reconciliação do tipo de nó pela propriedade `name` que pertence ao TEC `ip_address`.

Objeto de reconciliação. O objeto é criado pelo adaptador de acordo com os dados de reconciliação recebidos. Esse objeto deve se referir a um EC externo e é usado pelo Mecanismo de Mapeamento para se conectar entre os ECs externos e os ECs do CMDB.

Tipo de EC de reconciliação. O tipo de ECs que representam objetos de reconciliação. Esses ECs precisam ser armazenados tanto no CMDB quanto em repositórios de dados externos.

Mecanismo de mapeamento. Um componente que identifica os relacionamentos entre os ECs de diferentes repositórios de dados que têm um relacionamento virtual entre eles. A identificação é executada através dos objetos de reconciliação do CMDB e dos objetos de reconciliação de ECs externos.

Mecanismo de mapeamento

O Federation Framework usa o Mecanismo de Mapeamento para calcular a consulta TQL federada. O Mecanismo de Mapeamento se conecta entre os ECs que são recebidos de diferentes repositórios de dados e são conectados por relacionamentos virtuais. O Mecanismo de Mapeamento também fornece dados de reconciliação para o relacionamento virtual. Uma extremidade do relacionamento virtual precisa se referir ao CMDB. Essa extremidade é um tipo `reconciliation`. Para o cálculo dos dois subgráficos, um relacionamento virtual pode começar de qualquer nó final.

Adaptador federado

O adaptador Federado oferece dois tipos de dados dos repositórios de dados externos: dados de EC externos e objetos de reconciliação que pertencem a ECs externos.

- **Dados de EC externos.** Os dados de EC externos que não existem no CMDB. São os dados de destino do repositório de dados externo.
- **Dados de objeto de reconciliação.** Os dados auxiliares que são usados pelo Federation Framework para se conectar entre os ECs do CMDB e os dados externos. Cada objeto de reconciliação deve se referir a um EC externo. O tipo de objeto de reconciliação é o tipo (ou subtipo) de uma das extremidades de relacionamento virtual das quais os dados são recuperados. Os objetos de reconciliação devem ajustar o adaptador recebido aos dados de reconciliação. O objeto de reconciliação pode ser de um destes três tipos:

`IdReconciliationObject`, `PropertyReconciliationObject` ou `TopologyReconciliationObject`.

Nas interfaces baseadas em `DataAdapter` (`DataAdapter`, `PopulateDataAdapter` e `PopulateChangesDataAdapter`), a reconciliação é solicitada como parte da definição da consulta.

Interações entre o Federation Framework, servidor, adaptador e mecanismo de mapeamento

Os diagramas a seguir ilustram as interações entre o Federation Framework, o UCMDDB, o adaptador e o Mecanismo de Mapeamento. A consulta TQL federada no diagrama de exemplo tem somente um relacionamento virtual, por isso somente o UCMDDB e um repositório de dados externo estão envolvidos na consulta TQL federada.

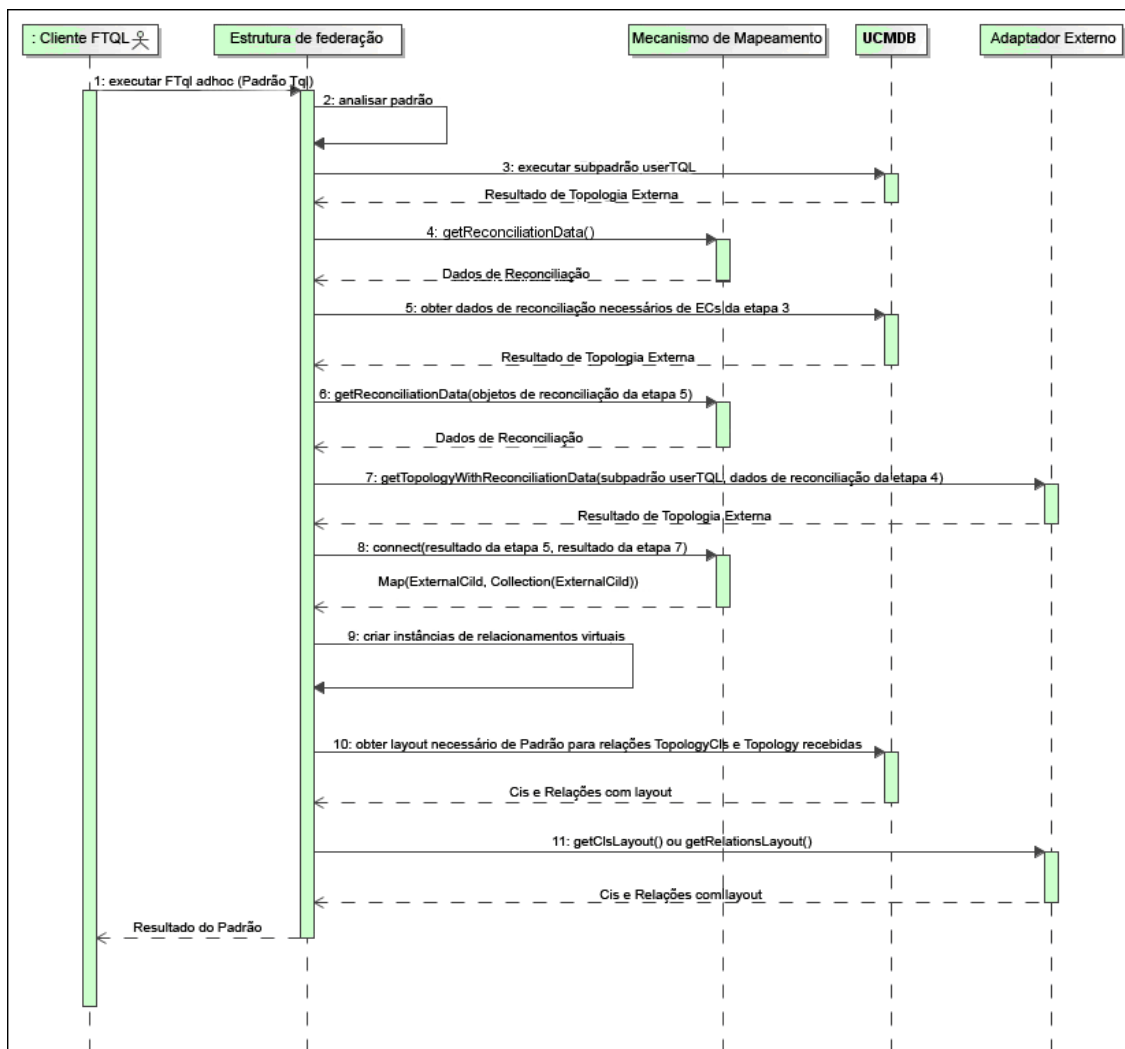
Esta seção inclui os seguintes tópicos:

- ["O cálculo começa na extremidade do servidor" abaixo](#)
- ["O cálculo começa na extremidade do adaptador externo" na página 163](#)
- ["Exemplo do fluxo do Federation Framework para consultas TQL federadas" na página 164](#)

No primeiro diagrama, o cálculo começa no UCMDDB e no segundo diagrama no adaptador externo. Cada etapa no diagrama inclui referências à chamada de método apropriada do adaptador ou da interface do mecanismo de mapeamento.

O cálculo começa na extremidade do servidor

O diagrama de sequência a seguir ilustra as interações entre o Federation Framework, o UCMDDB, o adaptador e o Mecanismo de Mapeamento. A consulta TQL federada no diagrama de exemplo tem somente um relacionamento virtual, por isso somente o UCMDDB e um repositório de dados externo estão envolvidos na consulta TQL federada.

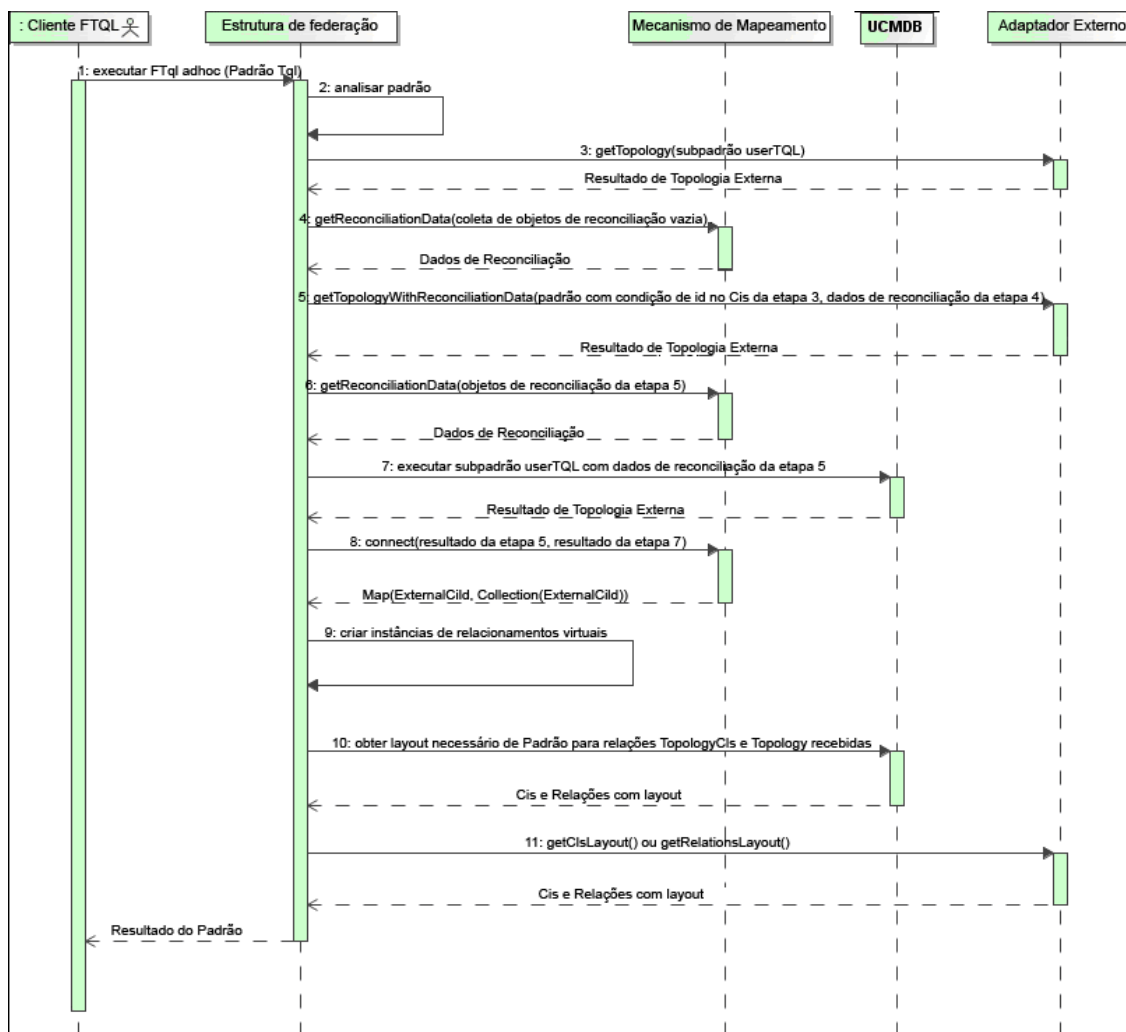


Os números nesta imagem são explicados a seguir:

Número	Explicação
1	O Federation Framework recebe uma chamada para um cálculo de TQL federado.
2	O Federation Framework analisa o adaptador, localiza o relacionamento virtual e divide o TQL original em dois subadaptadores – um para o UCMDB e outro para o repositório de dados externo.
3	O Federation Framework solicita a topologia do sub-TQL do UCMDB.
4	Depois de receber os resultados de topologia, o Federation Framework chama o Mecanismo de Mapeamento apropriado para o relacionamento virtual atual e solicita os dados de reconciliação. O parâmetro <code>reconciliationObject</code> está vazio nessa etapa, ou seja, nenhuma condição é adicionada aos dados de reconciliação nessa chamada. Os dados de reconciliação retornados definem quais dados são necessários para corresponder os ECs de reconciliação no UCMDB ao repositório de dados externo. Os dados de reconciliação podem ser de um destes tipos:

Número	Explicação
	<ul style="list-style-type: none"> • IdReconciliationData. Os ECs são reconciliados de acordo com seu ID. • PropertyReconciliationData. Os ECs são reconciliados de acordo com as propriedades de um dos ECs. • TopologyReconciliationData. Os ECs são reconciliados de acordo com a topologia (por exemplo, para reconciliar ECs de nó, o endereço IP de IP é necessário também).
5	O Federation Framework solicita dados de reconciliação para os ECs das extremidades do relacionamento virtual que foram recebidas na etapa "3" na página anterior do UCMDB.
6	O Federation Framework chama o Mecanismo de Mapeamento para recuperar os dados de reconciliação. Nesse estado (em contraste com a etapa "3" na página anterior), o Mecanismo de Mapeamento recebe os objetos de reconciliação da etapa "5" acima como parâmetros. O Mecanismo de Mapeamento traduz o objeto de reconciliação recebido para a condição nos dados de reconciliação.
7	O Federation Framework solicita a topologia do sub-TQL do repositório de dados externo. O adaptador externo recebe os dados de reconciliação da etapa "6" acima como um parâmetro.
8	O Federation Framework chama o Mecanismo de Mapeamento para se conectar entre os resultados recebidos. O parâmetro <code>firstResult</code> é o resultado de topologia externa recebido do UCMDB na etapa "5" acima e o parâmetro <code>secondResult</code> é o resultado de topologia externa recebido do Adaptador Externo na etapa "7" acima. O Mecanismo de Mapeamento retorna um mapa em que a ID de EC Externo do primeiro repositório de dados externo (UCMDB , neste caso) é mapeado para os IDs de EC Externo do segundo repositório de dados (externo).
9	Para cada mapeamento, o Mecanismo de Mapeamento cria um relacionamento virtual.
10	Depois do cálculo dos resultados da consulta TQL federada (somente na etapa de topologia), o Federation Framework recupera o layout TQL original para os ECs e relacionamentos resultantes dos repositórios de dados apropriados.

O cálculo começa na extremidade do adaptador externo



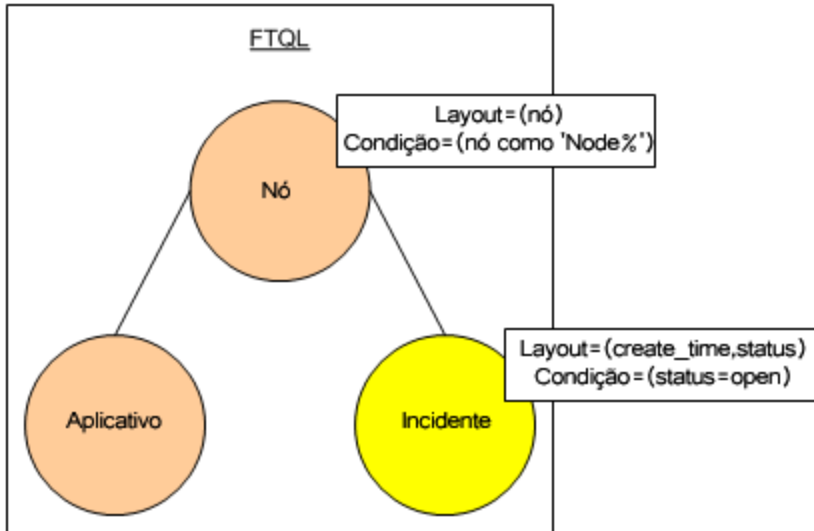
Os números nesta imagem são explicados a seguir:

Número	Explicação
1	O Federation Framework recebe uma chamada para um cálculo de TQL federado.
2	O Federation Framework analisa o adaptador, localiza o relacionamento virtual e divide o TQL original em dois subadaptadores – um para o UCMDB e outro para o repositório de dados externo.
3	O Federation Framework solicita a topologia do sub-TQL do Adaptador Externo. O <code>ExternalTopologyResult</code> retornado não deve conter nenhum objeto de reconciliação, porque os dados de reconciliação não fazem parte da solicitação.
4	Depois de receber os resultados de topologia, o Federation Framework chama o Mecanismo de Mapeamento apropriado com o relacionamento virtual atual e solicita os dados de reconciliação. O parâmetro <code>reconciliationObjects</code> está vazio nessa etapa, ou seja, nenhuma condição é adicionada aos dados de reconciliação

Número	Explicação
	<p>nessa chamada. Os dados de reconciliação retornados definem quais dados são necessários para corresponder os ECs de reconciliação no UCMDB ao repositório de dados externo. Os dados de reconciliação podem ser de um destes três tipos:</p> <ul style="list-style-type: none">• IdReconciliationData. Os ECs são reconciliados de acordo com seu ID.• PropertyReconciliationData. Os ECs são reconciliados de acordo com as propriedades de um dos ECs.• TopologyReconciliationData. Os ECs são reconciliados de acordo com a topologia (por exemplo, para reconciliar ECs de nó, o endereço IP de IP é necessário também).
5	<p>O Federation Framework solicita objetos de reconciliação para os ECs que foram recebidos na etapa 3 do repositório de dados externo. O Federation Framework chama o método getTopologyWithReconciliationData() no Adaptador Externo, em que a topologia solicitada é uma topologia de um nó com ECs recebidos na etapa 3 como sendo os dados de reconciliação e condição de ID da etapa 4.</p>
6	<p>O Federation Framework chama o Mecanismo de Mapeamento para recuperar os dados de reconciliação. Nesse estado (em contraste com a etapa 3), o Mecanismo de Mapeamento recebe os objetos de reconciliação da etapa 5 como parâmetros. O Mecanismo de Mapeamento traduz o objeto de reconciliação recebido para a condição nos dados de reconciliação.</p>
7	<p>O Federation Framework solicita a topologia do sub-TQL com dados de reconciliação da etapa 6 do UCMDB.</p>
8	<p>O Federation Framework chama o Mecanismo de Mapeamento para se conectar entre os resultados recebidos. O parâmetro <code>firstResult</code> é o resultado de topologia externa recebido do Adaptador Externo na etapa 5 e o parâmetro <code>secondResult</code> é o resultado de topologia externa do UCMDB na etapa 7. O Mecanismo de Mapeamento retorna um mapa em que a ID de EC Externo do primeiro repositório de dados (o repositório de dados externo, nesse caso) é mapeado para os IDs de EC Externo do segundo repositório de dados (UCMDB).</p>
9	<p>Para cada mapeamento, o Mecanismo de Mapeamento cria um relacionamento virtual.</p>
10	<p>Depois do cálculo dos resultados da consulta TQL federada (somente na etapa de topologia), o Federation Framework recupera o layout TQL original para os ECs e relacionamentos resultantes dos repositórios de dados apropriados.</p>

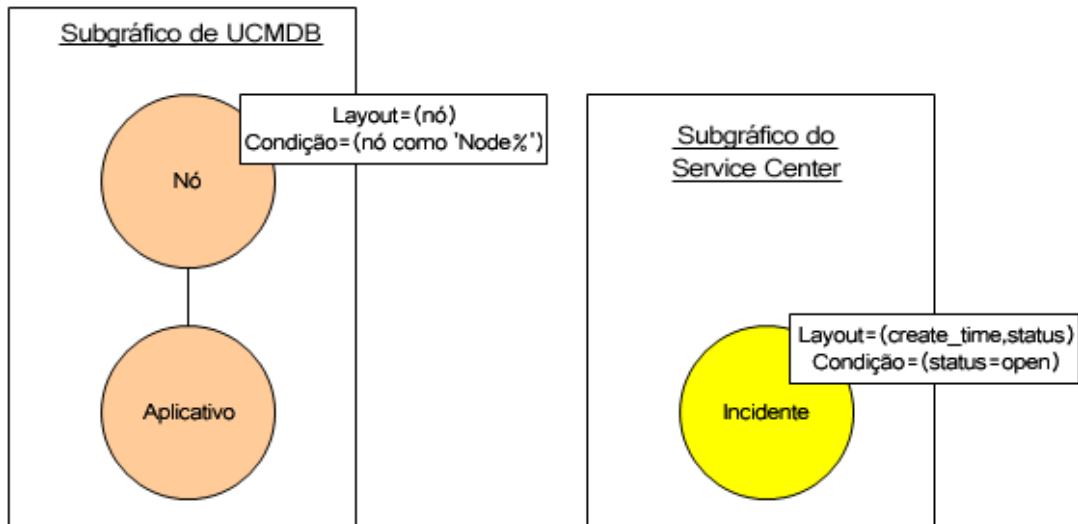
Exemplo do fluxo do Federation Framework para consultas TQL federadas

Este exemplo explica como visualizar todos os incidentes abertos em nós específicos. O repositório de dados do ServiceCenter é o repositório de dados externo. As instâncias de nó são armazenadas no UCMDB, enquanto as instâncias de incidente são armazenadas no ServiceCenter. Pressupõe-se que, para conectar as instâncias de incidente ao nó apropriado, as propriedades `node` e `ip_address` do host e do IP são necessárias. Essas são propriedades de reconciliação que identificam os nós do ServiceCenter no UCMDB.

