Mendelova univerzita v Brně
Provozně ekonomická fakulta

Eclipse plugin pre spúšťanie pracovných postupov projektu Newcastle

Bakalářská práce

Vedoucí práce:
Ing. Oldřich Faldík

Denis Richtárik

Brno 2016

Ďakujem Ing. Oldřichu Faldíkovi za vedenie mojej práce, rady a pripomienky. Tiež ďakujem Mgr. Štefanovi Bunčiakovi za trpezlivosť, ochotu a vedomosti, ktoré som vďaka nemu nadobudol. V neposlednom rade ďakujem Ing. Janu Kolomazníkovi, Ph.D. za konzultácie a podporu.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Eclipse plugin pre spúšťanie pracovných postupov projektu Newcastle**
vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*
Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.
Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

v Brne, 23. 5. 2016 .....

Abstract

Richtárik R. Eclipse plugin for triggering Project Newcastle workflows. Bachelor thesis. Brno 2016

The bachelor thesis deals with the design and implementation of Eclipse plugin for triggering builds on a remote PNC server for the company Red Hat. The theoretical part of this thesis describes the current state of the applied technologies and the design of custom solution based on specified requirements. The practical part involes the implementation of the designed solution. The plugin is developed in Java programming language using the building tool Maven.

Keywords Java, Eclipse, plugin, Maven

Abstrakt

Richtárik R. Eclipse plugin pro spúšťanie pracovných postupov projektu Newcastle. Bakalárska práca. Brno 2016

Bakalárska práca sa zaoberá návrhom a implementáciou Eclipse pluginu pre spúšťanie zostavení na vzdialenom PNC serveri pre firmu Red Hat. Teoretická časť tejto práce opisuje súčasný stav použitých technológií a návrh vlastného riešenia založeného na stanovených požiadavkách. V praktickej časti je zahrnutá implementácia navrhnutého riešenia. Plugin je realizovaný v programovacom jazyku Java a používa zostavovací nástroj Maven.

Kľúčové slová Java, Eclipse, plugin, Maven

Obsah

1 Úvod a cieľ práce

1.1 Úvod

Zadávateľom tejto práce je firma Red Hat, ktorá je najväčšou open source firmou na svete. Do dnešného dňa pracovala na viac ako 1 000 000 projektoch. Jeden z nich je aj projekt Newcastle, ktorý zjednodušuje nasadzovanie produktov na predaj. V súčasnosti projektu chýba plugin, ktorý by podporoval spúšťanie zostavení (builds) priamo vo vývojovom prostredí Eclipse. Tento plugin umožní komunitným vývojárom analýzu ich projektových závislostí a zosynchronizovať ich s verziami určenými na výsledné produkty.

Z tohto dôvodu vznikla iniciatíva zo strany firmy Red Hat vyriešiť daný problém prostredníctvom zadania bakalárskej práce.

1.2 Cieľ práce

Hlavným cieľom tejto práce je návrh a implementácia nového Eclipse pluginu podľa požiadaviek zadania. Pomocou tohto pluginu bude vývojár schopný spúšťať zostavovací proces na vzdialenom serveri. Vstupné parametre budú musieť spĺňať podmienky pracovných postupov projektu Newcastle.

Ďalším cieľom je tiež zoznámiť sa s konceptom a architektúrou open source projektov pod záštitou projektu Newcastle a s vývojom Eclipse pluginov.

Všetky zdrojové kódy budú obsahovať javadoc dokumentáciu pre zjednodušenie orientácie v projekte a pre možné úpravy v budúcnosti.

Na záver bude tento projekt nahraný na verejný GitHub repozitár.

2 Súčasný stav

V kapitole Súšasný stav je na úvod napísané, čo je to Project Newcastle a s akými technológiami sa pracovalo na tejto práci. Slúži na vysvetlenie najdôležitejších pojmov a na vysvetlenie problematiky, ktorou sa táto práca zaoberala. V neposledom rade slúži ako úvod k návrhu a implementácií projektu, ktoré sú samostatne opísané v nasledujúcich dvoch kapitolách.

Snahou bolo popísať tie najdôležitejšie techlonógie prehľadne a stručne tak, aby to čo najviac vyhovovalo účelu úvodnej kapitoly tejto práce.

2.1 Projekt Newcastle

Project Newcastle je interný projekt firmy Red Hat, ktorý rieši problémy tzv. "produktizácie". Produktizáciu môžeme chápať ako transformáciu projektu z komunitného na platený produkt. Je dôležité nasadzovať produkty rýchlo a s prehľadom, čo umožňuje vývojárom stabilnejší a ľahší postup práce a komunikácie. Vďaka nemu komunitní JBoss vývojári sú schopní využiť jeho projekty na uľahčenie práce a zjednodušenie nasadzovania produktov do finálneho štádia či predaja.

Projekt Newcastle je systém určený na menežovanie, spúšťanie a sledovanie zostavení (build). Projekt je vyvíjaný firmou Red Hat a jej zamestnancami z celého sveta. Úzko súvisí s pojmom Continuous delivery, ktorý je opísaný v nasledujúcej sekcii. Zastrešuje niekoľko projektov ako napríklad Dependency Analysis, PNC, pnc-e2e, repour a ďalšie. Medzi tieto projekty sa po dokončení zaradí aj tento plugin.

2.2 Continuous delivery

Cieľom continuous delivery je čo možno najviac zautomatizovať, zefektívniť a zrýchliť vývoj konkrétneho softwaru. Je to zabezpečenie vysokej kvality kódu v čo najkratšom čase. Podstatou je časté integrovanie nových verzií systému od vývojárov medzi sebou a zákazníkom. Časté integrovanie softwéru v zmysle continuous delivery väččinou znamená niekoľkokrát denne alebo týždenne.

Veľkou výhodou je rýchla spätná väzba od zákazníka. Spätnú väzbu od zákazníka dostaneme v takomto prípade oveľa rýchlejšie, ako keď sa nasadzujú zmeny alebo pridávajú nové prvky napríklad raz za pol roka. Pojem continuous delivery úzko súvisí s pojmom continuous integration.

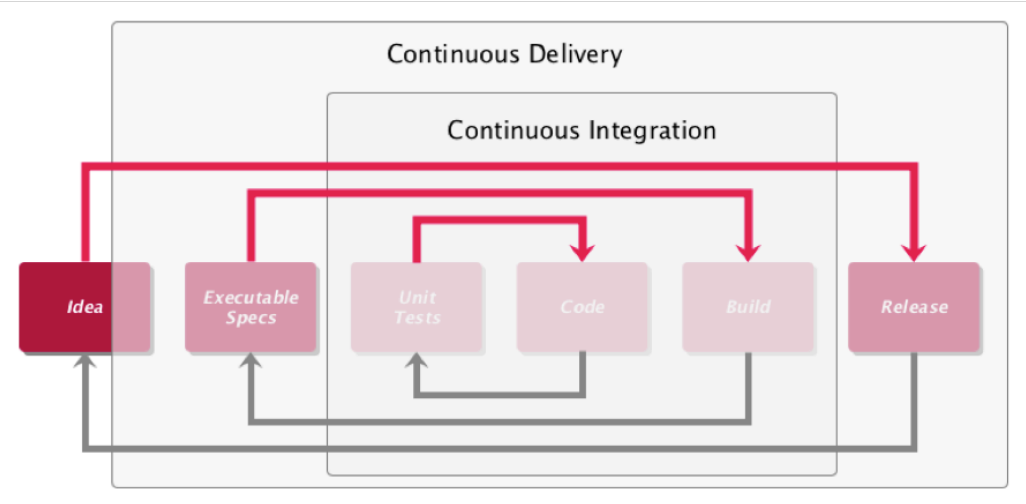
Vznik

Pojem Continuous delivery bol zavedený v roku 2010 autormi Jez Humble a David Farley, ktorí o ňom napísali rozsiahlu knihu. Koncept Continuous delivery je relatívne nový a nemá za sebou dlhú históriu.

Continuous integration

Continuous Integration je spôsob softvérového vývoja, kde členovia tímu integrujú svoju prácu pravidelne, obvykle každá osoba integruje aspoň denne - čo vedie k viacerým integráciam za deň. Každá integrácia je skontrolovaná automatickým zostavením (vrátane testov), aby sa zistili prípadné integračné chyby tak rýchlo ako je to len možné. Veľa tímov vďačí tomuto prístupu za výrazné zníženie integračných problémov a umožnenie vyvíjať súdržnú aplikáciu čo najrýchlejšie (Fowler, 2006).

Na nasledujúcom obrázku 1 je znázornený rozdieľ medzi continuous delivery continuous integration.



