Obsah

1. Zadání

Prozkoumejte, jak funguje plugin install v eclipse 3.3. Porovnejte s předcházející verzí eclipse, tj. verze 3.2.

2. Analýza

2.1. Balíky z veřejného API

2.1.1. Balík org.eclipse.update.configuration

Balík zastřešuje rozhraní a třídy pro práci s feature a pluginy, které máme nainstalovány.

2.1.2. Balík org.eclipse.update.configurator

Balík zastřešuje rozhraní a třídy pro práci s tzv. runtime daty (konfigurační data), která Eclipse za svého běhu využívá. Jakákoli změna v těchto datech se projeví až po restartu aplikace.

2.1.3. Balík org.eclipse.update.core

Balík zastřešuje rozhraní a třídy pro vlastní rozšíření (změnu) základního průběhu instalace a aktualizační služby (lze však využít jen pro vlastní feature, nikoli upravit instalační proces Eclipse obecně). Kromě toho můžeme pomocí příslušných tříd získat potřebné informace o nainstalovaných feature, jejich závislostech apod.

2.1.4. Balík org.eclipse.update.core.model

Balík obsahuje třídy popisující vzorový model pro rozšíření základní instalace a aktualizačních služeb. Jedná se v podstatě o ukázkovou implementaci balíku předchozího.

2.1.5. Balík org.eclipse.update.operations

Balík zastřešuje rozhraní pro operace vykonávající jednotlivé instalační a aktualizační akce bez využití uživatelského rozhraní update manažeru. Obsahuje také třídu *OperationsManager*, což je jakýsi vstupní bod pro práci s jednotlivými operacemi. Pomocí této třídy lze vytvářet operace pro instalaci, odinstalaci apod. Lze také pomocí této třídy získat validátor operací, kterým lze validovat (kontrolovat) prováděné operace.

2.1.6. Balík org.eclipse.update.search

Balík zastřešuje rozhraní a třídy vykonávající vyhledávání a filtrování feature při instalaci nebo jejich aktualizaci. Centrálním prvkem je třída *UpdateSearchRequest*.

2.1.7. Balík org.eclipse.update.standalone

Balík zastřešuje třídy pro vykonávání instalačních, aktualizačních a podobných akcí, které lze volat přímo z příkazové řádky (aplikace), tedy bez nutnosti využití uživatelského rozhraní update manažeru.

Všechny výše uvedené balíky jsou součástí tzv. dočasného API, které je stále ve fázi vývoje a kterákoli uvedená část se může kdykoli změnit.

2.2. Balíky z interního API

Mimo výše uvedené balíky z veřejného API jsou dostupné ještě další balíky, které jsou už čistě interní a není tedy dostupná ani řádná dokumentace. Při procházení zdrojového kódu Eclipse navíc často narazíme na fakt, že se třídy z veřejného API odkazují na třídy z API interního.

2.3. Porovnání verzí Eclipse 3.3 a 3.2

Změn mezi těmito verzemi je několik. Uvedu pouze změny, které se nějakým způsobem týkají naší oblasti, tj. feature a pluginů.

1. Byl změněn způsob zavádění jednotlivých bundlů – nás se přímo netýká.
2. Změna tzv. Lazy aktivace v případě bundlů – Eclipse 3.3 implementuje OSGi 4.1, a proto tato změna. Z důvodů zachování kompatibility se standardem – nás se přímo netýká.

Kompletní přehled změn je možné shlédnout v dokumentaci, která je součástí Eclipse SDK. Jedná se konkrétně o část *Platform Plug-in Developer Guide -> 3.3 Plug-in Migration Guide -> Migrating to Eclipse 3.3 from 3.2 -> Incompatibilities* .

2.4. Shrnutí analýzy

Na základě porovnání verzí Eclipse 3.2 a 3.3 se odvážím tvrdit, že žádná z provedených změn, které jsem výše uvedl, nijak nemění zamýšlené využití našeho pluginu (plugin pro kontrolu verzí feature a pluginů, které jsou instalovány / aktualizovány do Eclipse). Zároveň jsem též nenarazil na žádné změny, které by napomohly snadnější implementaci. Část API, kterou bychom rádi využili pro implementaci, je stále označena jako dočasná a její nezanedbatelná část je navíc označena jako interní API.