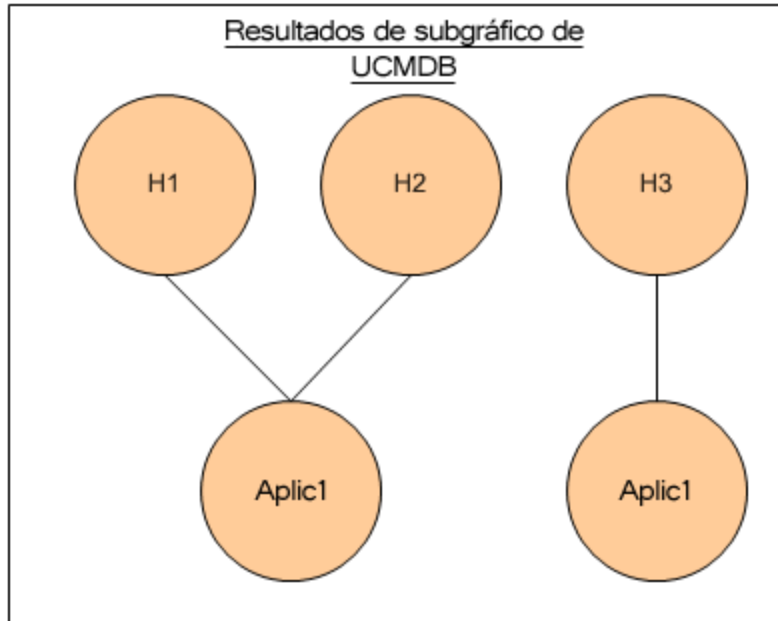


Observação: Para federação de atributos, o método `getTopology` do adaptador é chamado. Os dados de reconciliação são adaptados no TQL do usuário (nesse caso, o elemento de EC).

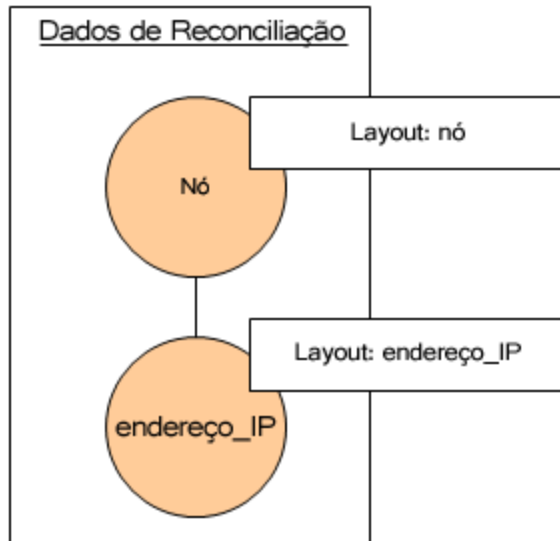
1. Depois de analisar o adaptador, o Federation Framework reconhece o relacionamento virtual entre `Node` e `Incident` e divide a consulta TQL federada em dois subgráficos.



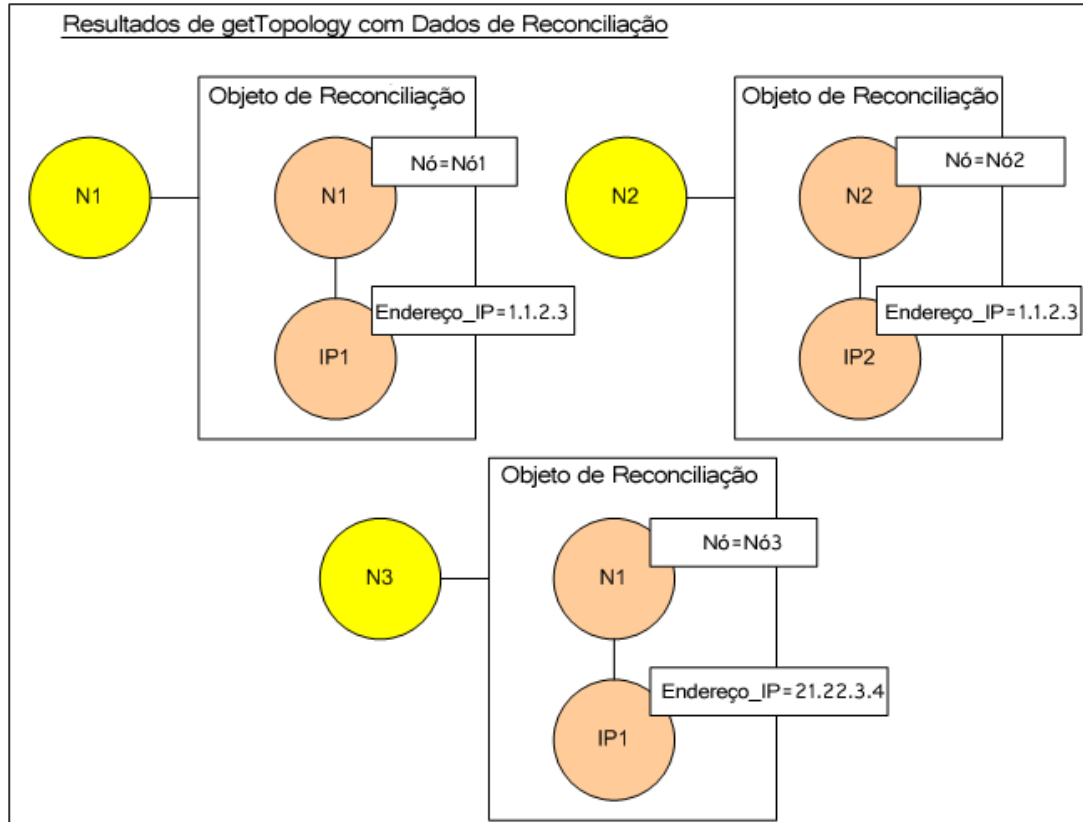
2. O Federation Framework executa o subgráfico do UCMDB para solicitar a topologia e recebe os seguintes resultados:



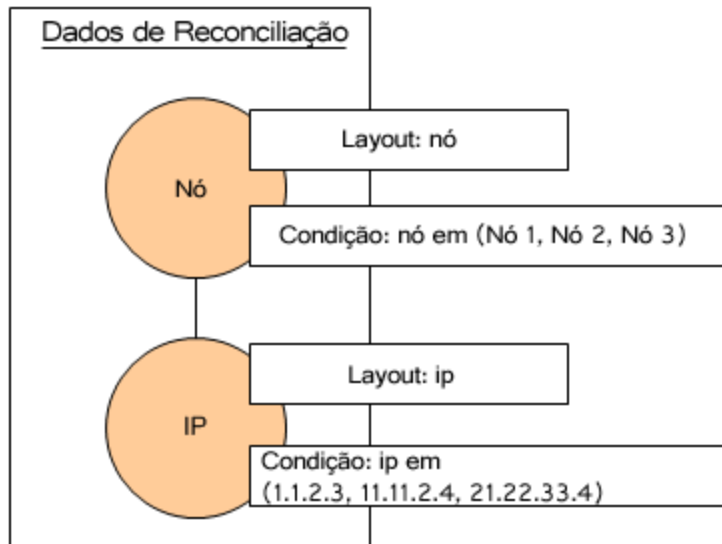
3. O Federation Framework solicita, no Mecanismo de Mapeamento, os dados de reconciliação para o primeiro repositório de dados (UCMDB) que contém as informações para se conectar entre os dados recebidos dos dois repositórios de dados. Os dados de reconciliação nesse caso são:



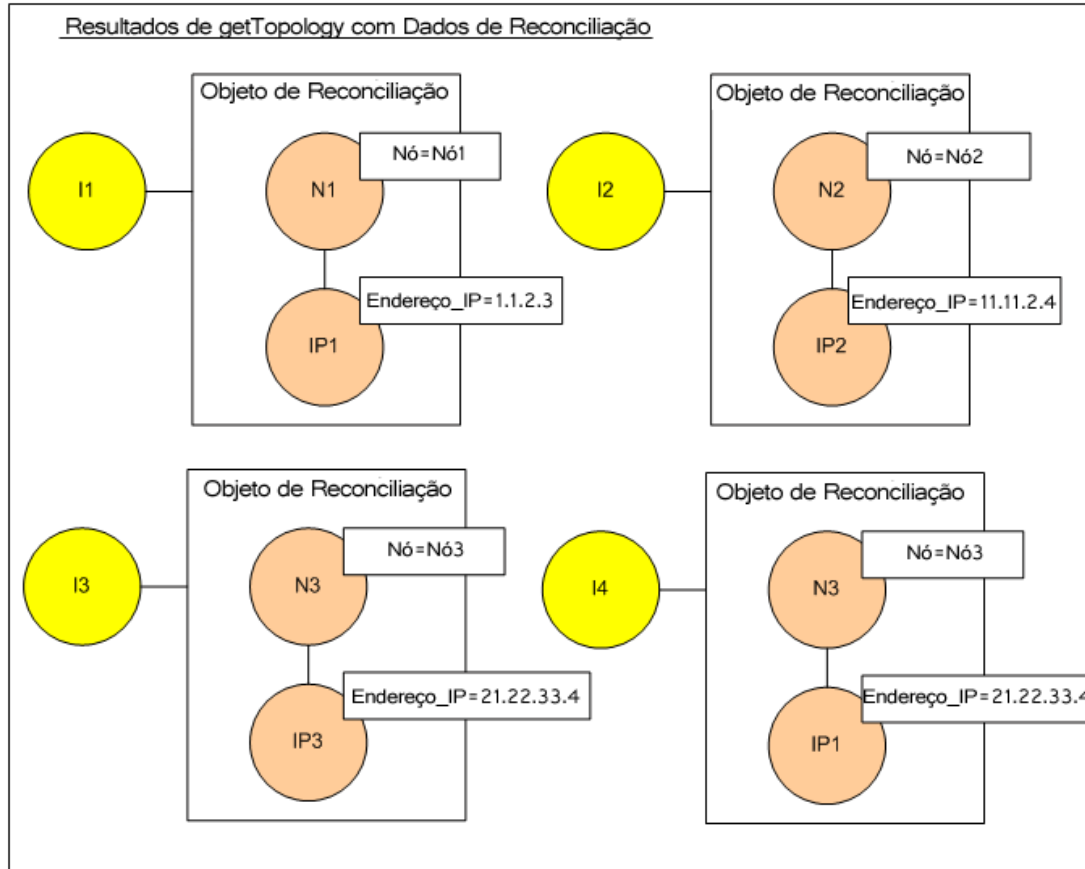
4. O Federation Framework cria uma consulta de topologia de um nó com as condições de Nó e ID nela do resultado anterior (`node` em H1, H2, H3) e executa essa consulta com os dados de reconciliação necessários no UCMDB. O resultado inclui ECs de Nó que são relevantes para a condição de ID e o objeto de reconciliação apropriado para cada EC:



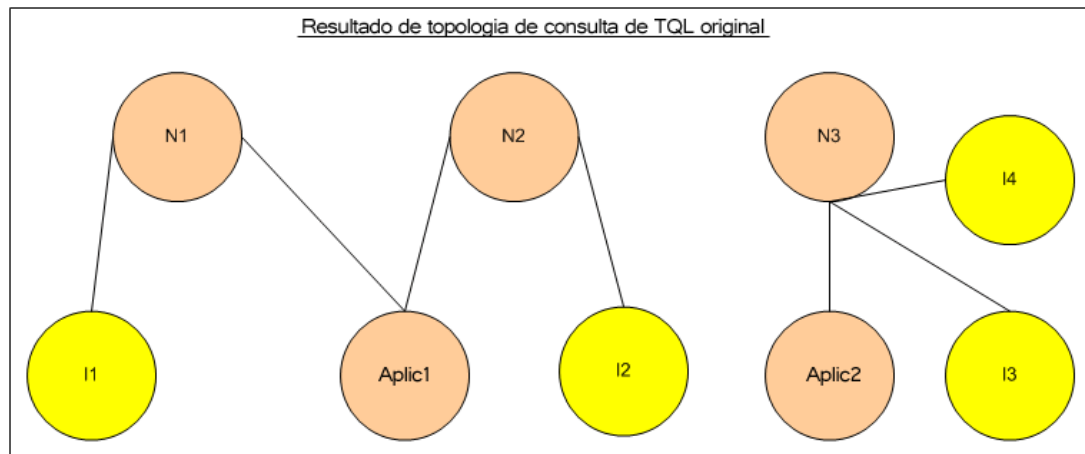
5. Os dados de reconciliação para o ServiceCenter devem conter uma condição para `node` e `ip` que é derivada dos objetos de reconciliação recebidos do UCMDB:



6. O Federation Framework executa o subgráfico do ServiceCenter com os dados de reconciliação para solicitar a topologia e os objetos de reconciliação apropriados e recebe os seguintes resultados:



7. O resultado após a conexão no Mecanismo de Mapeamento e a criação de relacionamentos virtuais é:



8. O Federation Framework solicita o layout de TQL original para instâncias recebidas do UCMDB e ServiceCenter.

Fluxo do Federation Framework para população

Esta seção inclui os seguintes tópicos:

- "Definições e termos" abaixo
- "Diagrama de fluxo" abaixo

Definições e termos

Assinatura. Indica o estado das propriedades no EC. Se alterações forem feitas nos valores de propriedade em um EC, também será necessário alterar a assinatura de EC. A assinatura de EC ajuda a detectar se um EC foi alterado sem recuperar e comparar todas as propriedades de EC. Tanto o EC quanto a assinatura de EC são fornecidos pelo adaptador apropriado. O adaptador é responsável por alterar a assinatura de EC quando as propriedades de EC são alteradas.

Diagrama de fluxo

O diagrama de sequência a seguir ilustra a interação entre o Federation Framework e os adaptadores de origem e de destino em um fluxo de população:



1. O Federation Framework recebe a topologia do resultado da consulta proveniente do adaptador de origem. O adaptador reconhece a consulta por seu nome e a executa no repositório de dados externo. O resultado de topologia contém a ID e a assinatura para cada EC e relacionamento no resultado. A ID é a ID lógica que define o EC como exclusivo no repositório de dados externo. A assinatura deverá ser modificada se o EC ou relacionamento for modificado.
2. O Federation Framework usa assinaturas para comparar os resultados de consulta de topologia recém-recebidos com os resultados salvos, e para determinar quais ECs foram alterados.
3. Depois que o Federation Framework localiza os ECs e relacionamentos que foram alterados, ele chamará o adaptador de origem com os IDs dos ECs e relacionamentos alterados como um parâmetro para recuperar o layout completo.
4. O Federation Framework envia a atualização ao adaptador de destino. O adaptador de destino atualiza a origem de dados externos com os dados recebidos.
5. Após a atualização, o Federation Framework salva o último resultado da consulta.

Interfaces de adaptador

Esta seção inclui os seguintes tópicos:

- ["Definições e termos" abaixo](#)
- ["Interfaces de adaptador para consultas TQL federadas" abaixo](#)

Definições e termos

Relacionamento externo. O relacionamento entre dois tipos de EC externos aceitos pelo mesmo adaptador.

Interfaces de adaptador para consultas TQL federadas

Use a interface de adaptador apropriada para cada adaptador, da seguinte maneira.

- Uma **one Node topology interface** é usada quando o adaptador não oferece suporte a nenhum relacionamento externos, ou seja, o adaptador nunca está para receber uma solicitação com mais de um EC externo. Todas as OneNode são criadas para simplificar fluxo de trabalho; para esses casos em que é necessário usar uma consulta mais extensiva, use a interface `DataAdapter`.
- Uma **interface DataAdapter** é usada para definir adaptadores que oferecem suporte a consultas federadas complexas. A solicitação de reconciliação nesses adaptadores faz parte do único parâmetro `QueryDefinition`. Esses adaptadores também podem ser usados para População.