Obrázok 1: Continuous delivery vs. continuous integration (Danciu, 2014)

2.3 Programovací jazyk Java

Hlavným cielom Javy je, aby bolo možné písať programy, ktoré budú bežať na najrôznejších počítačových systémoch a počítačovo ovládaných zariadeniach. To je niekedy nazývane "napísať raz, spustiť kdekoľvek" (Deitel, 2012).

Programovacie jazyky sú vyvýjané pre určité účely. Každý programovací jazyk má svoje špecifické vlastnosti, ktoré programátorom uľahčujú ich prácu. Nedá sa povedať, ktorý programovací jazyk je najlepší.

Java je objektovo orientovaný programovací jazyk vyvinutý firmou Sun Microsystems v roku 1995. Dnes patrí medzi najpopulájnejšie jazyky vďaka svojej jednoduchosti a faktu, že má syntax založenú na veľmi známych jazykoch z rodiny C. Jednou z jeho najväčších výhod je jeho prenositeľnosť. Každý program napísaný v Jave je sputiteľný na akejkoľvek platforme.

Pre zefektivnenie práce vývojárov sa vytvárajú programovacie prostredia, podporujúce základné programovacie jazyky. Jedná sa o vývojové prostredia, takzvané IDE. Vývojovým prostredím pre Javu je napr.: Eclipse alebo Netbeans. Pre ďalšie

rozšírenie funkčnosi sa využíva nesamostatne pracujúci softvér, plugin, ktorý využíva API. API je informatické rozhranie určujúce, akým spôsobom budú privolávané funkcie danej knižnice.

História

V roku 1991, James Gosling, Bill Joy, Mike Sheridan a Patrick Naughton z firmy Sun Microsystems započali projekt, ktorý mal za cieľ vytvoriť systém pre domáce spotebiče. Najskôr pracovali v programovacom jazyku C++, ktorý im časom prestal vyhovovať, preto vypracovali nový jazyk, ktorý suvisí s C a C ++, ale je organizovaný trochu odlišne, rada aspektov z C a C ++ je vynechaná a pár nápadov z iných jazykov je naopak pridaných. Má sa jednať o výrobný jazyk, nie o výskumný jazyk (Gosling, 2014).

V priebehu rokov sa vyvíjal až do dnešnej podoby, kedy sa používa v počítačoch, mobilných aplikáciach, čipových kartách a mnohých ďalších zariadeniach. Celá táto technológia sa nazýva platforma Java. Od roku 2007 je tento programovací jazyk ďalej vyvíjaný ako open source, to znamená, že je jej zdrojový kód prístupný pre úpravy, či prácu s ním (za splnenia istých podmienok). Najnovšou verziou je Java 8.

Zloženie javy

Programovací jazyk Java sa skládá z niekoľko ďalších programov, ktoré ovplyvňujú celkový chod, sú to napríklad JVM, JDK, JRE alebo sada knižníc. Každý napísaný program má svoj zdrojový kód, ktorý je uložený v projektoch. Zdrojový kód je spracovávaný interpretom, ktorý ho preloží do jednoduchšieho strojového kódu. V programovacom jazyku Java sa jedná o medzikód: Java bytecode (Liang, 2015). Java bytecode je mezikód, binárny kód, má jednoduchšiu inštrukčnú sadu a priamo podporuje objektové programovanie. Je spojkou mezi zdrojovým kódom a strojovým kódom.

Java Virtual Machine

Java bytecode je vďaka svojej jednoduchosti interpretovaný pomocou JVM - Java Virtual Machine. Jedná sa o virtuálny stroj, ktorý slúži na zjednodušenie zdrojového kódu a na jeho prepísanie na kód strojový. Strojový kód je ešte zjednodušený pomocou assembleru - jednoduchého prekladového nástroju, ktorý prekladá moduly kódu a linkeru, ktorý spája jednotlivé moduly na najzákladnejší binárny kód. S týmto kódom pracuje procesor počítača, ktorý vie vykonávať len obmedzený počet inštrukcií, ktoré sú uložené ako čísla - sekvencie bitov. Každý program je uložený v tomto najzákladnejšom binárnom kóde, aby mohol byť na procesore spustený.

Java development kit

Pre vývoj aplikácií je potrebný nástroj JDK - Java development kit, skladá sa z JRE (runtime prostredia Javy), Java compileru (prekladač zdrojového kódu do medzikódu) a Java API - Application Programming Interface. Jedná sa o programovacie rozhranie, ktoré určuje, akým spôsobom budú vyvolávané funkcie knižníc piamo zo zdrojového kódu programu.

Java runtime environment

JRE - Java runtime environment je prostredie, ktoré sprostredkováva podprogramy, knižnice programovacieho jazyka a typové kontroly priamo pri behu programu. Sada knižníc, čo je kolekcia predpísaných kódov, ktoré sú často používané, pomôžu programátorom ušetriť čas tým, že danú časť programu nemusia písať, ale stačí ju privolať z knižnice.

Garbage collector

Java využíva Garbage collector, zpôsosb automatickej správy pamäte. Garbage collector je spravovaný JVM a funguje na princípe prechádzania programu a odstraňovania už nepoužívaných úsekov pamäti. Keďže sa jedná o automatické čistenie, programátor sa nemusí o čistenie pamäte starať. V programovacom jazyku Java sa jedná o tzv. generačný garbage collection, ktorý si pamäť rozdeluje na úseky/generácie. Objekty v pamäti sú ukladané do týchto generácií podľa veku a presúvajú sa po splnení určitých podmienok do starších generacií. V jednotlivých úsekoch môže byť používaný iný algoritmus upratovania. Najstaršia generácia sa nazýva major garbage collection, tvoria ju najpoužívanejšie objekty, ktoré je treba zachovávať. Permanentná generácia obsahuje data požadované JVM pre definovanie knižníc a metód, ktoré sú využívané v aplikácií.

2.4 Standard Widget Toolkit

Standard Widget Toolkit (ďalej len SWT) je grafická knižnica, ktorá sprostredkováva štandardné ovládacie prvky užívateľského rozhrania ako napríklad zoznamy, menu, typy písma a farby (Mcaffer, 2010). SWT využíva grafické prvky toho konrétneho operačného systému, na ktorom je implementovaný a vďaka tomu, že je dostupná na mnohých operačných systémoch, aplikácie, ktoré požívajú SWT sú prenostiteľné na všetky tieto OS.

SWT pristupuje ku grafickým prvkom pomocou Java Native Interface (JNI) rámcu (framework). JNI je programovavý rámec, ktorý umožňuje Java kódu bežiacom v JVM zavolať a byť zavolaným natívnymi aplikáciami a knižnicami napísanými v iných jazykoch ako napríklad C, C++ či assembler (Vogel, 2016).

Eclipse využíva SWT ako východziu knižnicu grafického rozhrania. Ak chce vývojár navrhnúť plugin, ktorý bude rozširovať Eclipse IDE, musí použiť SWT pre-

tože Eclipse pre svoje vlastné grafické rozhranie SWT využíva. Na tejto práci sa preto pracovalo s mnohými SWT grafickými prvkami.

JFace

Radšej ako písať stále ten istý SWT kód dokola, dizajnéri Eclipse vytvorili JFace. Táto knižnica ponúka veľa skratiek okolo mnoho úkonov, ktoré môžu zaberať príliž veľa času používaním samotného SWT.

JFace však nie je náhrada za SWT a mnoho grafických uživatelských rozhraní potrebuje vlastnosti z oboch knižníc. Dôležitým príkladom zvýšenej efektívnosti JFace sú udalosti (events), ktoré vo veľa užívateľských rozhraniach používame. Môže to byť napríklad klik myšou, stlačenie klávesy na klávesnici alebo výber nejakého menu, ktoré vykonávajú tú istú funkciu. V SWT sa každá udalosť zaznamenáva a vyhodnocuje samostatne. Avšak JFace povoluje ich kombináciu do jedného jednoduchého objektu, takže sa nemusíme zaoberať tým, ktorá komponenta udalosť vyvolala, ale zameriavame sa na reakciu, ktorú spôsobila. Vďaka tomu sa kód skracuje a zjednodušuje (Scarpino, 2005).