3. Jak funguje plugin install a další operace

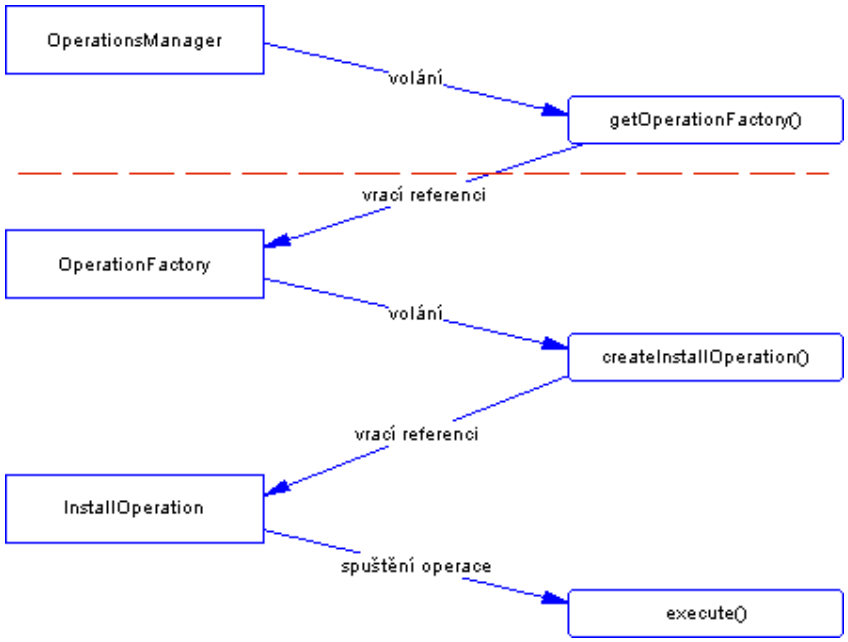
Vytváření operací instalace, aktualizace, operace změny verze používané feature apod. je zajištěno třídou *OperationFactory* (balík *org.eclipse.update.internal.operations*). Třída obsahuje metody *createNázevOperace()* (např. *createInstallOperation()*), které zajišťují správné vytváření požadovaných operací. Parametry jednotlivých metod se liší, ale ve všech se objevuje parametr pro feature nebo seznam feature, nad kterým bude operace pracovat. Metody vracejí referenci na objekt příslušné třídy, které danou operaci implementují.

Třídy těchto operací se nacházejí v balíku *org.eclipse.update.internal.operations* . Třídy dědí od třídy *FeatureOperation* a implementují příslušné rozhraní (např. třída *InstallOperation* implementuje rozhraní *IInstallFeatureOperation*). Operace se spouštějí pomocí metody *execute(IProgressMonitor pm, IOperationListener listener)*.

Ještě jsem neuvedl, jak se získá reference na objekt třídy *OperationFactory*. Zde využijeme třídu *OperationsManager* (balík *org.eclipse.update.operations*) a zavoláme její metodu *getOperationFactory()*, která příslušnou referenci vrátí.

3.1. Zjednodušený model operace instalace

Na obrázku 1 je znázorněn průběh operace instalace. Hlavním prvkem je, jak je vidět z obrázku, třída *OperationsManager*. Čárkovaná čára naznačuje předěl mezi veřejným a interním API. V dokumentaci je možné pro uvedené třídy dohledat odpovídající rozhraní z API veřejného.



Obrázek 1

4. Navrhované řešení

Pokud budeme chtít implementovat původní představy funkčnosti našeho pluginu, tj. poskytnout uživateli příjemné uživatelské rozhraní, pomocí kterého bude moct při nalezení konfliktu provést návrat feature ke starší verzi, bude nutné (vhodné) navrhnout a implementovat vlastní API. API by vycházelo z toho, které je nyní implementováno v Eclipse. Veškerá požadovaná funkčnost je zde zabudovaná. Bylo by tedy nutné zanalyzovat příslušné třídy a metody, které bychom následně využili při implementaci vlastního API.