Interfaces OneNode

As interfaces a seguir têm diferentes tipos de dados de reconciliação:

- **OneNodeTopologyIdReconciliationDataAdapter.** Use se o adaptador oferecer suporte a um **TQL de nó único** e a reconciliação entre repositórios de dados for calculada pelo ID.
- **OneNodeTopologyPropertyReconciliationDataAdapter.** Use se o adaptador oferecer suporte a um **TQL de nó único** e a reconciliação entre repositórios de dados for executada pelas propriedades de um EC.
- **OneNodeTopologyDataAdapter.** Use se o adaptador oferecer suporte a um **TQL de nó único** e a reconciliação entre repositórios de dados for executada pela topologia.

Interfaces de adaptador de dados

- **DataAdapter.** Use este adaptador para oferecer suporte a consultas TQL federadas complexas. Permite a maior diversidade.
- **PopulateDataAdapter.** Use este adaptador para oferecer suporte a consultas TQL federadas complexas e fluxos de população. Em um fluxo de população, este adaptador recupera todo o conjunto de dados, além de deixar no filtro de sonda a diferença desde a última execução do trabalho.
- **PopulateChangesDataAdapter.** Use este adaptador para oferecer suporte a consultas TQL federadas complexas e fluxos de população. Em um fluxo de população, este adaptador oferece

suporte à recuperação apenas das alterações que ocorreram desde a última execução do trabalho.

Observação: Ao desenvolver um adaptador que pode retornar grandes conjuntos de dados, é importante permitir o agrupamento implementando a interface `ChunkGetter`. Para obter mais informações, consulte o documento Java do adaptador específico.

Interfaces adicionais

- **SortResultDataAdapter**. Use se for possível classificar os ECs resultantes no repositório de dados externo.
- **FunctionalLayoutDataAdapter**. Use se for possível calcular o layout funcional no repositório de dados externo.

Interfaces de adaptador para sincronização

- **SourceDataAdapter**. Use para adaptadores de origem nos fluxos de população.
- **TargetDataAdapter**. Use para adaptadores de destino nos fluxos de push de dados.

Depurar recursos do adaptador

Esta tarefa descreve como usar o console JMX para criar, exibir e excluir recursos de estado do adaptador (qualquer recurso criado usando os métodos de manipulação de recursos na interface `DataAdapterEnvironment`, salvos no banco de dados do UCMDB ou da Sonda) para fins de depuração e desenvolvimento.

1. Inicie o navegador da Web e insira o endereço do servidor, da seguinte maneira:
 - Para o servidor UCMDB: `http://localhost:8080/jmx-console`
 - Para a Sonda: `http://localhost:1977`

Você pode precisar fazer logon com um nome de usuário e senha (os valores padrão são `sysadmin/sysadmin`).

2. Para abrir a página Visualização do JMX MBEAN, realize uma das seguintes opções:
 - No servidor UCMDB: clique em **UCMDB:service=FCMDB Adapter State Resource Services**
 - Na Sonda: clique em **type=AdapterStateResources**
3. Insira valores nas operações que deseja usar e clique em **Invoke**.

Adicionar um adaptador para uma nova fonte de dados externos

Esta tarefa explica como definir um adaptador para oferecer suporte a uma nova fonte de dados externos.

Esta tarefa inclui as seguintes etapas:

- "Pré-requisitos" abaixo
- "Definir relacionamentos válidos para relacionamentos virtuais" abaixo
- "Definir uma configuração de adaptador" na página seguinte
- "Definir classes aceitas" na página 176
- "Implementar o adaptador" na página 177
- "Definir as regras de reconciliação ou implementar o mecanismo de mapeamento" na página 177
- "Adicionar JARs necessários para implementação no caminho de classe" na página 177
- "Implantar o adaptador" na página 178
- "Atualizar o adaptador" na página 178

1. Pré-requisitos

Classes de adaptador aceitas pelo modelo para ECs e relacionamentos no Modelo de Dados do UCMDB. Como desenvolvedor de adaptadores, você deve:

- ter conhecimento da hierarquia dos tipos de EC do UCMDB para entender como os TECs externos estão relacionados aos TECs do UCMDB
- modelar os TECs externos no modelo de classe do UCMDB
- adicionar as definições para novos tipos de EC e seus relacionamentos
- definir relacionamentos válidos no modelo de classe do UCMDB para os relacionamentos válidos entre as classes internas do adaptador. (Os TECs podem ser colocados em qualquer nível da árvore do modelo de classe do UCMDB.)

A modelagem deve ser a mesma independentemente do tipo de federação (em tempo real ou replicação). Para ver detalhes sobre como adicionar novas definições de TEC ao modelo de classe do UCMDB, consulte "[Trabalhando com o Seletor de EC](#)" na página 1 no Guia de Modelagem do HP Universal CMDB.

Para que o adaptador ofereça suporte a atributos federados em TECs, adicione esse TEC às classes aceitas com atributos aceitos e à regra de reconciliação para esse TEC.

2. Definir relacionamentos válidos para relacionamentos virtuais

Observação: Esta seção é relevante apenas para federação.

Para recuperar TECs federados que estão conectados aos TECs do CMDB locais, é necessário haver uma definição de link válida entre os dois TECs no CMDB.


- a. Crie um arquivo XML de links válido que contenha esses links (se ainda não existirem).
- b. Adicione o arquivo XML de links ao pacote de adaptadores na pasta `\validlinks`. Para obter detalhes, consulte "[Gerenciador de Pacotes](#)" na página 1 no *Guia de Administração do HP Universal CMDB*.

Exemplo de uma definição de relacionamento válido:

No exemplo a seguir, o relacionamento de tipo `containment` entre as instâncias de tipo `node` com as instâncias de tipo `myclass1` é uma definição de relacionamento válido.

```
<Valid-Links>
  <Valid-Link>
    <Class-Ref class-name="containment">
      <End1 class-name="node">
        <End2 class-name="myclass1">
          <Valid-Link-Qualifiers>
        </Valid-Link-Qualifiers>
      </End2>
    </Class-Ref>
  </Valid-Link>
</Valid-Links>
```

3. Definir uma configurações de adaptador

- a. Vá até **Gerenciamento do Adaptador**.
- b. Clique no botão **Criar novo recurso** .
- c. Na caixa de diálogo Novo adaptador, selecione **Integração e Adaptador Java**.
- d. Clique com o botão direito do mouse no adaptador que você criou e selecione **Editar Origem do Adaptador** no menu de atalho.
- e. Edite as seguintes marcas XML:

```
<pattern xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
id="newAdapterIdName"
xsi:noNamespaceSchemaLocation="../../../Patterns.xsd"
description="Adapter Description" schemaVersion="9.0"
displayName="New Adapter Display Name">

<deletable>true</deletable>

<discoveredClasses>

<discoveredClass>link</discoveredClass>

<discoveredClass>object</discoveredClass>

</discoveredClasses>

<taskInfo
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.
AdapterService">

<params
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.
AdapterServiceParams" enableAging="true"
enableDebugging="false" enableRecording=
"false" autoDeleteOnErrors="success" recordResult="false"
maxThreads="1" patternType="java_adapter"
maxThreadRuntime="25200000">
```

```
<className>com.yourCompany.adapter.MyAdapter.MyAdapterClass
</className>

</params>

<destinationInfo
className="com.hp.ucmdb.discovery.probe.tasks.BaseDestinationData">

<!-- check -->

<destinationData name="adapterId"
description="">${ADAPTER.adapter_id}</destinationData>

<destinationData name="attributeValues"
description="">${SOURCE.attribute_values}</destinationData>

<destinationData name="credentialsId"
description="">${SOURCE.credentials_id}</destinationData>

<destinationData name="destinationId"
description="">${SOURCE.destination_id}</destinationData>

</destinationInfo>

<resultMechanism isEnabled="true">

<autoDeleteCITs isEnabled="true">

<CIT>link</CIT>

<CIT>object</CIT>

</autoDeleteCITs>

</resultMechanism>

</taskInfo>

<adapterInfo>

<adapter-capabilities>

<support-federated-query>

<!--<classes com suporte/> <!--consulte a seção sobre classes com
suporte-->

<topologia>

<pattern-topology /> <!--ou <one-node-topology> -->

</topologia>

</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 />
<parameters>
<!--O atributo de descrição pode ser escrito em texto simples ou
em HTML.-->
<!--O atributo de host é tratado como um caso especial pelo
UCMDB-->
<!--e selecionará automaticamente o nome da sonda (se possível)-
->
<!--de acordo com o valor do atributo.-->
<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>
```

Para ver detalhes sobre as marcas XML, consulte "[Marcas e propriedades de configuração XML](#)" na página 181.

4. Definir classes aceitas

Defina as classes aceitas do código do adaptador implementando o método *getSupportedClasses()* ou usando o arquivo XML de padrão.

```
<supported-classes>
  <supported-class name="HistoryChange" is-derived="false" is-
reconciliation-supported="false" federation-not-supported="false"
is-id-reconciliation-supported="false">
    <supported-conditions>
      <attribute-operators attribute-name="change_create_time">
        <operator>GREATER</operator>
        <operator>LESS</operator>
        <operator>GREATER_OR_EQUAL</operator>
        <operator>LESS_OR_EQUAL</operator>
        <operator>CHANGED_DURING</operator>
      </attribute-operators>
    </supported-conditions>
  </supported-class>
```

name	O nome do tipo de EC
is-derived	Especifica se essa definição inclui todos os filhos herdeiros
is-	Especifica se essa classe é usada para reconciliação

reconciliation-supported	
is-id-reconciliation-supported	Especifica se essa classe é usada para id-reconciliation
federation-not-supported	Especifica se esse TEC não deve ser permitido para federação (bloqueando determinados TECs, por exemplo, um TEC definido exclusivamente para federação)
<supported-conditions>	Especifica as condições aceitas em cada atributo

5. Implementar o adaptador

Selecione a classe de implementação de adaptadores correta de acordo com seus recursos definidos. A classe de implementação de adaptadores implementa as interfaces apropriadas de acordo com os recursos definidos.

O suporte à reconciliação do adaptador pode ser definido de acordo com a **global_id**, sendo que, nesse caso, **global_id** deve ser definida como parte dos atributos de reconciliação nas classes suportadas do adaptador. Se o suporte à reconciliação do adaptador for definido de acordo com **global_id**, **getTopologyWithReconciliationData()** deverá retornar **global_id** como parte das propriedades do objeto de reconciliação. O UCMDB usa **global_id** para a reconciliação de resultados federados para um TEC, em vez da regra de identificação.

6. Definir as regras de reconciliação ou implementar o mecanismo de mapeamento

Se o adaptador oferecer suporte a consultas TQL federadas, você terá três opções para definir seu Mecanismo de Mapeamento:

- Use o mecanismo de mapeamento padrão do CMDB 9.0x, que usa as regras de reconciliação interna para mapeamento do CMDB. Para usá-lo, deixe a marca XML **<default-mapping-engine/>** vazia.
Para obter detalhes, consulte "[O arquivo reconciliation_types.txt](#)" na página 132.
- Use o mecanismo de mapeamento do CMDB 8.0x. Para fazer isso, use a seguinte Marca XML: **<default-mapping-engine>com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine</default-mapping-engine>**
Para obter detalhes, consulte "[O arquivo reconciliation_rules.txt \(para compatibilidade com versões anteriores\)](#)" na página 132.
- Escreva seu próprio mecanismo de mapeamento implementando a interface do mecanismo de mapeamento e colocando o JAR com o restante do código do adaptador. Para fazer isso, use a seguinte marca XML: **<default-mapping-engine>com.yourcompany.map.MyMappingEngine</default-mapping-engine>**

7. Adicionar JARs necessários para implementação no caminho de classe

Para implementar suas classes, adicione o arquivo **federation_api.jar** ao caminho de classe do editor de código.

8. Implantar o adaptador

Implante o pacote de adaptadores. Para ver detalhes gerais sobre como implantar um pacote, consulte "[Gerenciador de Pacotes](#)" na página 1 no *Guia de Administração do HP Universal CMDB*.

O pacote deve conter as seguintes entidades:

- Nova definição de TEC (opcional):
- Usada somente se o adaptador oferecer suporte a novos tipos de EC que ainda não existem no UCMDB.
- As novas definições de TEC estão localizadas na pasta `class` do pacote.
- Nova definição de tipo de dados (opcional):
- Usada somente se os novos TECs exigirem novos tipos de dados.
- As novas definições de tipo de dados estão localizadas na pasta `typedef` do pacote.
- Nova definição de relacionamentos válidos (opcional):
- Usada somente se o adaptador oferecer suporte a consultas TQL federadas.
- As novas definições de relacionamentos válidos estão localizadas na pasta `validlinks` do pacote.
- O arquivo XML de configuração de padrões deve estar localizado na pasta `discoveryPatterns` do pacote.
- **Descriptor.** Determina as definições do pacote.
- Coloque suas classes compiladas (normalmente um arquivo jar) no pacote na pasta `adapterCode\<id_do_adaptador>`.

Observação: O nome da pasta `id_do_adaptador` tem o mesmo valor da configuração do adaptador.

- Se você criar seu próprio arquivo de configuração, deverá colocá-lo no pacote na pasta `adapterCode\<id_do_adaptador>`.

9. Atualizar o adaptador

Alterações em qualquer um dos arquivos não-binários do adaptador podem ser feitas no módulo Gerenciamento do Adaptador. Fazer alterações em arquivos de configuração no módulo Gerenciamento do Adaptador faz com que o adaptador seja recarregado com as novas configurações.

As atualizações também podem ser feitas editando os arquivos no pacote (tanto arquivos binários quanto não-binários) e reimplantando o pacote usando o Gerenciador de Pacotes. Para ver detalhes, consulte "Implantar um pacote" no *Guia de Administração do HP Universal CMDB*.

Implementar o mecanismo de mapeamento

A configuração do mecanismo de mapeamento depende de qual mecanismo de mapeamento você está usando.

Esta tarefa inclui as seguintes etapas:

- "Configure o arquivo `reconciliation_types.txt` (para o mecanismo de mapeamento padrão do UCMDB 9.0x)" abaixo
- "Configure o arquivo `reconciliation_rules.txt` (para o mecanismo de mapeamento do UCMDB 8.0x)" abaixo

1. Configure o arquivo `reconciliation_types.txt` (para o mecanismo de mapeamento padrão do UCMDB 9.0x)

O arquivo é usado para definir quais tipos de EC são usados para reconciliação no adaptador.

Escreva cada tipo de EC usado para reconciliação em uma única linha, da seguinte maneira:

```
node business_application
```

Coloque o arquivo no pacote de adaptadores na pasta `adapterCode\<ID_do_adaptador>\META-INF\`. Para suportar a reconciliação da ID (reconciliação baseada no mapeamento da ID entre a ID do CMDB no CMDB para um valor no banco de dados remoto), você deve mapear um atributo especial do CMDB chamado `cmdb_id` para uma coluna no banco de dados da cadeia (char, varchar) ou tipo `byte[]` (raw/bytes).

2. Configure o arquivo `reconciliation_rules.txt` (para o mecanismo de mapeamento do UCMDB 8.0x)

Este arquivo é usado para configurar as regras de reconciliação. Cada linha no arquivo representa uma regra. Por exemplo:

```
reconciliation_type[node] expression[^node.name OR ip_address.name]  
end1_type[node] end2_type[ip_address] link_type[containment]
```

O parâmetro **reconciliation_type** é preenchido com o tipo de EC no qual a reconciliação é executada (o nome de classe UCMDB que está conectado à classe federada no TQL).

O parâmetro **expression** é a lógica que decide se dois objetos de reconciliação são iguais (um objeto de reconciliação do lado do UCMDB e o outro do lado do adaptador Federado).

A expressão consiste em ORs e ANDs.

A convenção em relação a nomes de atributos na parte da expressão é **[nome_de_classe].[nome_de_atributo]**. Por exemplo, o atributo `ip_address` na classe `ip` é escrito `ip.ip_address`.

Você pode definir correspondências ordenadas. A correspondência ordenada verifica a primeira subexpressão OR. Se dois objetos de reconciliação tiverem o valor nos atributos da subexpressão e for retornado falso (os objetos de reconciliação não são iguais), isso significará que a segunda subexpressão OR não é comparada.

Para ver uma correspondência ordenada, use **ordered expression** em vez de **expression**.

O sinal se circunflexo (^) é usado para não diferenciar maiúsculas de minúsculas durante as comparações.

Os outros parâmetros (**end1_type**, **end2_type** e **link_type**) são usados somente se os dados de reconciliação contêm dois nós e não só o nó do tipo de reconciliação (os dados de reconciliação de topologia). Nesse caso, os dados de reconciliação são **end1_type** **-(link_type)>** **end2_type**.

Não é necessário adicionar o layout relevante porque ele é recuperado da expressão.

Para executar a reconciliação por ID do UCMDb, use **cmdb_id** como nome do atributo na expressão.

Coloque o arquivo no pacote de adaptadores na pasta **adapterCode\<ID_do_adaptador>\META-INF**.

Exemplos:

- Você pode adicionar uma regra de reconciliação somente para um TEC de nó. Isso ocorre porque somente TECs de nó têm relacionamentos válidos com TECs externos. Por exemplo, um EC de nó no CMDB tem correspondência com um EC de nó no ServiceCenter através do atributo `node.name` ou do atributo `ip_address.name`.
- A regra de reconciliação nesse caso é uma regra de topologia e a expressão é ordenada. A regra executa as seguintes verificações nos ECs em comparação:
 - Se o atributo `node.name` for igual, a regra fará correspondência dos nós.
 - Se o atributo `node.name` não for igual, a regra não fará correspondência dos nós.
 - Se o atributo `node.name` for nulo em um dos ECs comparados, a regra verificará o atributo `ip_address.name`. Se o atributo `ip_address.name` for igual, a regra fará correspondência dos nós.

Criar um adaptador de amostra

Este exemplo ilustra como criar um adaptador de exemplo. Esta tarefa inclui as seguintes etapas:

- "Selecionar lógica do adaptador" abaixo
- "Carregar o projeto" na página seguinte

1. Selecionar lógica do adaptador

Ao implementar um adaptador, você precisa escolher como manipular a lógica de condição na implementação (condições de propriedade, condições de ID, condições de reconciliação e condições de link).

- a. Recupere todos os dados na memória do adaptador e deixe-o selecionar ou filtrar as Instâncias de EC necessárias.
- b. Converta todas as condições na linguagem de fonte de dados e deixe-a filtrar e selecionar os dados. Por exemplo:
 - Converta a condição em uma consulta SQL.
 - Converta a condição em um objeto de filtro de API Java.

- c. Filtre alguns dos dados no serviço remoto e faça com que o adaptador selecione e filtre o restante.

No exemplo MyAdapter, a lógica na etapa a é usada.