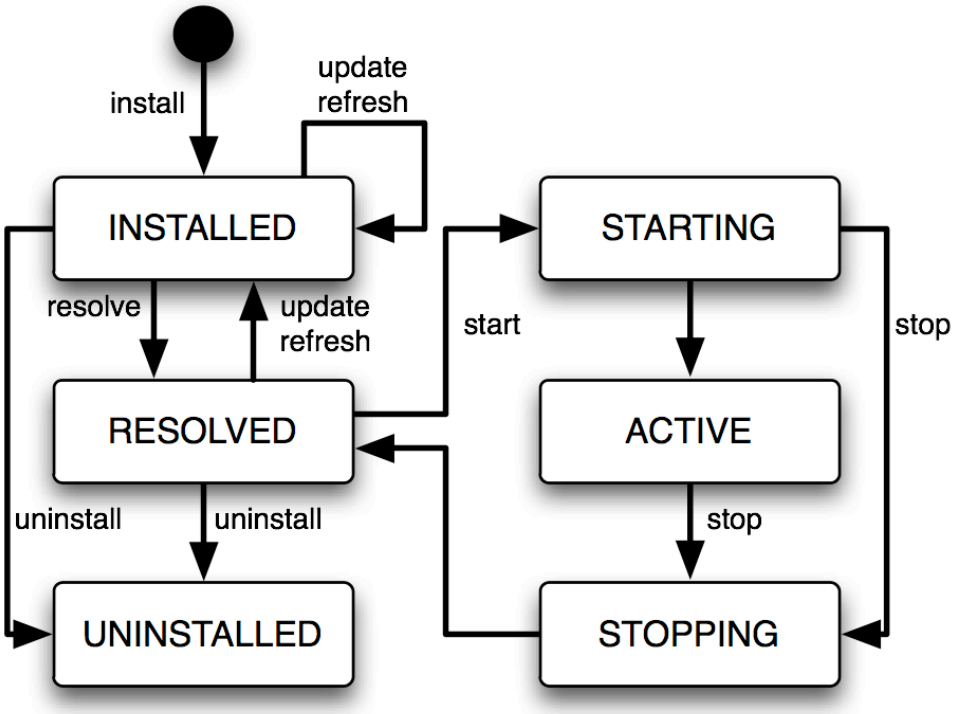
2.5 OSGi

Názov OSGi je skratkou Open Services Gateway initiative. Je to dynamický modulárny systém pre programovací jazyk Java, ktorého najväčším zástupcom je práve platforma Eclipse. Vývoj začal v roku 1999 zoskupením OSGI Alliance a do Eclipse, konrétne vo verzii Eclipse 3.0, sa dostal v roku 2003 implementovaný ako systém Equinox výmenou za proprietárny modulárny systém. Medzi ďalšie často používané implementácie OSGi patrí Apache Felix alebo Knopflerfish. Väčšina aplikačných serverov dnes už podporuje OSGi, poprípade ho plánuje nasadiť.

Dynamický modulárny systém je definovaný ako architektúra pre modulárne vyvíjanie aplikácií, ktorý za behu programu môže rôzne pridávať alebo odoberať moduly. V prípade Eclipse ich označujeme ako bundle alebo plugin. Bundle je .jar archív, ktorý zahŕňa súbor MANIFEST.MF. Manifest obsahuje údaje o jeho verzii, názve, symbolickom názve, verziu bundlu a závislosti na ďalších bundloch a importovaných balíčkoch. Závislosti udávané v manifeste sú veľmi dôležité, pretože bez nich by náš projekt nebolo možné spustiť. Na obrázku 2 je zobrazený životný cyklus bundlu.

2.6 Eclipse

Eclipse je open source komunita pre jednotlivcov i organizácie so záujmom vyvýjania rozšíriteľnej platformy, runtime a nástrojov pre nasadzovanie a riadenie softvéru počas celého jeho životného cyklu. Primárne je učený pre vývoj v programovacom jazyku Java, v ktorom je aj napísaný.



Obrázok 2: Životný cyklus bundlu (VMware Inc., 2011)

Pôvodne bol vytvorený firmou IBM v novembri roku 2001. V roku 2004 bola vytvorená Nádacia Eclipse (Eclipse Foundation) ako nezávislá nezisková korporácia, aby slúžila ako stevard pre Eclipse komunitu. Dnes Eclipse komunita pozostáva z ľudí a organizácií z prierezu softvérového priemyslu. Nadácia Eclipse tiež spravuje IT infraštruktúru pre open source komunitu Eclipse, ako napríklad Git repozitár, Bugzilla databázu, fóra a webovú stránku. Využíva vlastnú implementáciu spomínaného dynamického modulárneho systému OSGi s názvom Equinox (Eclipse, 2016).

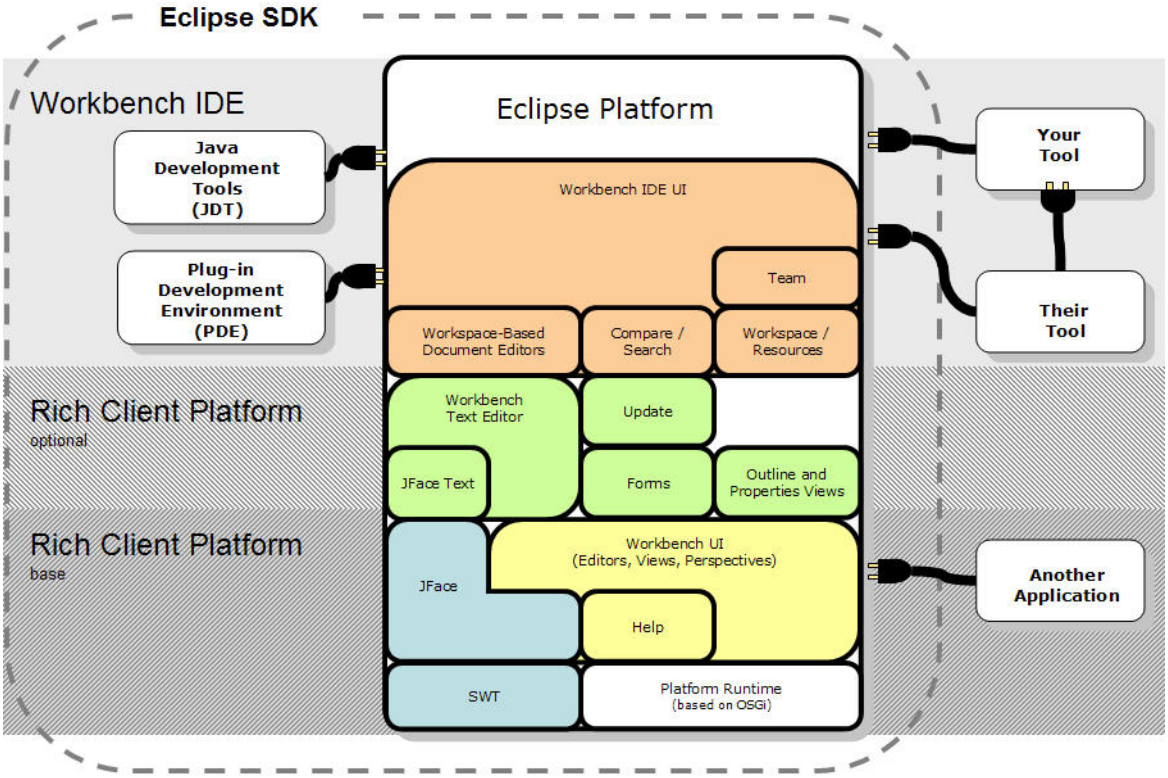
Hodnota tejto platformy je v tom, čo podporuje: rýchly vývoj integrovaných funkcií založených na plugin modeli. Základná verzia Eclipse obsahuje iba integrované prostriedky pre vývoj Javy, všetky rozšírenia sú vykonávané pomocou už spomínaných modulov, pluginov. Na webovej sránke `https://eclipse.org` sú dostupné rôzne distribúcie Eclipse. Napríklad: Eclipse IDE pre Java vývojárov, C/C++ vývojárov, PHP vývojárov, Eclipse RCP a RAP vývojárov, ktorá umožňuje vývoj samostatných rich-client aplikácií, Eclipse PDE (Plug-in Development Environment), ktorá obsahuje minimálnu množinu pluginov, potrebných pre vytváranie, vývíjanie, testovanie, zostavovanie a nasadzovanie Eclipse pluginov. Na tejto bakalárskej práci bola použitá distribúcia Eclipse Eclipse IDE for Eclipse Committers 4.5.2.

Inštalácia Eclipse je veľmi jednoduchá. Stačí z oficiálnych webových stránok stiahnuť a rozbaliť .zip súbor, v ktorom sa nachádza spustiteľný eclipse.exe (Microsoft Windows) súbor. Nové verzie Eclipse vychádzajú každý rok v júni. Poslednou

verziou je Eclipse Mars (4.5).