2. Carregar o projeto

Copie os arquivos de **C:\hp\UCMDB\UCMDBServer\tools\adapter-dev-kit\SampleAdapters** e siga as instruções nos arquivos leia-me.

Observação: Se você usar um adaptador com conjuntos de dados extensos, talvez precisar usar armazenamento em cache e indexação para melhorar o desempenho para Federação.

A documentação de javadocs online está disponível em:

C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DBAdapterFramework_JavaAPI\index.html

Marcas e propriedades de configuração XML

<code>id="newAdapterIdName"</code>	Define o nome real do adaptador. É usado para logs e pesquisas de pasta.
<code>displayName="New Adapter Display Name"</code>	Define o nome de exibição do adaptador, conforme aparece na UI.
<code><className>...</className></code>	Define a interface do adaptador que implementa a classe Java.
<code><category >My Category</category></code>	Define a categoria do adaptador.
<code><parameters></code>	Define as propriedades para configuração disponíveis na UI ao configurar um novo ponto de integração.
<code>name</code>	O nome da propriedade (usado principalmente pelo código).
<code>description</code>	A dica de exibição da propriedade.
<code>type</code>	Cadeia de caracteres ou inteiro (use valores válidos com cadeia de caracteres para booleano).
<code>display-name</code>	O nome da propriedade na UI.
<code>mandatory</code>	Especifica se essa propriedade de configuração é obrigatória para o usuário.
<code>order-index</code>	A ordem de colocação da propriedade (small = up).
<code>valid-values</code>	Uma lista de possíveis valores válidos

		separados por caracteres ‘;’ (por exemplo, valid-values="Oracle;SQLServer;MySQL" or valid-values="True;False").
<adapterInfo>		Contém a definição das configurações estáticas e recursos do adaptador.
	<support-federated-query>	Define esse adaptador como capaz de federação.
	<one-node-topology>	A capacidade de consultas federadas com um nó de consulta federado.
	<pattern-topology>	A capacidade de federar consultas complexas.
	<support-replication-data>	Define o recurso para executar fluxos de push de dados e de população.
	<source>	Esse adaptador também pode ser usado para fluxos de população.
	<push-back-ids>	Devolva a ID global do EC à coluna global_id da tabela (deve ser definida no orm.xml). O comportamento pode ser substituído pela implementação do plug-in FcmdbPluginPushBackIds .
	<changes-source>	Esse adaptador pode ser usado para fluxos de alterações de população.
	<target>	Esse adaptador pode ser usado para fluxos de push de dados.
	<default-mapping-engine>	Permite a definição de um mecanismo de mapeamento para o adaptador (por padrão, o adaptador usa o mecanismo de mapeamento padrão). Para qualquer outro mecanismo de mapeamento, insira o nome de classe implementador do mecanismo de mapeamento (para o mecanismo de mapeamento do UCMDB 8.0x, use: com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine)
	<removedAttributes>	Obriga a remoção de atributos específicos do resultado.
	<full-population-days-interval>	Especifica quando executar um trabalho de população completo em vez de um trabalho diferencial (a cada ‘x’ dias). Usa o mecanismo de envelhecimento juntamente com o fluxo de alterações.

	<code><adapter-settings></code>	A lista de configurações do adaptador.
	<code><list.attributes.for.s- et></code>	Determina quais atributos substituem o valor anterior (se existir).

Capítulo 6

Desenvolvimento de adaptadores push

Este capítulo inclui:

Visão geral sobre o desenvolvimento de adaptadores push	184
Sincronização diferencial	184
Preparar os arquivos de mapeamento	185
Escrever scripts Jython	187
Suporte à sincronização diferencial	190
Criar um pacote de adaptador	192
Esquema do arquivo de mapeamento	193
Esquema de resultados de mapeamento	202

Visão geral sobre o desenvolvimento de adaptadores push

O Adaptador Push Genérico fornece uma plataforma que permite desenvolver rapidamente integrações que enviam por push dados do UCMDB 9.0x para repositórios de dados externos (bancos de dados e aplicativos de terceiros). O desenvolvimento de uma integração personalizada baseada no Adaptador Push Genérico exige:

- Um arquivo XML para o mapeamento entre os tipos de vínculo de EC do UCMDB e itens de dados externos.
- Um script Jython para enviar por push itens de dados ao repositório de dados externo.

Sincronização diferencial

Para que o adaptador push permita a sincronização diferencial, a função **DiscoveryMain** deve retornar um objeto que implemente a interface **DataPushResults**, que contém os mapeamentos entre as IDs que o script Jython recebe do arquivo XML e as IDs que o script Jython cria na máquina remota. As últimas IDs são do tipo **ExternalId**.

O comando **ExternalIdUtil.restoreExternal**, que recebe a ID do EC no CMDB como um parâmetro, restaura a ID externa a partir da ID do EC no CMDB. Esse comando pode ser usado, por exemplo, ao realizar uma sincronização diferencial e um link é recebido onde uma de suas extremidades não está na massa (já foi sincronizada).

Se o método **DiscoveryMain** no script Jython no qual o adaptador push foi baseado retornar uma instância de **ObjectStateHolderVector** vazia, o adaptador não dá suporte à sincronização diferencial. Isso significa que mesmo que um trabalho de sincronização diferencial seja executado, na realidade uma sincronização completa será realizada. Portanto, nenhum dado poderá ser

atualizado ou removido do sistema remoto, uma vez que todos os dados serão adicionados ao CMDB durante cada sincronização.

Preparar os arquivos de mapeamento

Observação: Você pode recuperar todos os ECs e relacionamentos como eles são no CMDB sem mapeamento, não criando o arquivo **mappings.xml**. Isso retornará todos os ECs e relacionamentos com todos os seus atributos.

Há dois modos diferentes de preparar arquivos de mapeamento:

- Você pode preparar um arquivo de mapeamento único e global.
Todos os mapeamentos são colocados em um único arquivo nomeado **mappings.xml**.
- Você pode preparar um arquivo separado para cada consulta push.
Esses arquivos de mapeamento são nomeados com o formato **<nome da consulta>.xml**.

Para obter detalhes, consulte "Esquema do arquivo de mapeamento" na página 193.

Esta tarefa inclui as seguintes etapas:

- ["Criar arquivo de mapeamento" abaixo](#)
- ["Mapear ECs" abaixo](#)
- ["Mapear vínculos " na página seguinte](#)

1. Criar arquivo de mapeamento

A estrutura do arquivo de mapeamento é criada como a seguir:

```
<?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. Mapear ECs

Há duas maneiras de mapear tipos de EC do CMDB:

- Mapear um tipo de EC para que ECs desse tipo e todos os tipos herdados sejam mapeados da mesma forma:

```
<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>
```

- Mapear um tipo de EC para que somente ECs desse tipo sejam processados. ECs de tipos herdados não serão processados, a menos que seu tipo também seja mapeado (de uma das suas maneiras a seguir):

```
<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>
```

Um tipo de EC mapeado indiretamente (um de seus ancestrais é mapeado usando **source_ci_type_tree**) também pode substituir o mapa de seu pai fazendo com que ele seja exibido em seu próprio **source_ci_type_tree** ou **source_ci_type**.

É recomendável usar **source_ci_type_tree** sempre que possível. Caso contrário, os ECs resultantes de um tipo de EC não exibidos nos arquivos de mapeamento não serão transferidos ao script do Jython.

3. Mapear vínculos

Há duas maneiras de mapear links:

- Mapeie um link que também mapeará todos os tipos de links herdados desse link específico:

```
<source_link_type_tree name="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>
</source_link_type_tree>
```

- Mapeie um link que também mapeará somente esse tipo de link específico, e não os tipos de links herdados dele:

```
<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>
```

Escrever scripts Jython

O script de mapeamento é um script Jython comum, e deve seguir as regras aplicáveis a scripts Jython. Para obter detalhes, consulte ["Desenvolvimento de adaptadores Jython"](#) na página 38.

O script deve conter a função **DiscoveryMain**, que pode retornar uma instância de **OSHVResult** vazia ou uma de **DataPushResults** ao ser bem-sucedido.

Para relatar falhas, o script deve retornar uma exceção, por exemplo:

```
raise Exception('Falha ao inserir no UCMDB remoto usando
TopologyUpdateService. Consulte o log do UCMDB remoto')
```

Na função **DiscoveryMain**, os itens de dados aos quais enviar por push ou que devem ser excluídos do aplicativo externo podem ser obtidos da seguinte forma:

```
# get add/update/delete result objects (in XML format) from the
Framework
addResult = Framework.getTriggerCIData('addResult')
updateResult = Framework.getTriggerCIData('updateResult')
deleteResult = Framework.getTriggerCIData('deleteResult')
```

O objeto-cliente do aplicativo externo pode ser obtido do seguinte modo:

```
oracleClient = Framework.createClient()
```

Esse objeto-cliente usa automaticamente a ID, nome de host e número da porta de credenciais passadas pelo adaptador pela metodologia.

Se você precisar usar os parâmetros de conexão que definiu para o adaptador (para obter detalhes, consulte a etapa ["Edite o arquivo discoveryPatterns\push_adapter.xml."](#) em ["Criar um pacote de adaptador"](#) na página 192), use o seguinte código:

```
propValue = str(Framework.getDestinationAttribute('<Connection
Property Name'))
```

Por exemplo:

```
serverName = Framework.getDestinationAttribute('ip_address')
```

Esta seção inclui também:

- ["Trabalhando com resultados do mapeamento" abaixo](#)
- ["Lidando com a conexão de teste no script" na página 190](#)

Trabalhando com resultados do mapeamento

O Adaptador Push Genérico cria cadeias XML que descrevem os dados a serem adicionados, atualizados ou excluídos do sistema de destino. O script Jython precisa analisar esse XML para, em seguida, realizar a adição, atualização ou exclusão no destino.

No XML da operação de adição que o script Jython recebe, o atributo `mamId` dos objetos e vínculos é sempre o identificador do UCMDb do objeto ou vínculo original antes que seu tipo, atributo ou outra informação tenha sido alterada no esquema do sistema remoto.

No XML das operações de atualização e remoção, o atributo `mamId` de cada objeto ou vínculo contém a representação da cadeia do mesmo `ExternalId` que foi retornado pelo script Jython na sincronização anterior.

No XML, o atributo `id` de um EC armazena `cmdbId` como uma id externa ou `ExternalId` desse EC se o EC tinha um `ExternalId` quando o EC foi enviado ao script. Os campos `end1Id` e `end2Id` do link mantêm para cada uma das extremidades do link `cmdbId` como uma id externa ou `ExternalId` da extremidade desse link se o EC da extremidade do link tinha um `ExternalId` quando ele foi enviado ao script.

Ao processar os ECs no script do Jython, o valor de retorno do script é um mapeamento entre a id do CMDB do EC e a id fornecida (a id fornecida a cada EC do script). Se um CI sofreu push pela primeira vez, a id que está no XML desse EC é a id do CMDB. Se um EC não sofreu push pela primeira vez, a id do EC é a mesma id que foi fornecida a esse EC no script quando ele sofreu push pela primeira vez.

A id é recuperada do script XML do EC como a seguir:

1. A partir do elemento do EC no XML, recupere a id do atributo de id. Por exemplo: `id = objectElement.getAttributeValue('id')`.
2. Após recuperar a id do XML, restaure a id do atributo (cadeia). Por exemplo: `objectId = CmdbObjectID.Factory.restoreObjectID(id)`.
3. Verifique se o `objectId` recebido na etapa anterior é a id do CMDB. Você pode fazer isso verificando se o `objectId` tem a nova id fornecida a ele pelo script. Se tiver, a id retornada não é a ID do CMDB. Por exemplo:
`newId = objectId.getPropertyValue(<o nome do atributo id fornecido pelo script>)`.
Se `newId` for nulo, a id retornada no XML é uma id do CMDB.
4. Se a id for uma id do CMDB (isto é, `newId` for nulo), faça o seguinte (se a id não for uma id do CMDB, vá para a etapa 5):
 - a. Crie uma propriedade para aquele EC com a nova id. Por exemplo: `propArray = [TypesFactory.createProperty('<o nome do atributo id fornecido pelo script>', '<new id>')]`.
 - b. Crie um `externaId` para aquele EC. Por exemplo:
`cmdbId = extI.getPropertyValue('internal_id')`
`className = extI.getType()`
`externalId = ExternalIdFactory.createExternalCiId(className,`

```
propArray)
```

- c. Mapeie a id do CMDB para o recém-criado externalId (e na próxima etapa retorne esse mapeamento ao adaptador). Por exemplo: `objectMappings.put (cmdbId, externalId)`
- d. Quando todos os ECs e links forem mapeados:

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

5. Se a id for a nova id (isto é, `newId` não for nulo), `externalId` é `newId`.

Exemplo do resultado do XML

```
<root>
  <data>
    <objects>
      <Object mode="update_else_insert" name="UCMDB_UNIX"
operation="add" mamId="0c82f591bc3a584121b0b85efd90b174"
id="HiddenRmiDataSource%0Aunix%0A1%0Ainternal_
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%3D41372a1cbcaba27b214b84a2ec9eb535%0A%5D%0Aend2id%
5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A%5D%0AHiddenRmi
DataSource%0Atalk%0A1%0Ainternal_
id%3DSTRING%3D265e985c6ec51a8543f461b30fa58f81%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>
```

Observação: Se `datatype="BYTE"`, o valor do resultado retornado é uma **cadeia** gerada como: nova cadeia([o atributo de matriz de byte]). O objeto `byte[]` pode ser reconstruído por: `<a cadeia recebida>.getBytes()`. Pode haver uma diferença na localidade padrão entre o servidor e a sonda. Nesse caso, a reconstrução está de acordo com a localidade padrão do servidor.

Lidando com a conexão de teste no script

Um script Jython pode ser chamado para testar a conexão a um aplicativo externo. Nesse caso, o atributo `testConnection` do destino retornará `true`. Esse atributo pode ser obtido da metodologia do seguinte modo:

```
testConnection = Framework.getTriggerCIData('testConnection')
```

Quando executado no modo de conexão de teste, um script deve resultar numa exceção quando a conexão ao aplicativo externo não pode ser estabelecida. Do contrário, se a conexão for bem-sucedida, a função **DiscoveryMain** deverá retornar um **OSHVResult** vazio.

Suporte à sincronização diferencial

Importante: Se você estiver implementando a sincronização diferencial em um adaptador existente criado na versão 9.00 ou 9.01, deverá usar o arquivo `push-adapter.zip` da versão 9.02 ou posterior para recriar seu pacote de adaptador. Para obter detalhes, consulte "[Criar um pacote de adaptador](#)" na página 192.

Essa tarefa permite que o adaptador push realize a sincronização diferencial. Para obter detalhes, consulte "[Sincronização diferencial](#)" na página 184.

O script Jython retorna o objeto **DataPushResults** que contém dois mapas Java - um para mapeamentos de ID de objeto (chaves e valores são objetos do tipo `ExternalCiid`) e um para IDs de vínculos (chaves e valores são objetos do tipo `ExternalRelationId`).

- Adicione as seguintes declarações **from** ao seu 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.fcmbd.spi.data.query.types import  
ExternalIdUtil
```

- Use a classe de fábrica **DataPushResultsFactory** para obter o objeto **DataPushResults** da função **DiscoveryMain**.

```
# Criar o objeto UpdateResult  
  
updateResult = DataPushResultsFactory.createDataPushResults  
(objectMappings, linkMappings);
```

- Use os comandos a seguir para criar mapas Java para o objeto **DataPushResults**:

```
# Preparar os mapas para armazenar mapeamento se IDs  
  
objectMappings = HashMap()  
  
linkMappings = HashMap()
```

- Use a classe **ExternalIdFactory** para criar as seguintes IDs ExternalId:
 - ExternalId para objetos ou vínculos originários de um CMDB (por exemplo, todos os ECs em uma operação de adição são do CMDB):

```
externalCIId = ExternalIdFactory.createExternalCmdbCiId(ciType,  
ciIDAsString)  
  
externalRelationId =  
ExternalIdFactory.createExternalCmdbRelationId(linkType,  
end1ExternalCIId, end2ExternalCIId, linkIDAsString)
```

- ExternalId para objetos ou vínculos que não são originários de um CMDB (por exemplo, todas as operações de atualização e remoção contêm tais objetos):

```
myIDField = TypesFactory.createProperty("systemID", "1")  
  
myExternalId = ExternalIdFactory.createExternalCiId(type,  
myIDField)
```

Observação: Se o script Jython atualizou informações existentes e a ID do objeto (ou vínculo) for alterada, você deverá retornar um mapeamento entre a ID externa anterior e a nova.

- Use os métodos **restoreCmdbCiIDString** ou **restoreCmdbRelationIDString** da classe **ExternalIdFactory** para recuperar a cadeia de ID do UCMDB de uma ID externa de um objeto ou vínculo originário do UCMDB.
- Use os métodos **restoreExternalCIId** e **restoreExternalRelationId** da classe **ExternalIdUtil** para restaurar o objeto **ExternalId** do valor do atributo mamId do XML das operações de atualização ou remoção.

Observação: Objetos **ExternalId** são, de fato, uma matriz de propriedades. Isso significa que você pode usar um objeto **ExternalId** para armazenar quaisquer informações de que possa precisar e que identifiquem os dados no sistema remoto.

Criar um pacote de adaptador

1. Extraia o conteúdo do arquivo **C:\hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip** em uma pasta temporária. No pacote do adaptador, o arquivo **sql_queries** localizado em **adapterCode > PushAdapter > sqlTablesCreation**, contém as consultas necessárias para criar tabelas em um novo esquema no Oracle para testar o adaptador. As tabelas correspondem ao arquivo **adapterCode\<adapter ID>\mappings\mappings.xml**.

Observação: O arquivo **sql_queries** não é necessário para o adaptador. É apenas um exemplo.