Architektúra Eclipse

Eclipse platforma je štrukturovaná okolo konceptu pluginov. Čo je to plugin je vysvetlené v nasledujúcej podsekcii. Každý subsystém v platforme je sám definovaný ako množina pluginov, ktoré implementujú nejakú kľúčovú funkcionalitu. Eclipse SDK obsahuje základnú platformu a dva hlavné nástroje užitočné pre vyvíjanie pluginov. Java development tools (JDT) a Plug-in Developer Environment (PDE). Na obrázku 3 je graficky znázornená architektúra Eclipse.



Obrázok 3: Architektúra Eclipse (J2EEBrain)

Plugin

Každá Eclipse aplikácia pozostáva z niekoľko Eclipse komponentov. Tieto softvérové komponenty sú nazývané puginy. Plugin sú štruktúrované bundly kódu alebo dát, ktoré pridávajú funkcionalitu do systému. Toho dosiahneme tým, že vytvoríme vlastný plugin obsahujúci java triedy, ktoré rozširujú Eclipse triedy. Tieto funkctionality môžu byť vo forme knižníc kódu, rozšírení platformy alebo dokumentácie. Takto môžeme vytvoriť napríklad vlastné pohľady, editory, menu, toolbary, kategórie, príkazy a sprievodcov. Táto práca sa zaoberala práve vytvorením sprievodcu zloženého z troch dialógových okien.

Pluginy môžu obsahovať body rozšírenia (extension points), presne definované miesta, kam môžu ostatné pluginy pridávať funkcionalitu.

2.7 Maven

Apache Maven je pokročilý nástroj pre zostavenie softvéru, ktorý pomáha vývojárom počas celého životného cyklu ich projektu. Typickými úlohami zostavovacieho nástroja je kompilácia zdrojového kódu, spúšťanie testov a zabalovanie výsledku do JAR súborov (Maven podporuje aj iné typy balíčkov). Okrem týchto základných procesov, Maven dokáže rôzne ďalšie úkony, ako napríklad vytváranie web stránok, nahrávanie výsledkov zostavenia a generovanie výsledných správ.

Maven umožňuje vývojárovi zautomatizovať proces vytvárania štruktúry počiatočnej zložky pre Java aplikáciu, kompilácie, testov a nasadzovania finálneho produktu. Je implementovaný v samotnej Jave a to ho robí nezávislým na platforme. Java je taktiež najlepším programovacím jazykom pre Maven (Vogel, 2015).

Ciele Mavenu

Primárnym cieľom Mavenu je umožniť vývojárovi v čo najkratšom čase pochopiť stav postupu vývoja. Aby tento cieľ dosiahol, existuje niekoľko oblastí záujmu, s ktorými sa Maven snaží vyposriadať:

- zjednodušiť zostavovací proces
- poskytnúť jednotný zostavovací systém
- poskytnúť kvalitné informácie o projekte
- poskytnúť inštrukcie pre správne postupy vývoja
- umožniť transparentný prechod k novým funkciám (Apache, 2016)

Základným stavebným kameňom každého Maven projektu je súbor pom.xml (Project object model), ktorý nám hovorí, čo a ako zostavovať. Je to XML reprezentácia daného projektu. POM obsahuje všetky základné informácie o projekte, ako aj zoznam pluginov a ich konfigurácií použitých pri zostavení. Na obrázku 4 je znázornené, ako Maven funguje.

Minimálna množina atribútov povinných v každom POM súbore sú tieto tri:

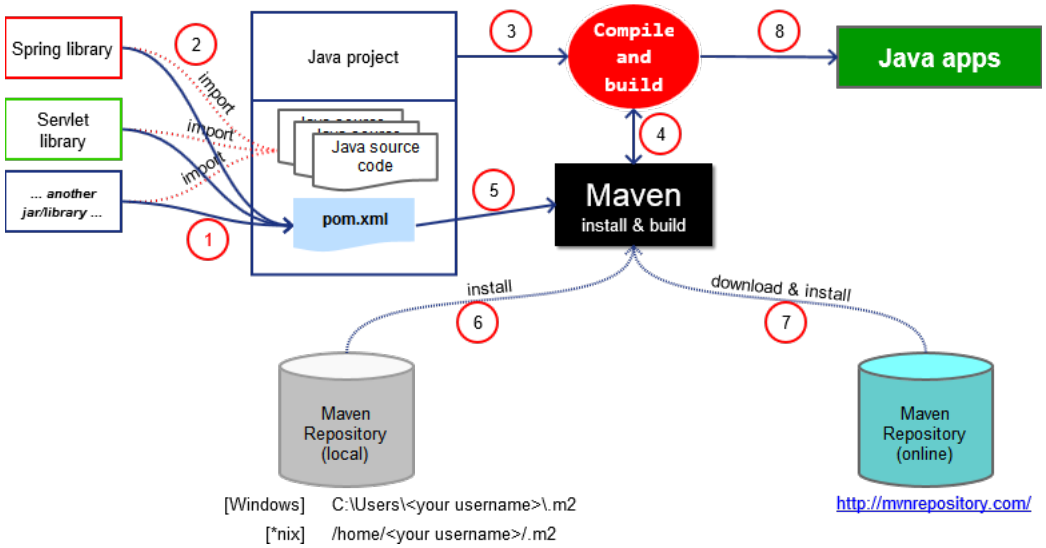
- **groupId**: definuje unikátne meno organizácie alebo skupiny, ktorá vytvorila projekt. Typicky to je reverzné doménové meno. Napríklad org.apache.maaven.plugins je groupid pre všetky Maven pluginy.
- **artifactId**: je unikátne meno projektu,
- **version**: definuje verziu projektu. Pomocou tohoto atribútu sa môžeme rozhodovať, ktorú verziu projektu, pluginu alebo závislosti chceme použiť.

Ďalším dôležitým atribútom POM je **packaging**, ktorý definuje metódu balenia projektu. Implicitne je nastavený na JAR, ale môžeme si predvoliť vlastné metody. Typicky WAR, EAR, pom, atď.

Tycho

Tycho je množina Maven pluginov pre zostavovanie Eclipse komponentov cez Maven zostavovací systém. Tycho podporuje zostavovací proces Eclipse pluginov, OSGi bundlov, Eclipse vlastností (features), aktualizačných stránok (update sites) a produktoch. S kombináciou s continuous integration serverom Tycho umožňuje continuous integration zostavenia. Tycho používa metadáta Eclipse komponentov. Napríklad, Tycho určute závislosti pluginu cez `MANIFEST.MF` súbor toho pluginu.

Hlavnou funkcionalitou Tycho je pomocou `tycho-maven-plugin` pluginu. Tento plugin slúži Mavenu aby rozumel typom balenia (packaging types) ako napríklad eclipse-plugin, eclipse-feature a eclipse-repository (Vogel, 2016).



Obrázok 4: Ako Maven funguje (Munif, 2013)

2.8 Keycloak

Keycloak je SSO (Single sing-on) riešenie pre webové aplikácie, mobilné a RESTful webové služby. Je to autentizačný server, na sa ktorom môžu užívatelia prihlasovať, odhlasovať, registrovať a spravovať ich účty. Užívateľské rozhranie administrátora môže spravovať role a tie mapovať na akékoľvek aplikácie zabezpečené pomocou služby Keycloak. Keycloak server sa dá použiť aj na vykonávanie prihlásení do sociálnych sietí, ako napríklad Google, Facebook, Twitter a podobne.

Základným konceptom v Keacloak je Realm (oblasť). Realm zabezpečuje a spravuje ochranné metadata pre zoznam užívatelov a registrovaných klientov (Apache ASL 2.0, 2016).

Keycloak používa OAuth 2.0 protokol.

OAuth

OAuth znamená Open Authorization. Je open source, vyvinutý v roku 2006 a licencovaný Open Web Foundation. Je to autorizačný protokol, ktorý umožňuje aplikáciam získať limitovaný prístup k užívateľským účtom na HTTP službách ako Facebook, GitHub a podobne. Funguje tak, že predáva právomoc k užívateľskej autentizácii službe, ktorá účet spravuje a autorizuje aplikácie tretích strán k získaniu prístupu k účtu. OAuth poskytuje autorizačné toky k webovým, desktopovým a mobliným aplikáciam a zariadeniam (Anicas, 2014).

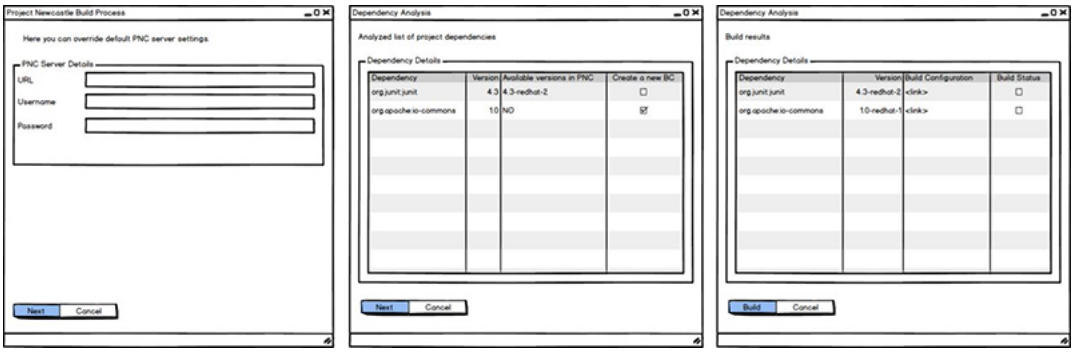
V roku 2012 bola vydaná nová generácia: OAuth 2.0.

3 Vlastné riešenie

Na základe zadania a konzultácie s vedúcim tejto práce z firmy Red Hat, Mgr. Štefanom Bunčiakom, som si naštudoval technológie spomenuté v kapitole Súčasný stav. Pán Bunčiak mi navrhol možný spôsob riešenia tohto problému, ktorý mi znázornil pomocou schématického náčrtu. Tento náčrt som previedol do diagramového obrázku 12 a v nasledujúcich riadkoch bude opísaný jeho obsah.

3.1 Návrh užívateľského rozhrania pluginu

Prvým krokom bolo naštudovať si spôsob fungovania a tvorenia pluginov do Eclipse a následne vytvoriť `JFace Wizard` (sprievodca). Návrh tohoto grafického rozhrania bol vložený v zadaní bakalárskej práce na stránke thesis.redhat.com a v tejto práci je vidieť na obrázku 5.



Obrázok 5: Návrh užívateľského prostredia

Snahou bolo, čo najvernejšie previesť tento návrh do reálnej podoby. Sprievodca musí byť vyvolaný z menu po kliknutí pravým tlačítkom na myši na projekt v zobrazení projektov alebo balíkov. Tento sprievodca má mať tri stránky. V provom má užívateľ zadať URL servera, na ktorý sa prihlasuje, užívateľské meno a heslo. Po prihlásení sa užívateľ môže presnunúť na druhú stránku sprievodcu, kde je `JFace TableViewer` tabuľka, ktorá zobrazuje závislosti Maven projektu, na ktorom bol plugin spustený a ich alternatívne verzie na PNC serveri. Na tretej stánke je podobná tabuľka ako na predošlej stránke. Táto stránka zobrazuje výsledky zostavenia.

3.2 Prihlasovanie

Po návrhu užívateľského rozhrania bude potrebné naprogramovať prihlasovanie pomocou služby Keycloak. Na server, ktorého URL zadá užívateľ, sa odošlú údaje o prihlasovaní. Služba Keycloak po úspšnom prihlásení odpovie odoslaním OAuth 2.0 tokenu. Ak sa nepodarí prihlásiť, vypíše sa chyba. Tento token sa uloží do premennej pre ďalšie použitie.

3.3 Zobrazenie závislostí

Po prihhlásení sa užívateľ môže presunúť na druhú stránku sprievodcu obsahujúcu tabuľku. Prvým krokom bude zobraziť závislosti uložené v POM súbore projektu, na ktorom bol spustený tento plugin. Na to bude potrebné vytvoriť triedu, ktorá nájde absolútnu cestu k pom.xml súboru projektu a uložiť do premennej instancie triedy špeciálneho Maven pluginu, ktorý umožní objektový prístup k atribútom POM súboru.

Zobrazenie závislostí pomocou Dependency Analysis

Ďaľšou častou bude zobrazenie závislostí pomocou programu Dependency Analysis (ďalej len DA). Odoslaním GAV (groupId, artifactId, version) na restový endpoint DA, DA odošle JSON súbor s dostupnými verziami závislostí na serveri. Tie sa zobrazia v tabuľke v samostatnom stĺpci.

3.4 Vytvorenie nového zostavenia

Aby boli potrebné údaje kompletné, užívateľ musí zadať ešte tri príkazy: SCM repository, SCM revision a build script. Prvé dve vloží do tabuľky a build script vloží do textového pola pod tabuľkou.

Plugin následne vytvorí nový Build Configuration (konfiguráciu zostavenia) a odošle na PNC endpoint. Po úspešnom vytvorení odošle trigger (spustenie) build configuration, ktorý ma za úlohu zostavenie spustiť. Po potvrdení zostavenia odošle žiadosť a vypísanie výsledku, ktorý zobrazí na tretej stránke sprievodcu.

4 Implementácia

V tejto kapitole je popísaný postup a spôsob práce na celom projekte. Text je sprevádzaný útržkami kódov použitých pri vývoji a taktiež ilustračnými obrázkami.

4.1 Založenie projektu

V Eclipse sa založil nový projekt ako nový plugin projekt. Názov projektu bol zvolený `com.redhat.dependencyAnalysis`. V ponuke cielenej platformy sa zvolila Eclipse verzie 3,5 a vyššie. V tomto sprievodcovi sa tiež dá zvoliť, či má tento plugin byť Rich client aplikácia a rôzne prednastavené šablony. Keďže tento plugin nemá byť standalone Rich client aplikácia, RCP checkbox sa nechal voľným. Šablonu pre účely tohoto projektu nebolo nutné využiť. Sprievodca bol dokončený a projekt vytvorený tlačítkom finish.

Nový projekt sa vytvoril ako adresár v prednastavenom workspace (pracovný adresár) a v Eclipse sa zobrazil v prehliadači balíčkov alebo prehliadači projektov. V tomto adresári sú všetky základné súbory a zložky vrátane zložky `META-INF`, ktorá obsahuje dôležitý `MANIFEST.MF` súbor, kľúčový pre každý Eclipse plugin projekt.

4.2 Prevedenie na Maven projekt

V prehliadači balíčkov sa projekt pomocou kliku pravého tlačítka na myši v ponuke Configure previedol na Maven projekt voľbou Create to Maven project. Týmto pomocou špeciálneho pluginu m2Eclipse vznikol z obyčajného Eclipse projektu Maven projekt. M2Eclipse výrazne uľahčuje a zjednodušuje komunikáciu medzi Maven a Eclipse a navyše prináša špeciálne Maven menu, aby vývojár nemusel zadávať príkazy typické pre Maven cez terminál. V tomto menu sa dajú spúšťať príkazy, ako napríklad `maven build`, `clean`, `install` alebo `package`. M2Eclipse taktiež vytvorí v adresári projetu POM súbor. Do tohto súboru sám zapíše potrebné informácie a závislosti tak, ako potrebuje pre čo najhladší proces vývoja v Eclipse. Ako základú závislosť na ďalšom maven plugine použil `Apache Maven Compiler Plugin`. Ukážka 1 súboru pom.xml tohoto projektu.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.redhat.dependencyAnalysis</groupId>
  <artifactId>com.redhat.dependencyAnalysis</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
```