2. Edite o arquivo **discoveryPatterns\push_adapter.xml**.
 - a. Modifique a marca `<pattern>` usando uma nova id e rótulo de exibição. Substitua:

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

por:

```
<pattern id="MyPushAdapter" displayLabel="My Push Adapter"
xsi:noNamespaceSchemaLocation="../../../Patterns.xsd"
description="Discovery Pattern Description" schemaVersion="9.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```
 - b. Atualize a lista de parâmetros para que reflita os atributos de conexão necessários. Não remova o atributo **probeName**.
3. Renomeie a pasta **adapterCode\PushAdapter** com a ID de adaptador usada na etapa anterior (por exemplo, **adapterCode\MyPushAdapter**).
4. No arquivo **discoveryScript**, há um script **script pushScript.py** que insere os ECs e vincula a um banco de dados externo Oracle. Substitua **discoveryScripts\pushScript.py** pelo script que escreveu (para obter detalhes, consulte ["Escrever scripts Jython"](#) na página 187). Se você renomear o script, a propriedade `jythonScript.name` em **adapterCode\<adapter ID>\push.properties** deverá ser atualizada de acordo.
5. O arquivo **adapterCode\<adapter ID>\mappings\mappings.xml**, localizado em **adapterCode\<adapter ID>\mappings**, é um arquivo de mapeamento de amostra que contém um mapeamento do:
 - Tipo de EC de nó com todos os tipos de EC herdados dele
 - Tipo de EC UNIX sem todos os tipos de EC herdados dele

- Link de dependência com todos os tipos de links herdados dele
- Tipo de link de fala sem os tipos de links herdados dele

Esse exemplo de mapeamento corresponde ao exemplo das tabelas criadas no ORACLE no arquivo `sql_queries` (consulte a etapa 1).

Substitua o arquivo `adapterCode\<adapter ID>\mappings\mappings.xml` pelos arquivos de mapeamento que preparou (para obter detalhes, consulte "[Preparar os arquivos de mapeamento](#)" na página 185).

Se quiser usar um arquivo de mapeamento para cada método TQL, atribua o nome do TQL correspondente a cada arquivo XML, seguido por `.xml`. Nesse caso, o arquivo `mappings.xml` será usado como padrão, caso nenhum arquivo de mapeamento específico seja encontrado para o nome do TQL atual. O nome do arquivo de mapeamento padrão pode ser modificado alterando-se a propriedade `mappingFile.default` em `adapterCode\<adapter ID>\push.properties`.

Esquema do arquivo de mapeamento

Nome e caminho do elemento	Descrição	Atributos
integração	Define o conteúdo de mapeamento do arquivo. Deve ser o bloco mais periférico no arquivo, exceto pela linha de início e quaisquer comentários.	
info (integration)	Define informações sobre os repositórios de dados sendo integrados.	
source (integration > info)	Define informações sobre o repositório de dados de origem.	<ol style="list-style-type: none">1. Nome: tipo Descrição: Nome do repositório de dados de origem. É necessário?: Obrigatório Tipo: Cadeia2. Nome: versões Descrição: Versão(ões) do repositório de dados de origem. É necessário?: Obrigatório Tipo: Cadeia3. Nome: fornecedor Descrição: Fornecedor do repositório de dados de origem. É necessário?: Obrigatório Tipo: Cadeia

Nome e caminho do elemento	Descrição	Atributos
target (integration > info)	Define informações sobre o repositório de dados de destino.	<ol style="list-style-type: none"> 1. Nome: tipo Descrição: Nome do repositório de dados de origem. É necessário?: Obrigatório Tipo: Cadeia 2. Nome: versões Descrição: Versão(ões) do repositório de dados de origem. É necessário?: Obrigatório Tipo: Cadeia 3. Nome: fornecedor Descrição: Fornecedor do repositório de dados de origem. É necessário?: Obrigatório Tipo: Cadeia
targetcis (integration)	Elemento de contêiner de todos os mapeamentos de TEC.	
source_ci_type_tree (integration > targetcis)	Define um TEC de origem e todos os tipos de EC herdados dele.	<ol style="list-style-type: none"> 1. Nome: nome Descrição: Nome do TEC de origem. É necessário?: Obrigatório Tipo: Cadeia 2. Nome: modo Descrição: Tipo da atualização necessária ao tipo de EC atual. É necessário?: Obrigatório Tipo: Uma das seguintes cadeias: <ol style="list-style-type: none"> a. insert: Use apenas se o EC ainda não existir. b. update: Use apenas se o EC já existir. c. update_else_insert: Se o EC existir, atualize; caso contrário, crie um novo EC. d. ignore: Não faça nada com esse tipo de EC.
source_ci_type (integration > targetcis)	Define um TEC de origem sem os tipos de EC herdados dele.	<ol style="list-style-type: none"> 1. Nome: nome Descrição: Nome do TEC de origem. É necessário?: Obrigatório Tipo: Cadeia 2. Nome: modo Descrição: Tipo da atualização

Nome e caminho do elemento	Descrição	Atributos
		necessária ao tipo de EC atual. É necessário?: Obrigatório Tipo: Uma das seguintes cadeias: a. insert: Use apenas se o EC ainda não existir. b. update: Use apenas se o EC já existir. c. update_else_insert: Se o EC existir, atualize; caso contrário, crie um novo EC. d. ignore: Não faça nada com esse tipo de EC.
target_ci_type (integração > targetcis > source_ci_type -OU- integração > targetcis > source_ci_type_tree)	Define um TEC de destino.	1. Nome: nome Descrição: Nome do tipo de EC de destino. É necessário?: Obrigatório Tipo: Cadeia 2. Nome: esquema Descrição: Nome do esquema que será usado para armazenar esse tipo de EC no destino. É necessário?: Não é obrigatório Tipo: Cadeia 3. Nome: namespace Descrição: Indica o namespace desse tipo de EC no destino. É necessário?: Não é obrigatório Tipo: Cadeia
targetprimarykey (integration > targetcis > source_ci_type) -OU- (integration > targetcis > source_ci_type_tree -OU- (integration > targetrelations > link) -OU- (integration > targetrelations >	Identifica atributos de chave primária do TEC de destino.	

Nome e caminho do elemento	Descrição	Atributos
<p>source_link_type_tree)</p> <p>pkey</p> <p>(integration > targetcis> source_ci_type > targetprimarykey</p> <p>-OU-</p> <p>integration > targetcis > source_ci_type_tree > targetprimarykey</p> <p>-OU-</p> <p>(integration > targetrelations > link > targetprimarykey)</p> <p>-OU-</p> <p>integration > targetrelations > source_link_type_tree > targetprimarykey)</p>	<p>Identifica um atributo de chave primária.</p> <p>Obrigatório somente se o modo for update ou insert_else_update.</p>	
<p>target_attribute</p> <p>(integration > targetcis> source_ci_type</p> <p>-OU-</p> <p>integration > targetcis > source_ci_type_tree</p> <p>-OU-</p> <p>integration > targetrelations > link</p> <p>-OU-</p> <p>integration > targetrelations > source_link_type_tree)</p>	<p>Define o atributo do TEC de destino.</p>	<ol style="list-style-type: none"> 1. Nome: nome Descrição: Nome do atributo do TEC de destino. É necessário?: Obrigatório Tipo: Cadeia 2. Nome: datatype Descrição: Tipo de dados do atributo do TEC de destino. É necessário?: Obrigatório Tipo: Cadeia 3. Nome: tamanho Descrição: Para tipos de dados cadeia/caractere, esse é o tamanho inteiro do atributo de destino. É necessário?: Não é obrigatório Tipo. Inteiro 4. Nome. option Descrição. A função de conversão a ser aplicada ao valor. É obrigatório. Falso Tipo. Uma das seguintes cadeias:

Nome e caminho do elemento	Descrição	Atributos
		<p>a. uppercase – Converte para maiúscula</p> <p>b. lowercase – Converte para minúscula</p> <p>Se esse atributo estiver vazio, nenhuma função de conversão será aplicada.</p>
<p>map</p> <p>(integration > targetcis > source_ci_type > target_attribute</p> <p>-OU-</p> <p>integration > targetcis > source_ci_type_tree > target_attribute)</p> <p>-OU-</p> <p>(integration > targetrelations > link > target_attribute</p> <p>-OU-</p> <p>integration > targetrelations > source_link_type_tree > target_attribute)</p>	<p>Especifica como obter o valor do atributo de TECs de origem.</p>	<ol style="list-style-type: none"> <p>Nome. tipo</p> <p>Descrição. O tipo de mapeamento entre os valores de origem e destino.</p> <p>É obrigatório. Obrigatório</p> <p>Tipo. Uma das seguintes cadeias:</p> <ol style="list-style-type: none"> direct – Especifica um mapeamento de 1 para 1 entre o valor do atributo de origem e o valor do atributo de destino. compoundstring – Subelementos são unidos em uma única cadeia e o valor do atributo de destino é definido. childattr – Subelementos são um ou mais atributos de um TEC filho. Um TEC filho é aquele com um relacionamento do tipo composition ou containment. constant – Cadeia estática <p>Nome. value</p> <p>Descrição. Cadeia constante para tipo=constant</p> <p>É obrigatório. Necessário apenas quando tipo=constant</p> <p>Tipo. Cadeia</p> <p>Nome. attr</p> <p>Descrição. Nome de atributo de origem para tipo=direct</p> <p>É obrigatório. Necessário apenas quando tipo=direct</p> <p>Tipo. Cadeia</p>
<p>aggregation</p> <p>(integration > targetcis > source_ci_type > target_attribute > map</p> <p>-OU-</p> <p>integration > targetcis > source_ci_type_tree > target_attribute > map</p>	<p>Especifica como os valores de atributo de EC filho do EC de origem são combinados em um único valor para o mapeamento ao atributo de EC de destino. Opcional.</p>	<p>Nome: tipo</p> <p>Descrição. O tipo da função de agregação</p> <p>É necessário?: Obrigatório</p> <p>Tipo. Uma das seguintes cadeias:</p> <ul style="list-style-type: none"> csv – Agrega todos os valores em uma lista separada por vírgula (numérico ou cadeia/caractere) count – Retorna o total numérico de todos os valores incluídos. sum – Retorna o total numérico de todos

Nome e caminho do elemento	Descrição	Atributos
<p>-OU-</p> <p>(integration > targetrelations > link > target_attribute > map</p> <p>-OU-</p> <p>integration > targetrelations > source_link_type_tree > target_attribute > map)</p> <p>Válido apenas quando o tipo do mapa é childattr</p>		<p>os valores incluídos.</p> <ul style="list-style-type: none"> • average – Retorna a média numérica de todos os valores incluídos. • min – Retorna o menor valor numérico/caractere dentre os valores incluídos. • max – Retorna o maior valor numérico/caractere dentre os valores incluídos.
<p>source_child_ci_type</p> <p>(integration > targetcis > source_ci_type > target_attribute > map</p> <p>-OU-</p> <p>integration > targetcis > source_ci_type_tree > target_attribute > map</p> <p>-OU-</p> <p>(integration > targetrelations > link > target_attribute > map</p> <p>-OU-</p> <p>integration > targetrelations > source_link_type_tree > target_attribute > map)</p> <p>Válido apenas quando o tipo do mapa é childattr.</p>	<p>Especifica a partir de qual EC conectado o atributo filho é retirado.</p>	<ol style="list-style-type: none"> 1. Nome. name Descrição. O tipo do EC filho É obrigatório. Obrigatório Tipo. Cadeia 2. Name. source_attribute Descrição. O atributo do EC filho mapeado. É obrigatório. Necessário somente se o tipo de agregação childAttr (que está no mesmo caminho) não for =count. Tipo. Cadeia

Nome e caminho do elemento	Descrição	Atributos
<p>validation (integration > targetcis > source_ci_type > target_attribute > map -OU- integration > targetcis > source_ci_type_tree > target_attribute > map -OU- (integration > targetrelations > link > target_attribute > map -OU- integration > targetrelations > source_link_type_tree > target_attribute > map)</p> <p>Válido apenas quando o tipo do mapa é childatt</p>	<p>Permite a filtragem para exclusão de ECs filhos do EC de origem com base em valores de atributo. Usado com o subelemento aggregation para alcançar a granularidade que demonstre exatamente que atributos de filhos serão mapeados ao valor de atributo do TEC de destino. Opcional.</p>	<ol style="list-style-type: none"> Nome. minlength Descrição. Exclui cadeias mais curtas do que o valor determinado. É necessário?: Não é obrigatório Tipo. Inteiro Nome. maxlength Descrição. Exclui cadeias mais longas do que o valor determinado. É necessário?: Não é obrigatório Tipo. Inteiro Nome. minvalue Descrição. Exclui números menores do que o valor determinado. É necessário?: Não é obrigatório Tipo. Numérico Nome. maxvalue Descrição. Exclui números maiores do que o valor determinado. É necessário?: Não é obrigatório Tipo. Numérico
<p>targetrelations (integration)</p>	<p>Elemento de contêiner de todos os mapeamentos de relacionamento. Opcional.</p>	
<p>source_link_type_tree (integration > targetrelations)</p>	<p>Faz o mapeamento entre um tipo de relacionamento de origem sem os tipos herdados dele para um relacionamento de destino. Obrigatório apenas se targetrelation estiver presente.</p>	<ol style="list-style-type: none"> Nome: nome Descrição. Nome do relacionamento de origem. É necessário?:Necessário Tipo. Cadeia Nome: target_link_type Descrição. Nome do relacionamento de destino É necessário?: Obrigatório Tipo. Cadeia Nome: nameSpace Descrição: O namespace do vínculo que

Nome e caminho do elemento	Descrição	Atributos
		<p>será criado com o destino. É necessário?: Não é obrigatório Tipo: Cadeia</p> <p>4. Nome: modo Descrição: Tipo da atualização necessária para o vínculo atual. É necessário?: Obrigatório Tipo: Uma das seguintes cadeias:</p> <ul style="list-style-type: none"> ▪ insert – Use-a apenas se o EC ainda não existir. ▪ update – Use-a apenas se o EC já existir. ▪ update_else_insert – Se o EC existir, será atualizado; caso contrário, um novo EC será criado. ▪ ignore – Não faz nada com esse tipo de EC. <p>5. Nome: source_ci_type_end1 Descrição: Tipo de EC End1 do relacionamento de origem. É necessário?: Obrigatório Tipo: Cadeia</p> <p>6. Nome: source_ci_type_end2 Descrição: Tipo de EC End2 do relacionamento de origem. É necessário?: Obrigatório Tipo: Cadeia</p>
<p>link (integration > targetrelations)</p>	<p>Faz o mapeamento entre um relacionamento de origem e um relacionamento de destino. Obrigatório apenas se targetrelation estiver presente.</p>	<p>1. Nome: source_link_type Descrição: Nome do relacionamento de origem. É necessário?: Obrigatório Tipo: Cadeia</p> <p>2. Nome: target_link_type Descrição: Nome do relacionamento de destino. É necessário?: Obrigatório Tipo: Cadeia</p> <p>3. Nome: nameSpace Descrição: O namespace do vínculo que será criado com o destino. É necessário?: Não é obrigatório Tipo: Cadeia</p>

Nome e caminho do elemento	Descrição	Atributos
		<p>4. Nome: modo Descrição: Tipo da atualização necessária para o vínculo atual. É necessário?: Obrigatório Tipo: Uma das seguintes cadeias:</p> <ul style="list-style-type: none"> ■ insert – Use-a apenas se o EC ainda não existir. ■ update – Use-a apenas se o EC já existir. ■ update_else_insert – Se o EC existir, será atualizado; caso contrário, um novo EC será criado. ■ ignore – Não faz nada com esse tipo de EC. <p>5. Nome: source_ci_type_end1 Descrição: Tipo de EC End1 do relacionamento de origem É necessário?: Obrigatório Tipo: Cadeia</p> <p>6. Nome: source_ci_type_end2 Descrição: Tipo de EC End2 do relacionamento de origem. É necessário?: Obrigatório Tipo: Cadeia</p>
<p>target_ci_type_end1 (integration > targetrelations > link -OU- integration > targetrelations > source_link_type_tree)</p>	<p>Tipo de EC End1 do relacionamento de destino.</p>	<p>1. Nome: nome Descrição: Nome do tipo de EC End1 do relacionamento de destino. É necessário?: Obrigatório Tipo: Cadeia</p> <p>2. Nome: superclass Descrição: Nome da superclasse do tipo de EC End1. É necessário?: Não é obrigatório Tipo: Cadeia</p>
<p>target_ci_type_end2 (integration > targetrelations > link -OU- integration > targetrelations ></p>	<p>Tipo de EC End2 do relacionamento de destino.</p>	<p>1. Nome: nome Descrição: Nome do tipo de EC End2 do relacionamento de destino. É necessário?: Obrigatório Tipo: Cadeia</p> <p>2. Nome: superclass Descrição: Nome da superclasse do tipo</p>

Nome e caminho do elemento	Descrição	Atributos
source_link_type_tree)		de EC End2. É necessário?: Não é obrigatório Tipo: Cadeia

Esquema de resultados de mapeamento

Nome e caminho do elemento	Descrição	Atributos
root	A raiz do documento de resultado.	
dados (raiz)	A raiz do próprio dado.	
objetos (raiz > dados)	O elemento raiz dos objetos a atualizar.	
Object (root > data > objects)	Descreve a operação de atualização de um único objeto e todos os seus atributos.	<ol style="list-style-type: none"> Nome: nome Descrição: O nome do tipo de EC É necessário?: Obrigatório Tipo: Cadeia Nome: modo Descrição: Tipo da atualização necessária ao tipo de EC atual. É necessário?: Obrigatório Tipo: Uma das seguintes cadeias: <ol style="list-style-type: none"> insert – Use apenas se o EC ainda não existir. update – Use apenas se o EC já existir. update_else_insert – Se o EC existir, atualize; caso contrário, crie um novo EC. ignore – Não faça nada com esse tipo de EC. Nome: operação Descrição: A operação a ser realizada com esse EC. É necessário: Obrigatório Tipo: Uma das seguintes cadeias: <ol style="list-style-type: none"> add – O EC deve ser adicionado update – O EC deve ser atualizado delete – O EC deve ser excluído Se nenhum valor estiver definido, o valor padrão de add será usado.

Nome e caminho do elemento	Descrição	Atributos
		<p>4. Nome: mamId Descrição: O ID do objeto no CMDB de origem. É necessário?: Obrigatório Tipo: Cadeia</p>
<p>field (root > data > objects> Object -OU- root > data > links > link)</p>	<p>Descreve o valor de um único campo de um objeto. O texto do campo é o novo valor no campo, e se o campo contiver um vínculo, o valor será a ID de uma das extremidades. Cada ID de extremidade aparece como um objeto (sob <objects>).</p>	<p>1. Nome: nome Descrição: O nome do campo. É necessário?: Obrigatório Tipo: Cadeia</p> <p>2. Nome: chave Descrição: Especifica se esse campo é uma chave do objeto. É necessário?: Obrigatório Tipo: Booleano</p> <p>3. Nome: datatype Descrição: O tipo do campo. É necessário?: Obrigatório Tipo: Cadeia</p> <p>4. Nome: tamanho Descrição: Para tipos de dados cadeia/caractere, esse é o tamanho inteiro do atributo de destino. É necessário?: Não é obrigatório Tipo: Inteiro</p>
<p>links (raiz > dados)</p>	<p>O elemento raiz dos vínculos a atualizar.</p>	<p>1. Nome: targetRelationshipClass Descrição: O nome do relacionamento (vínculo) no sistema de destino. É necessário?: Obrigatório Tipo: Cadeia</p> <p>2. Nome: targetParent Descrição: O tipo da primeira extremidade do vínculo (pai). É necessário?: Obrigatório Tipo: Cadeia</p> <p>3. Nome: targetChild Descrição: O tipo da segunda extremidade do vínculo (filho). É necessário?: Obrigatório Tipo: Cadeia</p> <p>4. Nome: modo Descrição: Tipo da atualização necessária ao tipo de EC atual.</p>

Nome e caminho do elemento	Descrição	Atributos
		<p>É necessário?: Obrigatório</p> <p>Tipo: Uma das seguintes cadeias:</p> <ul style="list-style-type: none"> a. insert – Use apenas se o EC ainda não existir. b. update – Use apenas se o EC já existir. c. update_else_insert – Se o EC existir, atualize; caso contrário, crie um novo EC. d. ignore – Não faça nada com esse tipo de EC. <p>5. Nome: operação</p> <p>Descrição: A operação a ser realizada com esse EC.</p> <p>É necessário?: Obrigatório</p> <p>Tipo: Uma das seguintes cadeias:</p> <ul style="list-style-type: none"> a. add – O EC deve ser adicionado b. update – O EC deve ser atualizado c. delete – O EC deve ser excluído <p>Se nenhum valor estiver definido, o valor padrão de add será usado.</p> <p>6. Nome: mamId</p> <p>Descrição: O ID do objeto no CMDB de origem.</p> <p>É necessário?: Obrigatório</p> <p>Tipo: Cadeia</p>

Capítulo 7

Desenvolvendo adaptadores push genéricos aprimorados

Este capítulo inclui:

Visão Geral sobre o desenvolvimento de adaptadores push aprimorados	205
O arquivo de mapeamento	205
O viajante do Groovy	208
Escrever scripts Groovy	211
Implementar interface PushConnector	211
Criar um pacote de adaptador	212
Esquema do arquivo de mapeamento	213

Visão Geral sobre o desenvolvimento de adaptadores push aprimorados

Um adaptador push aprimorado é uma estrutura de dados que representa o resultado da consulta TQL. Cada adaptador criado com base no adaptador push aprimorado tratará dessa estrutura de dados e a aplicará ao seu destino obrigatório.

A estrutura de dados é chamada **ResultTreeNode (RTN)**. O RTN é criado de acordo com o arquivo de mapeamento do adaptador e os resultados da consulta TQL. As consultas usadas para o adaptador push aprimorado devem ser baseadas na raiz, isto é, a consulta deve conter um nó de consulta com o nome do elemento **root** ou um ou mais elementos de relacionamento que começam com o prefixo **root**. Esse EC ou relacionamento serve como o elemento raiz da consulta. Para obter detalhes, consulte "Push de dados" na página 1 no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

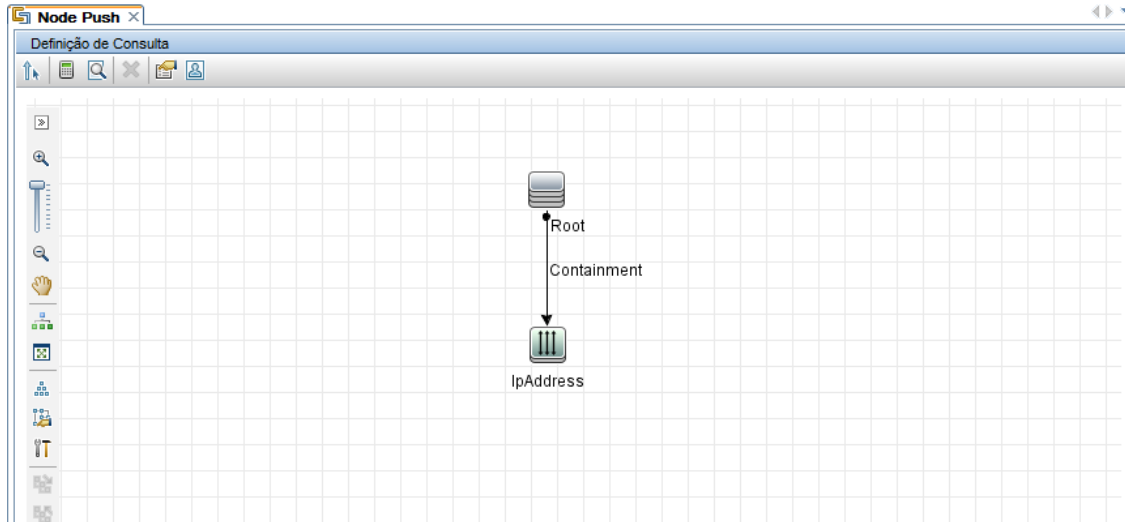
Há duas etapas básicas envolvidas no desenvolvimento do adaptador push aprimorado:

1. Implementar a interface PushConnector – essa interface recebe os dados a adicionar, atualizar e excluir como uma lista de RTNs e executa o push no destino.
2. Criar o arquivo de mapeamento – o arquivo de mapeamento determina a criação da estrutura do RTN, mapeando ECs e atributos do resultado da TQL.

O arquivo de mapeamento

O exemplo a seguir demonstra como criar o arquivo de mapeamento.

Nesse exemplo, simularemos o push de um nó e endereço IP. Criaremos uma consulta TQL chamada: **Node Push**, como a seguir:

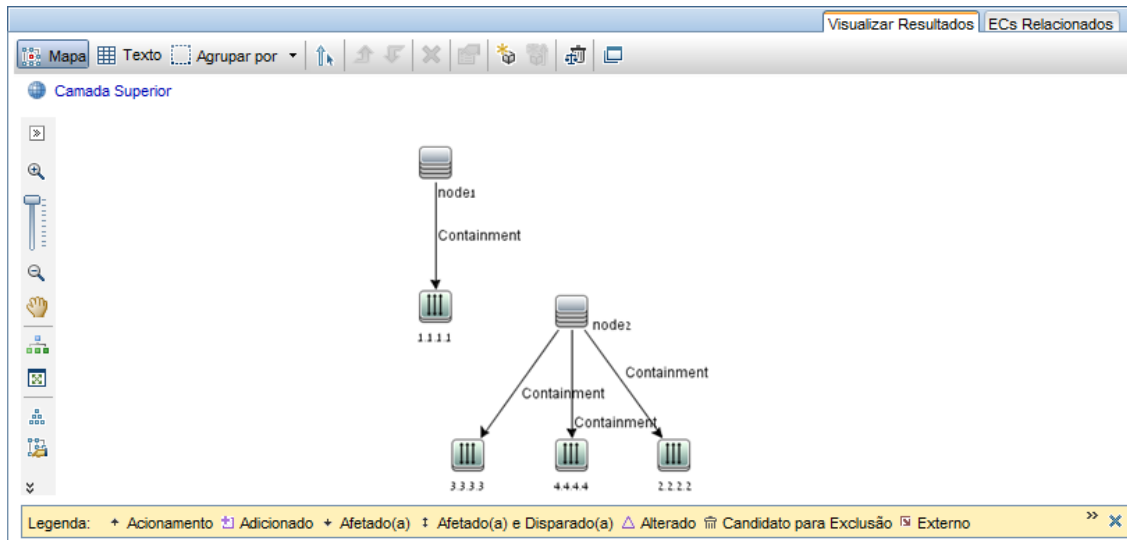


No arquivo de mapeamento, criamos dois tipos de EC de destino: **Computer** e **IP**. O computador tem uma variável e dois atributos. O IP tem um atributo.

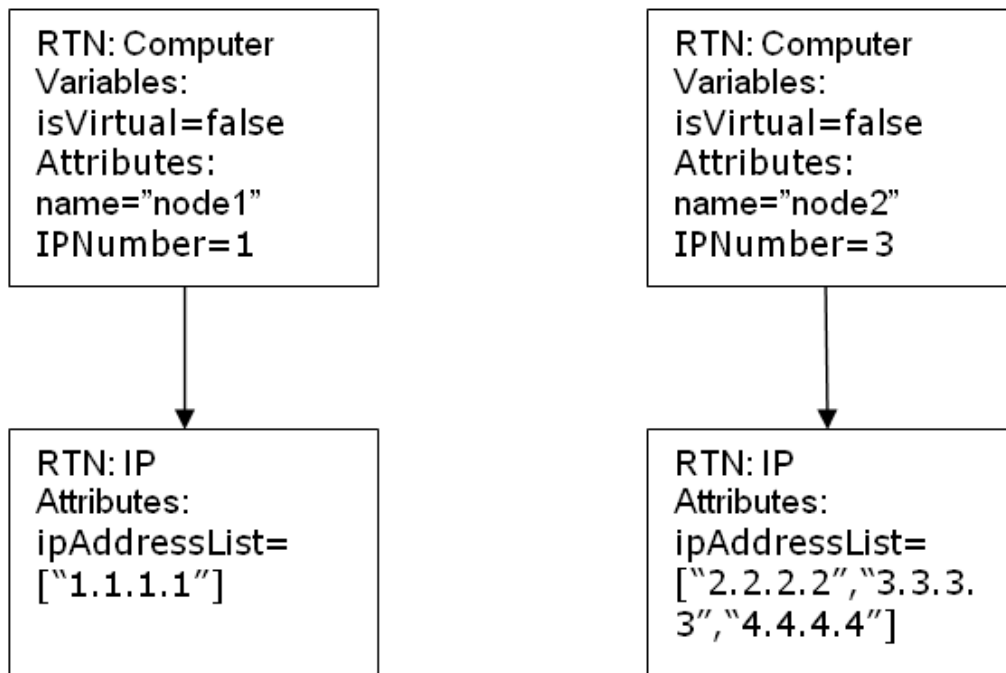
Veja a seguir o arquivo XML de mapeamento:

```
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://schemas.hp.com/ucmdb/1/pushAdap
ter">
  <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>
```

Os resultados da consulta aparecem da seguinte forma:



A lista do RTN que será criada de acordo com esse arquivo de mapeamento será parecida com:



Cada instância raiz é mapeada separadamente usando o arquivo de mapeamento. Portanto, nesse exemplo, o PushConnector recebe uma lista de duas raízes RTN.

Observação: O adaptador push anterior tinha a capacidade de criar um mapeamento geral para um tipo de EC. O novo mapeamento de adaptador push é por consulta TQL. Enquanto estiver executando um trabalho de push que usa uma consulta chamada x, o adaptador procura o arquivo de mapeamento relevante (o que tem o atributo: query-name=x).

Você pode calcular os valores no arquivo de mapeamento usando a linguagem de script groovy. Para obter detalhes, consulte ["O viajante do Groovy"](#) abaixo.

O viajante do Groovy

É possível acessar os resultados da consulta TQL da seguinte maneira:

- **Root[*attr*]** retorna o atributo ***attr*** do elemento raiz.
- **Root.Query_Element_Name** retorna uma lista das instâncias do EC nomeadas na TQL `Query_Element_Name` que estão vinculadas ao EC raiz atual.
- **Root.Query_Element_Name[2][*attr*]** retorna o atributo ***attr*** do terceiro `Query_Element_Name` vinculado ao EC raiz atual.
- **Root.Query_Element_Name*.getAt(*attr*)** retorna uma lista dos atributos ***attr*** das instâncias do EC chamadas `Query_Element_Name` na TQL que estão vinculadas ao EC raiz atual.

Esses são atributos adicionais que podem ser acessados pelo viajante do groovy:

- ***cmdb_id*** – retorna a ID do UCMDB do EC ou relacionamento como uma cadeia.
- ***external_cmdb_id*** – retorna a ID externa do EC ou relacionamento como uma cadeia.
- ***Element_type*** – retorna o tipo de elemento do EC ou relacionamento como uma cadeia.

A cadeia de importação:

```
<import>
<scriptFile path="mappings.scripts.PushFunctions"/>
</import>
```

Isso significa que estamos declarando uma importação para todos os scripts do groovy no arquivo de mapeamento. Nesse exemplo, **PushFunctions** é um arquivo de script do groovy que contém algumas funções estáticas, e podemos acessá-lo durante o mapeamento (isto é, `value="PushFunctions.foo()"`)

source_instance_type

O mapeamento é feito por TQL, o valor do nome da consulta é a TQL relacionada do mapeamento atual. O `*` significa que esse arquivo de mapeamento é associado a todas as consultas TQL que começam com o prefixo: **Node Push**.

```
<source_instance_type query-name="Node Push*" root-element-name="Root">
```

A tag `source_instance_type` designa o elemento raiz que estamos mapeando.

`root-element-name` deve ser exatamente igual ao nome da raiz na TQL.

target_ci_type

Essa tag é usada para a criação do RTN.

O atributo de nome representa o nome `target_ci_type`: `name=Computer`

O atributo ***is-valid*** é um valor booleano calculado durante o mapeamento e determina se o `target_ci` atual é válido. `target_ci_types` inválidos não são adicionados ao RTN. Neste exemplo, não

queremos criar uma instância `target_ci_type` para a qual o atributo `root_iscandidatefordeletion` no UCMDB é verdadeiro.

O `target_ci_type` pode ter variáveis calculadas durante o mapeamento:

```
<variable name="vSerialNo" datatype="STRING" value="Root['serial_
number']"/>
```

A variável `vSerialNo` obtém o valor do `serial_number` da raiz atual.

O atributo do RTN é criado pela tag `target_mapping`. O resultado da execução do script do groovy no campo `value` é atribuído ao atributo do RTN.

```
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
```

`SerialNo` atribui o valor da variável `vSerialNo`.

É possível definir um `target_ci_type` como filho de outro `target_ci_type`, como indicado a seguir:

```
<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>
```

O **Portfólio** do RTN terá um RTN filho chamado **Asset**.

for-each-source-ci

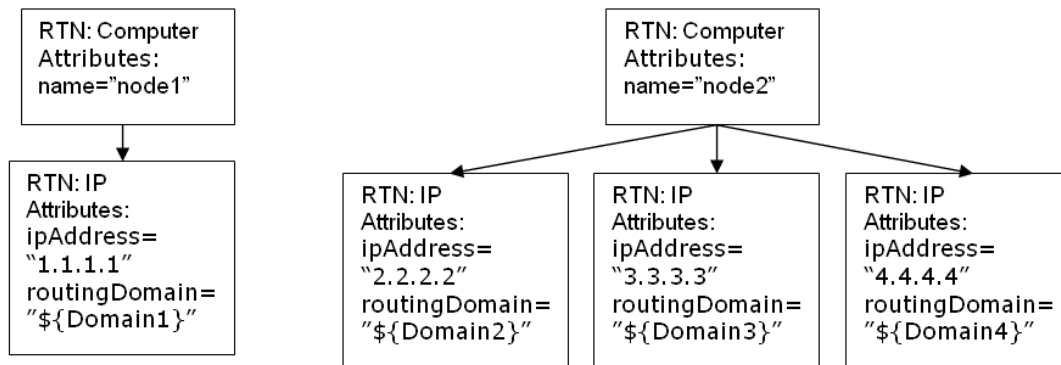
Essa tag lista os ECs específicos da instância raiz. Ela tem os seguintes campos:

- `source-cis=""` – a lista de ECs para as quais um EC de destino é criada. Esta lista é definida pelo viajante do Groovy no campo `Root.IpAddress`.
- `count-index=""` – uma variável que mantém o índice do EC na iteração atual do loop.
- `var-name=""` – o nome do EC na iteração atual do loop.

Vamos modificar nosso arquivo de mapeamento de exemplo:

```
<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>
```

A lista do RTN que será criada de acordo com esse arquivo de mapeamento será parecida com:



dynamic_mapping

Essa tag adiciona a capacidade de criar um mapeamento de dados a partir do armazenamento de dados de destino durante a criação da estrutura do RTN.

Exemplo: Suponha que o destino seja um banco de dados com uma tabela chamada **Computador** com uma coluna **id** e uma coluna **nome** correlacionada a **Node.name** no UCMDb. As duas colunas são únicas. O banco de dados também tem uma tabela chamada **IP** que tem uma chave com referência a **parentID** na tabela Computador. O 'dynamic_mapping' pode criar um mapa que armazena o nome e a id como <nome,id>. Com base nesse mapa, o adaptador pode corresponder ids com computadores e aplicar o valor correto ao atributo parentID na tabela IP. Você pode usar esse mapa para atribuir um valor ao atributo parentID ao criar o RTN.

O mapeamento é determinado pelo **map_property**. O dynamic_mapping é executado uma vez para cada bloco.

```
<dynamic_mapping name="IdByName " keys-unique="true">
```

O atributo **name** representa o nome do mapa. O atributo **keys-unique** indica se as chaves são únicas (cada chave é mapeada para um valor ou para um conjunto de valores).

O nome do mapa neste exemplo é **IdByName** e tem chaves exclusivas. Para acessar o mapa no script, execute o seguinte comando:

```
DynamicMapHolder.getMap('IdByName')
```

Ele retorna uma referência para aquele mapa.

A marcação **map_property** cria a propriedade na qual o mapeamento se baseia.

Exemplo:

```
<map_property property-name="SQLQuery" datatype="STRING"
property-value="SELECT name, id FROM Computer"/>
```

Nesse exemplo, o nome da propriedade é **SQLQuery** e seu valor é uma instrução SQL que cria o mapa. A implementação dos métodos **retrieveUniqueMapping** e **retrieveNonUniqueMapping** para a interface PushConnector determinará o conteúdo real do mapa retornado.

Variáveis globais

As seguintes variáveis globais são acessíveis para o script groovy no arquivo de mapeamento:

- **Topology** – Tipo: Topology. Uma instância da topologia do bloco atual.
- **QueryDefinition** - Tipo: QueryDefinition. Uma instância da definição de consulta da TQL atual.
- **OutputCI** – Tipo: ResultTreeNode. O RTN do elemento raiz no mapeamento de árvore atual.
- **ClassModel** – Tipo: ClassModel. Uma instância do modelo de classe.
- **CustomerInformation** – Tipo: CustomerInformation. Informações sobre o cliente que executa o trabalho.
- **Logger** – Tipo: DataAdapterLogger. Esse logger está disponível no adaptador para escrever logs na estrutura de logs do UCMDB.

Escrever scripts Groovy

Nesta seção, criamos o arquivo **PushFunctions.groovy**. Este arquivo conterá funções estáticas usadas durante o mapeamento da instância raiz.