```
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.3</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
 </build>
</project>
```

Listing 1: Základný pom.xml súbor

4.3 Nastavenie Maven závislostí

Pravdepodobne najnáročnejšou časťou práce na tomto projekte bolo pochopenie a nastavenie správnych závislostí na ďalších Maven projektoch. Je to dôležité kvôli tomu, aby Maven pri zadaní príkazu `mvn clean install` zahrnul všetky závislosti, či už zadané v súbore `MANIFEST.MF` alebo v súbore `pom.xml`. Použité boli mnohé pluginy, ktoré boli zadané do pom.xml. Základnými boli `Tycho Maven Build Extension`, ďalej `Tycho Target Platform Configuration Plugin`, ktorý zabezpečuje správne zabalenie projektu na každej platforme. Veľmi dôležitý `Apache Maven Dependency Plugin`, ktorý presunie závislosti zadané v `pom.xml` do adresára `lib`. `Tycho P2 Plugin` pomocou zadanej vyššie v súbore napovedá, z akej URL má sťahovať závislosti zadané v manifeste.

4.4 Vytvorenie menu

Nasledujúcim krokom bolo vytvorenie vyskakovacieho menu, ktoré sa má zobraziť v už vytvorenom menu, ktoré sa vyvolá po kliknutí pravým tlačítkom na myši na projekt. Existuje viacero spôsobov ako tento výsledok dosiahnuť a pri práci na tomto projekte bol zvolený spôsob cez Extensions (rozšírenia) v MANIFEST.MF súbore.

Extension sa používa vtedy, ak chce vývojár svojim pluginom rozšíriť chovanie Eclipse. Najskôr bolo potrebné vyhľadať konkrétne body rozšírenia, ktoré bolo potrebné použiť. Na pridanie položky "Dependency Analysis" do vyskakovacieho okna sa musel rozšíriť extension point `org.eclipse.ui.menus`. V záložke Extensions je tlačítko Add, ktorým sa vyhľadal spomínaný extension point a pomocou neho pridali nové menu contribution (príspevok). Menu contribution obsahuje niekoľko nastavení, z ktorých bolo nutné vyplniť jedno a to `locationURI`. `LocationURI` presne popisuje miesto, kde sa bude extension point nachádzať a navyše určité príkazy. Zadalo sa `popup:org.eclipse.ui.navigator.ProjectExplorer#PopupMenu ?after=additions`. `Popup` značí, že menu bude vyskakovacie, `org.eclipse.ui.navigator.ProjectExplorer#PopupMenu` značí, že menu bude

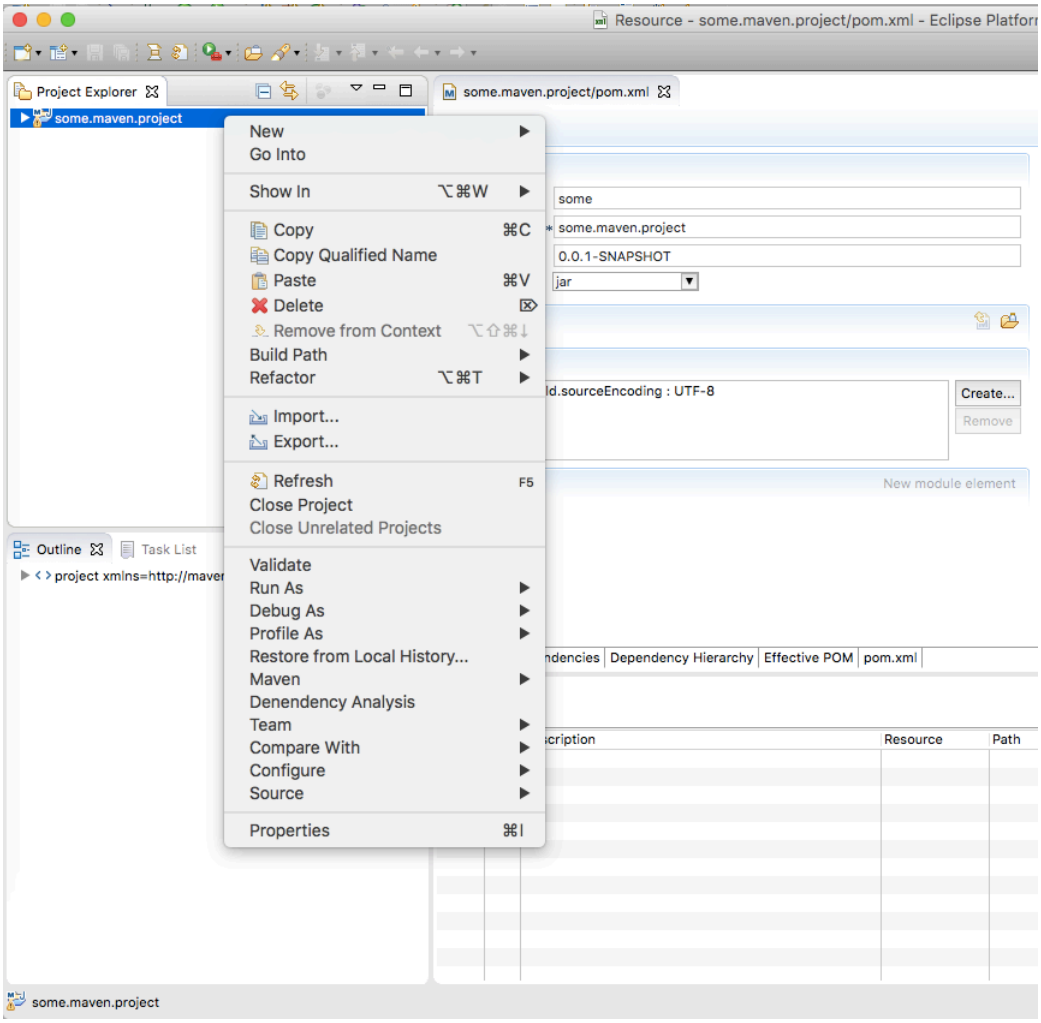
patriť do prehliadača projektov a `after=additions` značí, že menu sa bude nachádzať za skupinou menu nazvaným additions. Pre prehliadač balíčkov bolo `locationURI` nastavené podobne, a to:
`popup:org.eclipse.jdt.ui.PackageExplorer?after=additions`.

Ďalej bolo nastavené, aký typ menu je potrebné priadať. Pre účely tohto pluginu bol vybraný dynamic menu. Tento typ menu má dva atribúty, a to `id` a `class`. `Id` sa vygenerovalo samo a class je trieda, ktorá sa postará o zobrazenie tohto menu. Vytvorila sa trieda `DependencyAnalysis`, ktorá rozširuje abstraktnú triedu `ContributionItem`. V triede `DependencyAnalysis` bola prekrytá abstraktná metóda `fill(Menu menu, int index)`. V tejto metóde bola vytvorená instancia triedy `MenuItem` s rodičovským menu definovaným v extension point a indexom, ktorý definuje pozíciu medzi ostatnými menu. Následne bol nastevený text menu na Dependency Analysis a nakoniec listener, ktorý reaguje na potvrdenie klávesnicou či kliknutím myšou na menu. Ukážka 2 MANIFEST.MF súboru tohto projektu a obrázok 6 zobrazujúci vyskakovacie menu.

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
 <extension
    point="org.eclipse.ui.menus">
  <menuContribution
     allPopups="false"
     locationURI="popup:org.eclipse.ui.navigator.
     ProjectExplorer#PopupMenu?after=additions">
    <dynamic
       class="com.redhat.dependencyAnalysisThirdB.handlers.
       DependencyAnalysis"
       id="com.redhat.dependencyAnalysisThird.dynamic1">
    </dynamic>
  </menuContribution>
  <menuContribution
     allPopups="false"
     locationURI="popup:org.eclipse.jdt.ui.
     PackageExplorer?after=additions">
    <dynamic
       class="com.redhat.dependencyAnalysisThirdB.handlers.
       DependencyAnalysis"
       id="com.redhat.dependencyAnalysisThird.dynamic2">
    </dynamic>
  </menuContribution>
 </extension>
</plugin>
```

Listing 2: MANIFEST.MF súbor

Pri spustení Dependency Analysis sa vytvorí instancia triedy CreateDA, ktorá implementuje IHandler interface. Spustí sa metóda `Execute`, ktorá vytvorí a zobrazí sprievodcu.



Obrázok 6: Dependency Analysis vo vyskakovacom menu

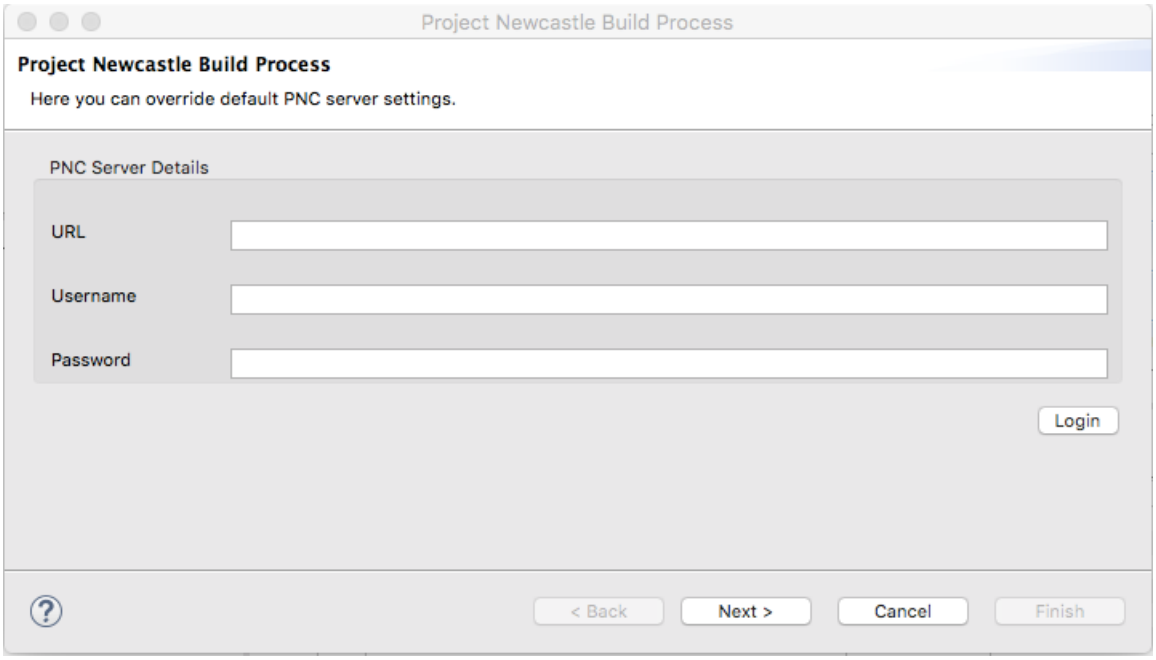
4.5 Vytvorenie UI - sprievodcu

Vytváranie Eclipse sprievodcu je výrazne uľahčené, ak je vytvorené pomocou JFace a Eclipse pluginu WindowBuilder. Po vytvorení JFace Wizard a jeho stránok sa môže ich obsah ručne naprogramovať, alebo si môže programátor skrátiť a uľahčiť prácu použitím pluginu WindowBuilder. WindowBuilder sa vyvolá kliknutím na záložku s názvom Design v hlavnom editore Eclipse.