```
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. } }
```

Implementar interface PushConnector

A implementação deve suportar as seguintes etapas básicas:

```
public class PushExampleAdapter implements PushConnector
{

public UpdateResult pushTreeNodes(ResultTreeNodeActionData
resultTreeNodes, QueryDefinition queryDefinition) throws
DataAccessException{

// 1. build an UpdateResult instance - the UpdateResult is used to
return mappings between the sent ids to the actual ids that entered
the data store. // Also has an update status which allows to pass
errors to the statistics in the UI. // 2. handle the data: // a.
handle data to add. Can be retrieved by:
```

```
resultTreeNodeActionData.getDataToAdd()); // 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
DataAccessException { // 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 will be called in the // mapping stage of the adapter execution,
before the 'UpdateResult pushTreeNodes' method. // This method is
called when the 'keys-unique' attribute of the 'dynamic_mapping' tag
is true. }

Map<Object, Set<Object>> retrieveNonUniqueMapping(MappingQuery
mappingQuery){ // This method is called when the 'keys-unique'
attribute of the 'dynamic_mapping' tag is false. // In this case a
key can be mapped to several values. } }
```

Criar um pacote de adaptador

O pacote do adaptador deve conter as seguintes pastas:

- **adapterCode.** Nessa pasta, crie uma pasta chamada **PushExampleAdapter**, que conterá o arquivo `.jar` que criamos a partir de `PushExampleAdapter.java`. Ela também conterá uma pasta chamada **mappings**, onde você poderá colocar o arquivo de mapeamento criado anteriormente, **computerIPMapping.xml**. Ela também deve conter outra pasta chamada **scripts** que contém o arquivo **PushFunctions.groovy**.
- **discoveryConfigFiles.** Conter arquivos de configuração como os códigos de erro usados ao reportar um erro usando `UpdateResult`. Neste exemplo, a pasta está vazia.
- **discoveryPatterns.** Conter **push_example_adapter.xml**.
- **tql.** Conter a consulta TQL criada para o exemplo. Essa pasta é opcional, mas quando o pacote é implantado, o TQL é criado automaticamente.

Esquema do arquivo de mapeamento

Nome e caminho do elemento	Descrição	Atributos
Integração	Define o conteúdo de mapeamento do arquivo. Deve ser o bloco mais periférico no arquivo, exceto pela linha de início e quaisquer comentários.	
informações (integração)	Define informações sobre os repositórios de dados sendo integrados.	
origem (informações)	O produto de origem	Nome: nome Descrição: o nome do produto É necessário?: necessário Tipo: Cadeia
		Nome: fornecedor Descrição: o fornecedor do produto É necessário?: necessário Tipo: Cadeia
		Nome: versões Descrição: a versão do produto É necessário?: necessário Tipo: decimal
destino (informações)	O produto de destino	Nome: nome Descrição: o nome do produto É necessário?: necessário Tipo: Cadeia
		Nome: fornecedor

Nome e caminho do elemento	Descrição	Atributos
		<p>Descrição: o fornecedor do produto</p> <p>É necessário?: necessário</p> <p>Tipo: Cadeia</p>
		<p>Nome: versões</p> <p>Descrição: a versão do produto</p> <p>É necessário?: necessário</p> <p>Tipo: decimal</p>
Importação (integração)	Um elemento de contêiner para arquivos de script importados.	
scriptFile (integração>importação)	Define o arquivo de script groovy a ser importado.	<p>Nome: caminho</p> <p>Descrição: o caminho do arquivo de script</p> <p>É necessário?: necessário</p> <p>Tipo: cadeia</p>
Targetcis (integração)	Um elemento de contêiner para tipos de EC de destino.	
Source_instance_type (integração>targetcis)	Define o tipo de instância do EC de origem. Deve ser o elemento raiz conforme definido no TQL.	<p>Nome: query-name.</p> <p>Descrição: O nome do TQL relevante</p> <p>É necessário?: necessário</p> <p>Tipo: Cadeia</p>
		<p>Nome: nome</p> <p>Descrição: O elemento raiz conforme definido no TQL.</p> <p>É necessário?: necessário</p> <p>Tipo: Cadeia</p>
dynamic_mapping	Define um mapa criado uma vez por	Nome: nome

Nome e caminho do elemento	Descrição	Atributos
(integração>targetcis>source_instance_type)	bloco.	<p>Descrição: o nome do mapa</p> <p>É necessário?: necessário</p> <p>Tipo: Cadeia</p> <hr/> <p>Nome: keys-unique</p> <p>Descrição: Declara se as chaves são exclusivas ou não.</p> <p>É necessário?: necessário</p> <p>Tipo: booleano</p>
target_ci_type (integração>targetcis>source_instance_type) -OU- (integração>targetcis>source_instance_type>for-each-source-ci)	Define o tipo de EC de destino a ser adicionado ao RTN.	<p>Nome: nome</p> <p>Descrição: o nome do tipo de EC de destino</p> <p>É necessário?: necessário</p> <p>Tipo: Cadeia</p> <hr/> <p>Nome: is-valid</p> <p>Descrição: verifica se o EC de destino atual é válido de acordo com o script fornecido.</p> <p>É necessário?: não é necessário</p> <p>Tipo: Cadeia (script groovy)</p>
Variável (target_ci_type)	Define uma variável para o tipo de EC de destino.	<p>Nome: nome</p> <p>Descrição: o nome da variável</p> <p>É necessário?: necessário</p> <p>Tipo: Cadeia</p> <hr/> <p>Nome: datatype</p>

Nome e caminho do elemento	Descrição	Atributos
		<p>Descrição: o tipo de dado da variável.</p> <p>É necessário?: necessário</p> <p>Tipo: type-enum (pode ser um dos seguintes: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)</p>
target_mapping (target_ci_type)	Define um atributo para o tipo de EC de destino.	<p>Nome: valor</p> <p>Descrição: O valor para atribuir à variável</p> <p>É necessário?: necessário</p> <p>Tipo: script groovy</p>
		<p>Nome: nome</p> <p>Descrição: o nome do atributo</p> <p>É necessário?: necessário</p> <p>Tipo: Cadeia</p>
		<p>Nome: datatype</p> <p>Descrição: o tipo de dado da variável.</p> <p>É necessário?: necessário</p> <p>Tipo: type-enum (pode ser um dos seguintes: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST,</p>

Nome e caminho do elemento	Descrição	Atributos
		<p>STRING_LIST)</p> <p>Nome: valor</p> <p>Descrição: O valor para atribuir à variável</p> <p>É necessário?: necessário</p> <p>Tipo: script groovy</p> <hr/> <p>Nome: ignore-on-null</p> <p>Descrição: Se o valor de mapeamento de destino for nulo e esse atributo for TRUE, ignore o atributo</p> <p>É necessário?: não é necessário</p> <p>Tipo: Booleano (script groovy)</p>
before-mapping (target_ci_type)	Um script groovy que é executado antes do mapeamento do tipo de EC de destino.	
after-mapping (target_ci_type)	Um script groovy que é executado após o mapeamento do tipo de EC de destino.	
for-each-source-ci (target_ci_type)	Define uma iteração de ECs específicos da instância raiz.	<p>Nome: count-index</p> <p>Descrição: o índice do EC iterado atual</p> <p>É necessário?: necessário</p> <p>Tipo: Cadeia</p> <hr/> <p>Nome: var-name</p> <p>Descrição: a variável que se refere ao EC iterado atual.</p> <p>É necessário?: não é necessário</p> <p>Tipo: Cadeia</p>

Nome e caminho do elemento	Descrição	Atributos
		<p>Nome: source-cis</p> <p>Descrição: O nome do EC na consulta que deve ser iterado</p> <p>É necessário?: necessário</p> <p>Tipo: Cadeia (script groovy)</p>

Usando APIs

Capítulo 8

Introdução a APIs

Este capítulo inclui:

Visão geral sobre APIs220

Visão geral sobre APIs

As seguintes APIs estão incluídas com o HP Universal CMDB:

- **API Java do UCMDB.** Explica como ferramentas de terceiros ou personalizadas podem usar a API Java para extrair dados e cálculos e gravar dados no UCMDB (Universal Configuration Management Database). Para obter detalhes, consulte "[API do HP Universal CMDB](#)" na página 221.
- **API de Serviço Web do UCMDB.** Permite a criação de definições de itens de configuração e seus relacionamentos topológicos com o UCMDB (Universal Configuration Management Database), além da pesquisa de informações usando consultas TQL e ad hoc. Para obter detalhes, consulte "[API de serviço Web do RTSM \(HP Universal CMDB\)](#)" na página 1.
- **API do serviço Web do Gerenciamento de Fluxo de Dados** Permite o gerenciamento de sondas, trabalhos, acionadores e credenciais para o Gerenciamento de Fluxo de Dados. Para obter detalhes, consulte "[API do Gerenciamento de Fluxo de Dados](#)" na página 228.

Observação: Para obter o valor total da documentação da API, é recomendável acessar a documentação online. A versão em PDF não tem os links para a documentação da API gerada em formato HTML.

Capítulo 9

API do HP Universal CMDB

Este capítulo inclui:

Convenções	221
Usando a API do HP Universal CMDB	221
Estrutura geral de um aplicativo	222
Colocar o arquivo Jar da API no classpath	224
Criar um usuário de integração	224
Referência de API do HP Universal CMDB	226
Casos de uso	226
Exemplos	227

Convenções

Este capítulo usa as seguintes convenções:

- **UCMDB** refere-se ao banco de dados de Gerenciamento de Configuração Universal em si. HP Universal CMDB refere-se ao aplicativo.
- Os elementos e argumentos de método do UCMDB são escritos levando em conta maiúsculas e minúsculas do mesmo modo como são especificados nas interfaces.

Para ver a documentação completa sobre as APIs disponíveis, consulte [HP UCMDB API Reference](#).

Esses arquivos estão localizados na seguinte pasta:

```
\\<diretório raiz do UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html
```

Usando a API do HP Universal CMDB

Observação: Use este capítulo junto com a documentação da API Java, disponível na Biblioteca de Documentação online.

A API do HP Universal CMDB é usada para integrar aplicativos ao Universal CMDB (CMDB). A API fornece métodos para:

- Adicionar, remover e atualizar ECs e relações no CMDB
- Recuperar informações sobre o modelo de classe

- Recuperar informações do histórico do UCMDB
- Executar cenários hipotéticos
- Recuperar informações sobre elementos de configuração e relacionamentos

Métodos para recuperar informações sobre elementos de configuração e relacionamentos geralmente usam o TQL (Topology Query Language). Para obter detalhes, consulte "[TQL \(Topology Query Language\)](#)" na página 1 no *Guia de Modelagem do HP Universal CMDB*.

Usuários da API do HP Universal CMDB devem ter familiaridade com:

- linguagem de programação Java
- HP Universal CMDB

Esta seção inclui os seguintes tópicos:

- "[Usos da API](#)" abaixo
- "[Permissões](#)" abaixo

Usos da API

A API é usada para cumprir inúmeros requisitos de negócios. Por exemplo, um sistema de terceiros pode consultar o modelo de classe para ver informações sobre os ECs (elementos de configuração) disponíveis. Para ver mais cenários de usos, consulte "[Casos de uso](#)" na página 226.

Permissões

O administrador fornece credenciais de logon para conexão com a API. O cliente da API precisa de nome de usuário e senha de um usuário de integração definido no CMDB. Esses usuários não representam pessoas que usam o CMDB, e sim aplicativos que se conectam ao CMDB.

Cuidado: O cliente da API também pode funcionar com usuários regulares desde que eles tenham permissão de autenticação na API. No entanto, essa opção não é recomendada.

Para obter detalhes, consulte "[Criar um usuário de integração](#)" na página 224.

Estrutura geral de um aplicativo

Há apenas uma fábrica estática, a `UcmdbServiceFactory`. Essa fábrica é o ponto de entrada de um aplicativo. A `UcmdbServiceFactory` expõe métodos `getServiceProvider`. Esses métodos retornam uma instância da interface **`UcmdbServiceProvider`**.

O cliente cria outros objetos usando métodos de interface. Por exemplo, para criar uma nova definição de consulta, o cliente:

1. Obtém o serviço de consulta do objeto de serviço CMDB principal
2. Obtém um objeto de fábrica de consulta do objeto de serviço
3. Obtém uma nova definição de consulta da fábrica

```
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 Query");
queryDefinition.addNode("Node").ofType("host");
Topology topology = queryService.executeQuery(queryDefinition);
System.out.println("There are " + topology.getAllCIs().size() + "
hosts in uCMDB");
    
```

Os serviços disponíveis no **UcmdbService** são:

Métodos de serviço	Uso
getClassModelService	informação sobre tipos de ECs e relações
getConfigurationService	Gerenciamento de configurações de infraestrutura, para configuração do servidor
getDDMConfigurationService	Configurar o sistema do Gerenciamento de Fluxo de Dados
getDDMManagementService	Analisar e exibir o andamento, resultados e erros do sistema de Gerenciamento de Fluxo de Dados
getHistoryService	Informações sobre o histórico de ECs monitorados (alterações, remoções e assim por diante)
getImpactAnalysisService	Executar o cenário de análise de impacto (também conhecido como correlação).
getQueryManagementService	Gerenciar acesso a consultas - salvar, excluir, listar existente. Fornece também validação de consulta e a descoberta de dependências de consultas.
getResourceBundleManagementService	Marcação de recurso (serviços de "agrupamento"). Permite a criação explícita de novas marcas e a remoção de marcas de todos os recursos marcados.
getStateService	Fornecer serviços para gerenciamento de estados (listar, adicionar, remover e assim por diante)
getSoftwareSignatureService	Definir itens de software a serem descobertos pelo sistema de Gerenciamento de Dependência e Descoberta
getSnapshotService	Fornecer serviços para gerenciamento de instantâneos (obter, salvar, comparar e assim por diante)
getTopologyQueryService	Obter informações sobre o universo de TI

Métodos de serviço	Uso
getTopologyUpdateService	Alterar informações no universo de TI
getViewService	Exibir o serviço de execução (definição de execução, salvamento de execução) e serviço de gerenciamento (salvar, excluir, listar existente). Fornece também validação da exibição e a descoberta de dependências.
getViewArchiveService	Exibir resultados de estatísticas. Permite salvar o resultado da exibição atual e recuperar resultados salvos anteriormente.
SystemHealthService	Fornecer serviços de integridade do sistema (indicadores básicos de desempenho do sistema, capacidade e métricas de disponibilidade)

O cliente comunica-se com o servidor usando HTTP.

Colocar o arquivo Jar da API no classpath

O uso desse conjunto de API exige o arquivo **ucmdb-api.jar**. Você pode baixar o arquivo inserindo `http://<localhost>:8080` em um navegador da Web onde `localhost` é a máquina onde o UCMDb está instalado e clicando no link de **download do cliente da API**.

Coloque o arquivo jar no caminho de classe antes de compilar ou executar seu aplicativo.

Observação: O uso do Jar da API Java do UCMDb requer que você tenha o JRE versão 6 ou posterior instalado.

Criar um usuário de integração

Você pode criar um usuário dedicado para integrações entre outros produtos e o UCMDbUCMDb. Esse usuário permite que um produto que use o SDK cliente do UCMDb seja autenticado no SDK servidor e execute as APIs. Aplicativos escritos com esse conjunto de API devem fazer logon usando credenciais de usuário de integração.

Cuidado: Também é possível conectar a um usuário comum do UCMDb, (por exemplo, admin); no entanto, essa opção não é recomendada. Para conectar a um usuário do UCMDb, você deve conceder a ele permissão de autenticação da API.

Para criar um usuário de integração:

1. Inicie o navegador da Web e insira o endereço do servidor, da seguinte maneira:

```
http://localhost:8080/jmx-console.
```

Você pode precisar fazer logon com um nome de usuário e senha (os valores padrão são sysadmin/sysadmin).

2. No UCMDB, clique em **service=UCMDB Authorization Services**.
3. Localizar a operação **createUser**. Esse método aceita os seguintes parâmetros:
 - **customerId**. ID do cliente.
 - **username**. O nome do usuário de integração.
 - **userDisplayName**. O nome de exibição do usuário de integração.
 - **userLoginName**. O nome de logon do usuário de integração.
 - **password**. A senha do usuário de integração.
4. Clique em **Chamar**.
5. Em um ambiente de um único locatário, localize o método **setRolesForUser** e insira os seguintes parâmetros:
 - **userName**. O nome do usuário de integração.
 - **funções**. SuperAdmin.Clique em **Chamar**.
6. Em um ambiente de locação múltipla, localize o método **grantRolesToUserForAllTenants** e insira os seguintes parâmetros para atribuir a função em conexão com todos os locatários:
 - **userName**. O nome do usuário de integração.
 - **funções**. SuperAdmin.Clique em **Chamar**.

Como alternativa, para atribuir a função em conexão com locatários específicos, chame o método **grantRolesToUserForTenants**, usando os mesmos valores de parâmetro de funções e userName. Para o parâmetro **tenantNames**, insira os locatários necessários.
7. Crie mais usuários ou feche o console JMX.
8. Faça logon no UCMDB como administrador.
9. Na guia **Administração**, execute o **Gerenciador de Pacotes**.
10. Clique no ícone **Criar pacote personalizado**.
11. Digite um nome para o novo pacote e clique em **Avançar**.
12. Na guia Seleção de Recurso, emr **Configurações**, clique em **Usuários**.
13. Selecione um ou mais usuários que tenha criado usando o console JMX.
14. Clique em **Avançar** e em **Concluir**. Seu novo pacote aparecerá na lista Nome do Pacote no Gerenciador de Pacotes.
15. Implemente o pacote aos usuários que executarão os aplicativos API.

Para ver detalhes, consulte "Implantar um pacote" no *Guia de Administração do HP Universal CMDB*.

Observação:

A senha do usuário de integração é por cliente. Para criar um usuário de integração mais forte para uso entre clientes, utilize um **systemUser** com o sinalizador **isSuperIntegrationUser** definido como **true**. Use os métodos **systemUser (removeUser, resetPassword, UserAuthenticate** e assim por diante).

Há dois usuários de sistema prontos; é recomendável alterar suas senhas após a instalação usando o método **resetPassword**.

- **sysadmin/sysadmin**
- **UISysadmin/UISysadmin** (Esse usuário é também o superusuário de integração **SuperIntegrationUser**).

Se você alterar a senha **UISysadmin** usando **resetPassword**, você deverá executar o seguinte método: no Console JMX, localize o serviço **UCMDB-UI:name=UCMDB Integration**. Execute **setCMDBSuperIntegrationUser** com o nome de usuário e a nova senha do usuário de integração.

Referência de API do HP Universal CMDB

Esses arquivos estão localizados na seguinte pasta:

\\<diretório raiz do UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Casos de uso

Os seguintes casos de uso pressupõem dois sistemas:

- Servidor HP Universal CMDB
- Um sistema de terceiros que contém um repositório dos elementos de configuração

Esta seção inclui os seguintes tópicos:

- ["Populando o CMDB" abaixo](#)
- ["Consultando o CMDB" na página seguinte](#)
- ["Consultando o modelo de classe" na página seguinte](#)
- ["Analisando o impacto das alterações " na página seguinte](#)

Populando o CMDB

Casos de uso:

- Um gerenciamento de ativos de terceiros atualiza o CMDB com informações disponíveis somente no gerenciamento de ativos.
- Inúmeros sistemas de terceiros populam o CMDB para criar um CMDB central que consiga controlar as alterações e executar análise de impacto.
- Um sistema de terceiros cria Elementos de Configuração e Relacionamentos de acordo com uma lógica de negócios de terceiros para aproveitar os recursos de consulta do UCMDB.

Consultando o CMDB

Casos de uso:

- Um sistema de terceiros obtém os Elementos de Configuração e Relacionamentos que representam o sistema SAP obtendo os resultados do TQL SAP.
- Um sistema de terceiros obtém a lista de servidores Oracle que foram adicionados ou alterados nas últimas cinco horas.
- Um sistema de terceiros obtém a lista de servidores cujo nome de host contém a subcadeia lab.
- Um sistema de terceiros localiza os elementos relacionados a um determinado EC obtendo seus vizinhos.

Consultando o modelo de classe

Casos de uso:

- Um sistema de terceiros permite que os usuários especifiquem o conjunto de dados que serão recuperados do CMDB. Uma interface do usuário pode ser criada com base no modelo de classe para mostrar aos usuários as propriedades possíveis e lhes solicitar os dados necessários. O usuário poderá escolher as informações que serão recuperadas.
- Um sistema de terceiros explora o modelo de classe quando o usuário não pode acessar a interface do usuário do UCMDB.

Analisando o impacto das alterações

Caso de uso:

Um sistema de terceiros gera saída de uma lista dos serviços comerciais que poderiam sofrer impacto de uma alteração em um host especificado.

Exemplos

Veja as amostras de código a seguir:

- [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](#)

Esses arquivos estão localizados no seguinte diretório:

```
\\<diretório raiz do UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\JavaSDK_Samples\
```

Capítulo 10

API do Gerenciamento de Fluxo de Dados

Este capítulo inclui:

Visão Geral da API do Gerenciamento de Fluxo de Dados	228
Convenções	228
Serviço Web do Gerenciamento de Fluxo de Dados	229
Chamar o serviço Web	229
Métodos de Gerenciamento de Fluxo de Dados	230
Amostra de código	240

Visão Geral da API do Gerenciamento de Fluxo de Dados

Este capítulo explica como ferramentas de terceiros ou personalizadas podem usar o serviço Web do Gerenciamento de Fluxo de Dados da HP para gerenciar o Gerenciamento de Fluxo de Dados.

Para obter a documentação completa sobre as operações disponíveis, consulte *Referência de Esquema de Mapeamento do HP Discovery and Dependency*. Esses arquivos estão localizados na seguinte pasta:

<Diretório Raiz do UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_Schema\webframe.html

Convenções

Este capítulo usa as seguintes convenções:

- Esse `Element` de estilo indica que um item é uma entidade no banco de dados ou um elemento definido no esquema, incluindo estruturas transmitidas ou retornadas por métodos. Um texto simples indica que o item está sendo discutido em um contexto geral.
- Os elementos e argumentos de método do Gerenciamento de Fluxo de Dados são escritos levando em conta maiúsculas e minúsculas do mesmo modo como são especificados no esquema. Isso normalmente significa que um nome de classe ou uma referência genérica a uma instância da classe é capitalizada. Um elemento ou argumento de um método não tem inicial maiúscula. Por exemplo, `credential` é um elemento de tipo `Credential` repassado para um método.

Serviço Web do Gerenciamento de Fluxo de Dados

O serviço Web do Gerenciamento de Fluxo de Dados da HP é uma API usada para integrar aplicativo com o HP Universal CMDB. A API fornece métodos para:

- **Gerenciar credenciais.** Exibir, adicionar, atualizar e remover.
- **Gerenciar trabalhos.** Exibir status, ativar e desativar.
- **Gerenciar intervalos de Sonda.** Exibir, adicionar e atualizar.
- **Gerenciar acionadores.** Adicionar ou remover um EC acionador e adicionar, remover ou desabilitar um TQL acionador.
- **Exibir dados gerais.** Dados sobre domínios e sondas.

Os usuários do serviço Web do Gerenciamento de Fluxo de Dados da HP devem estar familiarizados com:

- A especificação SOAP
- Uma linguagem de programação orientada a objeto como C++, C# ou Java
- HP Universal CMDB
- Gerenciamento de Fluxo de Dados

Permissões

O administrador fornece credenciais de logon para conexão com o Serviço Web. Os níveis de permissão são Exibir, Atualizar e Executar. Para exibir as permissões necessárias para cada operação, consulte a documentação de solicitação de cada operação na *Referência de Esquema de Mapeamento do HP Discovery and Dependency*.

Chamar o serviço Web

O serviço Web do HP Discovery and Dependency Mapping permite que sejam chamados métodos do servidor usando técnicas de programação SOAP padrão. Se não for possível analisar a instrução ou se houver um problema ao invocar o método, os métodos de API lançarão uma exceção `SoapFault`. Quando uma exceção `SoapFault` é lançada, o serviço popula um ou mais campos de mensagem de erro, código de erro e mensagem de exceção. Se não houver nenhum erro, os resultados da invocação serão retornados.

Para chamar o serviço, use:

- Protocolo: `http` ou `https` (depende da configuração do servidor)
- URL: `<Servidor do UCMDB>:8080/axis2/services/DiscoveryService`
- Senha padrão: `"admin"`
- Nome de usuário padrão: `"admin"`

Os programadores SOAP podem acessar o WSDL em:

- `axis2/services/DiscoveryService?wsdl`

Métodos de Gerenciamento de Fluxo de Dados

Esta seção contém uma lista das operações de serviço Web e um breve resumo de seu uso. Para ver uma documentação completa da solicitação e resposta de cada operação, consulte *Referência de Esquema do HP Discovery and Dependency*.

Esta seção inclui os seguintes tópicos:

- "Estruturas de dados" abaixo
- "Gerenciando métodos de trabalho de descoberta" na página seguinte
- "Gerenciando métodos de acionador" na página 232
- "Métodos de dados de sonda e domínio" na página 234
- "Métodos de dados de credenciais" na página 236
- "Métodos de atualização de dados" na página 238

Estruturas de dados

Algumas das estruturas de dados usadas na API do serviço Web do Gerenciamento de Fluxo de Dados.

CIProperties

`CIProperties` é uma coleção de coleções. Cada coleção contém propriedades de um tipo de dado diferente. Por exemplo, pode haver uma coleção `dateProps`, `strListProps`, `xmlProps` e assim por diante.

Cada coleção de tipo contém propriedades individuais do tipo fornecido. Os nomes dos elementos dessas propriedades são os mesmos do contêiner, mas no singular. Por exemplo, `dateProps` contém elementos `dateProp`. Cada propriedade é um par nome-valor.

Consulte `CIProperties` na *Referência de Esquema do HP Discovery and Dependency*.

IPList

Uma lista de elementos `IP`, sendo que cada um contém um endereço IPv4.

Consulte `IPList` na *Referência de Esquema de Mapeamento do HP Discovery and Dependency*.

IPRange

Um `IPRange` tem dois elementos, `Start` e `End`. Cada um contém um elemento `Address` que é um endereço IPv4.

Consulte `IPRange` na *Referência de Esquema de Mapeamento do HP Discovery and Dependency*.

Scope

Dois `IPRanges`. `Exclude` é uma coleção de `IPRanges` a excluir do trabalho. `Include` é uma coleção de `IPRanges` a incluir no trabalho.

Consulte `Scope` na *Referência de Esquema de Mapeamento do HP Discovery and Dependency*.

Gerenciando métodos de trabalho de descoberta

activateJob

Ativa o trabalho especificado.

Consulte "Amostra de código" na página 240

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
JobName	O nome do trabalho.

deactivateJob

Desativa o trabalho especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
JobName	O nome do trabalho.

dispatchAdHocJob

Distribui um trabalho na sonda ad-hoc. O trabalho precisa estar ativo e conter o EC acionador especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
JobName	O nome do trabalho.
CIID	A ID do EC acionador.
ProbeName	O nome da sonda.
Tempo limite	Em milissegundos

getDiscoveryJobsNames

Retorna a lista de nomes de trabalho.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.

Saída

Parâmetro	Comentário
strList	A lista de nomes de trabalho.

isJobActive

Verifica se o trabalho está ativo.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
JobName	O nome do trabalho a verificar.

Saída

Parâmetro	Comentário
JobState	Verdadeiro se o trabalho está ativo.

Gerenciando métodos de acionador

addTriggerCI

Adiciona um novo EC acionador ao trabalho especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
JobName	O nome do trabalho.
CIID	A ID do EC acionador.

addTriggerTQL

Adiciona um novo TQL acionador ao trabalho especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
JobName	O nome do trabalho.
TqlName	O nome do TQL a adicionar.

disableTriggerTQL

Impede o TQL de acionar o trabalho, mas não o remove permanentemente da lista de consultas que acionam o trabalho.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
JobName	O nome do trabalho.

removeTriggerCI

Remove o EC especificado da lista de ECs que acionam o trabalho.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
JobName	Nome do trabalho.
CIID	A ID do EC acionador.

removeTriggerTQL

Remove o TQL especificado da lista de consultas que acionam o trabalho.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
JobName	Coleção de nomes de trabalhos a verificar.
CIID	A ID do TQL a remover.

setTriggerTQLProbesLimit

Restringe as sondas em que o TQL está ativo no trabalho à lista especificada.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
JobName	O nome do trabalho.
tqjName	O nome do TQL.
probesLimit	A lista de sondas para as quais o TQL está ativo.

Métodos de dados de sonda e domínio

getDomainType

Retorna o tipo de domínio.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
domainName	O nome do domínio.

Saída

Parâmetro	Comentário
domainType	O tipo de domínio.

getDomainsNames

Retorna os nomes dos domínios atuais.

Consulte "Amostra de código" na página 240

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.

Saída

Parâmetro	Comentário
domainNames	A lista de nomes de domínios.

getProbeIPs

Retorna os endereços IP da sonda especificada.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
domainName	O domínio a verificar.
probeName	O nome da sonda usada nesse domínio.

Saída

Parâmetro	Comentário
probelIPs	A "IPList" dos endereços na sonda.

getProbesNames

Retorna os nomes das sondas no domínio especificado.

Consulte "Amostra de código" na página 240

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
domainName	O domínio a verificar.

Saída

Parâmetro	Comentário
probesName	A lista de sondas no domínio.

getProbeScope

Retorna a definição de escopo da sonda especificada.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
domainName	O domínio a verificar.
probeName	O nome da sonda.

Saída

Parâmetro	Comentário
probeScope	O "Scope" da sonda.

isProbeConnected

Verifica se a sonda especificada está conectada.

Consulte "Amostra de código" na página 240

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.

Parâmetro	Comentário
domainName	O domínio a verificar.
probeName	A sonda a ser verificada

Saída

Parâmetro	Comentário
isConnected	Verdadeiro se a sonda está conectada.

updateProbeScope

Define o escopo da sonda especificada, substituindo o escopo existente.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
domainName	O domínio.
probeName	A sonda a ser atualizada.
newScope	O "Scope" a definir para a sonda.

Métodos de dados de credenciais

addCredentialsEntry

Adiciona uma entrada de credenciais ao protocolo especificado para o domínio especificado.

Consulte "Amostra de código" na página 240

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
domainName	O domínio a atualizar.
protocolName	O nome do protocolo.
credentialsEntryParameters	A coleção "CIProperties" das novas credenciais.

Saída

Parâmetro	Comentário
credentialsEntryID	A ID do EC da nova entrada de credencial.

getCredentialsEntriesIDs

Retorna os IDs das credenciais definidas para o protocolo especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
domainName	O domínio para o qual obter as credenciais.
protocolName	O nome de um protocolo usado nesse domínio.

Saída

Parâmetro	Comentário
credentialsEntryIDs	A lista de IDs de credenciais para o protocolo no domínio.

getCredentialsEntry

Retorna as credenciais definidas para o protocolo especificado. Os atributos criptografados são retornados vazios.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
domainName	O domínio para o qual obter as credenciais.
protocolName	Um protocolo usado nesse domínio.
credentialsEntryID	A ID da credencial a obter.

Saída

Parâmetro	Comentário
credentialsEntryParameters	A coleção "CIProperties" das credenciais.

removeCredentialsEntry

Retorna as credenciais especificadas do protocolo.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
domainName	O domínio.

Parâmetro	Comentário
protocolName	Um protocolo usado no domínio.
credentialsEntryID	A ID da credencial a remover.

updateCredentialsEntry

Define novos valores para propriedades da entrada de credenciais especificada.

As propriedades existentes são excluídas e essas propriedades são definidas. Qualquer propriedade cujo valor não está definido nessa chamada permanece indefinido.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
domainName	O domínio no qual atualizar credenciais.
protocolName	Um protocolo usado no domínio.
credentialsEntryID	A ID da credencial a atualizar.
credentialsEntryParameters	A coleção "CIProperties" a definir como propriedades para as credenciais.

Métodos de atualização de dados

rediscoverCIs

Localiza os acionadores que descobriram os objetos de EC especificados e executa novamente esses acionadores.

Os **rediscoverCIs** são executados de modo assíncrono. Chame **checkDiscoveryProgress** para determinar quando a redescoberta estará concluída.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
CmdbIDs	Coleção de IDs dos objetos a redescobrir.

Saída

Parâmetro	Comentário
isSucceed	Verdadeiro se a redescoberta de ECs foi bem-sucedida.

checkDiscoveryProgress

Retorna o andamento da chamada **rediscoverCIs** mais recente nos IDs especificados. A resposta

será um valor de 0 a 1. Quando a resposta for 1, a chamada **rediscoverCIs** terá sido concluída.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
CmdbIDs	Coleção de IDs dos objetos na chamada de redescoberta a rastrear.

Saída

Parâmetro	Comentário
andamento	Um trabalho concluído tem um andamento de 1. Trabalhos não concluídos têm uma fração menor que 1.

rediscoverViewCIs

Localiza os acionadores que criaram os dados para popular a visualização especificada e executa novamente esses acionadores.

Os **rediscoverViewCIs** são executados de modo assíncrono. Chame **checkViewDiscoveryProgress** para determinar quando a redescoberta estará concluída.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
viewName	As visualizações a verificar.

Saída

Parâmetro	Comentário
isSucceed	Verdadeiro se a redescoberta de ECs foi bem-sucedida.

checkViewDiscoveryProgress

Retorna o andamento da chamada **rediscoverViewCIs** mais recente na visualização especificada. A resposta é um valor entre 0 e 1. Quando a resposta for 1, a chamada **rediscoverCIs** terá sido concluída.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 1.
viewName	A coleção de visualizações a verificar.

Saída

Parâmetro	Comentário
andamento	Um trabalho concluído tem um andamento de 1. Trabalhos não concluídos têm uma fração menor que 1.

Amostra de código

```
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 protocol
        addNTCMDCredentialsEntry();
    }

    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(
                new GetDomainsNamesRequest(cmdbContext)).
                getDomainNames();
    }
}
```



```
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create
credentials");
            return;
        }
        String domainName = domains.getStrValue(0);
        // Create properties with one byte param
        CIProperties newCredsProperties = new CIProperties();

        // Add password property - this is of type bytes
        newCredsProperties.setBytesProps(new BytesProps());
        setPasswordProperty(newCredsProperties);

        // Add user & domain properties - these are of type string
        newCredsProperties.setStrProps(new StrProps());
        setStringProperties("protocol_username", "test user",
newCredsProperties);
        setStringProperties("ntadminprotocol_ntdomain",
            "test doamin", newCredsProperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(
            new AddCredentialsEntryRequest(domainName,
            "ntadminprotocol", newCredsProperties, cmdbContext));
        System.out.println("new credentials craeted for domain: " +
domainName + " in ntcmd protocol");
    }

    private static void setPasswordProperty(CIProperties
newCredsProperties) {
        BytesProp bProp = new BytesProp();
        bProp.setName("protocol_password");
        bProp.setValue(new byte[] {101,103,102,104});
        newCredsProperties.getBytesProps().addBytesProp(bProp);
    }

    private static void setStringProperties(String propertyName,
String value, CIProperties newCredsProperties) {
        StrProp strProp = new StrProp();
        strProp.setName(propertyName);
        strProp.setValue(value);
        newCredsProperties.getStrProps().addStrProp(strProp);
    }

    private static void getProbesInfo(DiscoveryService
discoveryService) throws Exception {
        GetDomainsNamesResponse result =
discoveryService.getDomainsNames(new GetDomainsNamesRequest
```

```

(cmdbContext ));
    // Go over all the domains
    if (result.getDomainNames().sizeStrValueList() > 0) {
        String domainName =
            result.getDomainNames().getStrValue(0);
        GetProbesNamesResponse probesResult =
            discoveryService.getProbesNames(
                new GetProbesNamesRequest(domainName,
cmdbContext));
        // Go over all the probes
        for (int i=0; i<probesResult.getProbesNames
().sizeStrValueList(); i++) {
            String probeName = probesResult.getProbesNames
().getStrValue(i);
            // Check if connected
            IsProbeConnectedResponse connectedRequest =
                discoveryService.isProbeConnected(
                    new IsProbeConnectedRequest(
                        domainName, probeName, cmdbContext));
            Boolean isConnected = connectedRequest.getIsConnected
();

            // Do something ...
            System.out.println("probe " + probeName + "
isconnect=" + isConnected);
        }
    }

    private static DiscoveryService getDiscoveryService() throws
Exception {
        DiscoveryService discoveryService = null;
        try {
            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub =
                new DiscoveryServiceStub(url.toString());

            // Authenticate info
            HttpTransportProperties.Authenticator auth =
                new HttpTransportProperties.Authenticator();
            auth.setUsername (USERNAME);
            auth.setPassword (PASSWORD);
            serviceStub._getServiceClient().getOptions().setProperty(
                HTTPConstants.AUTHENTICATE, auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service
", e);
        }
    }

```

```
        return discoveryService;  
    }  
}
```