Pre každú stránku boli v zdrojovom kóde nadefinované ich názvy a krátky popis pomocou metód `setTitle()` a `setDescription()`.

Prvá stránka

Na túto stránku sa vložilo sedem prvkov: trikrát Text, trikrát Label a jeden Button. Labely slúžia na popis textových polí, do ktorých užívateľ zadáva potrebné údaje. Prvým je URL vzdialeného Project Newcastle servera. Druhým je jeho prihlasovacie meno a tretím heslo. Button slúži na odoslanie údajov na server (prihlásenie).



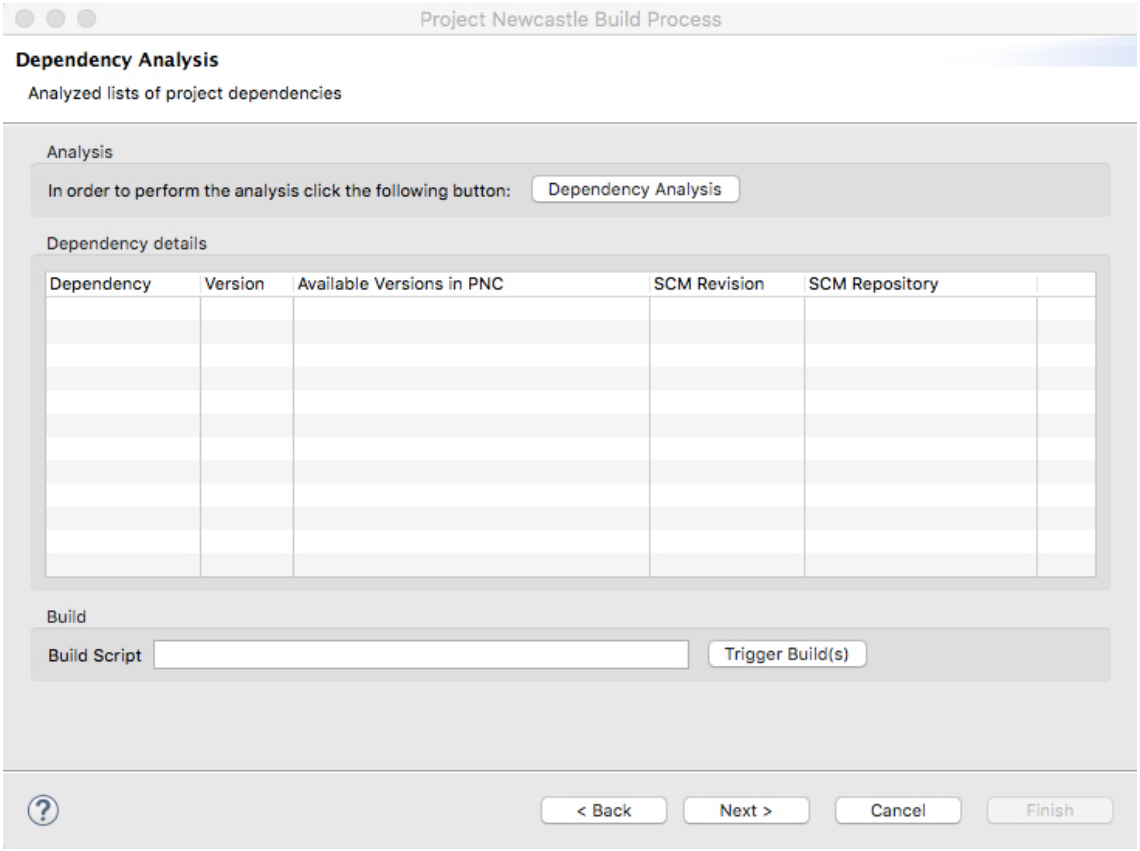
Obrázok 7: Prvá stránka sprievodcu - Login

Všetky prvky sa vložili pomocou WindowBuildera tak, že texty a labely patria do jednej skupiny. Pod túto skupinu bol uložený button Login. Pre čo najpohodlnejšie zobrazovanie sú tieto prvky nastavené tak, aby zachovali svoje rozloženie podľa toho, ako užívateľ okno zväčšuje alebo zmenšuje. Na obrázku 7 je vidieť, ako stránka Login vyzerá.

Druhá a tretia stránka

Druhá a tretia stránka obsahujú každá po jednej JFace `TableViewer` tabuľke. Táto tabuľka je JFace rozšírením pre SWT `Table`. Tabuľky boli tiež nastavené tak, aby menili veľkosť podľa toho, ako užívateľ rozťahuje okno sprievodcu. Prvá stránka navyše obsahuje textové pole pre zadávanie príkazov zostavenia nového build configuration.

Každá tabuľka obsahuje vlastné stĺpce pre zobrazovanie Maven závislostí. Pre tabuľku na druhej stránke sú to: Dependency, Version, Available Versions in PNC, SCM revision a SCM URL. Pre tabuľku na tretej stránke: Dependency, Version, Build Configrutaion a Build Status. Na obrázkoch 8 a 9 je vidieť, ako stránky Analyse a Build vyzerajú.



Obrázok 8: Druhá stránka sprievodcu - Analyse

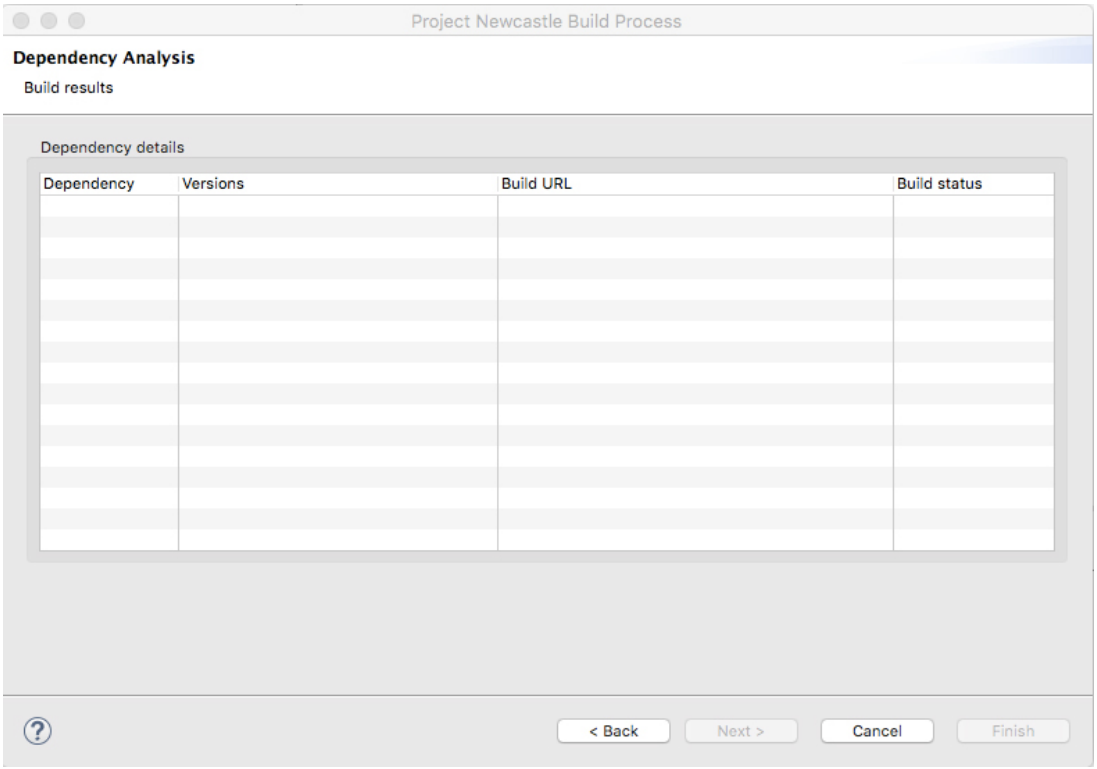
4.6 Prihlasovanie na server

Pre prihlasovanie na Project Newcastle server bolo treba naštudovať ako funguje služba Keycloak a OAuth2 protkol. Inšpiroval som sa kódom v PNC sekcii na `github.com`, kde sa nachádza ukážka, ako sa na PNC server prihlasuje. Pre ilustráciu je priložený kus kódu 8 z triedy OAuthConnect určenej na tento účel.

Pomocou metódy `connect` sa získa OAuth2 token, ktorý sa následne vypíše na Eclipse konzolu a uloží do premennej. Táto metóda pomocou tried `HttpPost`, `CloseableHttpClient` a `CloseableHttpResponse` odošle na Keycloak server údaje zadané na prvej stránke sprievodcu. Celý proces je spustený tlačítkom Login. Po úspešnom prihlásení sa užívateľ môže presunúť na ďalšiu stránku.

4.7 Zobrazenie POM závislostí v tabuľke

Pretože JFace `TableViewer` tabuľka dokáže zobrazovať hodnoty len z jedného `ArrayList`, bolo nutné, aby sa závislosti z pom.xml a závislosti z PNC servera uložili do jednej triedy. Pre tento účel bola vytvorená trieda `MyDep` s atribútmi: `groupId`, `artifactId`, `version`, `pncVersion` a `scmURL`.



Obrázok 9: Tretia stránka sprievodcu - Build

Získanie závislostí z POM súboru

Tento problém ako prvé vyžadoval naprogramovať triedu, ktorá bude zisťovať absolútnu cestu k `pom.xml` súboru patriacemu projektu, na ktorom bol plugin vyvolaný. V triede `PomPath` bola vytvorená statická metóda `public static File getPom()`, ktorá najskôr zistí instanciu aktuálnej workbench, ktorú uloží do premennej. Následne z tejto premennej zistí aktuálny projekt, na ktorom sa pracuje a potom k nej pridá string ”pom.xml”, pretože POM súbor sa nemôže volať inak. Takto sa získa absolútna cesta k potrebnému POM súboru.

Ďalším krokom, bolo prekonvertovanie `pom.xml` typu `File` na typ `MavenProject`. Pre tento účel boli potrebné tri projekty, ktoré boli zadané do závislostí POM súboru. Sú to: `Plexus Common Utilities`, `Maven Core` a `Maven Model`. Pomocou týchto závislostí a tohto kódu 3 sa z POM súboru dajú jednoducho čítať všetky jeho závislosti zavolaním metódy `getDependencies()`.

```
public void pomAsMvnProject(){
    pomFile = PomPath.getPom();

    Model model = null;
    FileReader reader = null;
    MavenXpp3Reader mavenreader = new MavenXpp3Reader();
```

```
    try {
      reader = new FileReader(pomFile);

        model = mavenreader.read(reader);

        model.setPomFile(pomFile);

    }catch(Exception ex){
      ex.printStackTrace();
    }

    project = new MavenProject(model);
  }
```

Listing 3: Prevedenie pom.xml z typu File na typ MavenProject

Získanie závislostí z PNC servera

V triede `PomTransform` bola vytvorená metóda `pomToJArray()` 4, ktorá získané závislosti z prichystaného `MavenProject` POM súboru prichystá do JSON Array v takom formáte, aby sa dala odoslať na analýzu.

```
public JSONArray pomToJArray(){
    List<Dependency> deps = this.project.getDependencies();

    JSONObject jObject = new JSONObject();
    try
    {
        JSONArray jArray = new JSONArray();
        for (Dependency dep : deps)
        {
            JSONObject depJSON = new JSONObject();
            depJSON.put("version", dep.getVersion());
            depJSON.put("artifactId", dep.getArtifactId());
            depJSON.put("groupId", dep.getGroupId());
            jArray.add(depJSON);
        }
        jObject.put("deps: ", jArray);
        System.out.println("String of dependencies to be sent to DA: " +
            jArray.toJSONString());
        return jArray;
    } catch (Exception e) {
        e.printStackTrace();
      return null;
    }
  }
```

Listing 4: Uloženie potrebných atribútov do `JSON Array`

Trieda `DAJsonHttp` potom odosiela na určenú URL JSON Array potrebných závislostí a prijme výsledok, ktorý sa parsuje tak, aby z výsledku uložil do premennej len žiadané údaje a to verzie nájdených závislostí na serveri. Tieto závislosti sú označené ako `Available Dependencies` konrétnej závislosti.

Následne sa tieto údaje vložia do zoznamu premenných triedy `MyDep`, ktorá bola pre tento účel vytvorená. Do premenných `MyDep` sa tiež vytvoria prázdne atribúty pre SCM adresy, ktoré bude neskôr vkladať užívateľ.

Nastavenie tabuľky

Aby tabuľka zobrazovala prichystané dáta musí sa použiť metóda setContentProvider, ktorá má jeden parameter a to `new ArrayContentProvider()` 5. Potom pomocou metódy `setInput` bol vložený zoznam závislostí na zobrazenie.

```
public void createViewer(Composite parent){
    tableViewer = new TableViewer(grpDependencyDetails, SWT.BORDER |
        SWT.FULL_SELECTION | SWT.MULTI | SWT.H_SCROLL | SWT.V_SCROLL |
        SWT.CHECK);
    createColumns(parent, tableViewer);

    table = tableViewer.getTable();

    content.setMyDeps();
    myDeps2 = content.myDeps;

    tableViewer.setContentProvider(new ArrayContentProvider());
      tableViewer.setInput(myDeps2);
```

Listing 5: Ukážka z nastavenia dát pre JFace tabuľku

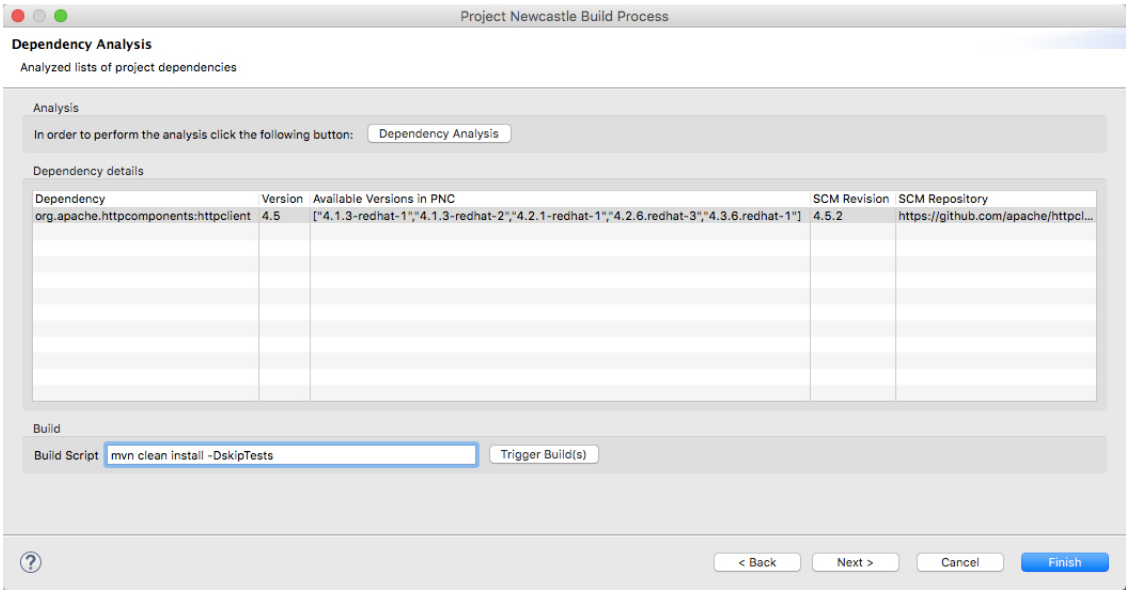
Posledné dva stĺpce boli naprogramované tak, aby sa dal ich obsah meniť užívateľom. Na to bolo potrebné nachystať pre každý stĺpec novú triedu, ktorá rozširuje (extends) triedu `EditingSupport`. Instancia tejto triedy bola potom pridaná konkrétnemu stĺpcu metódou `setEditingSupport()`. Pre ilustráciu je pridaný kód 6 nastavenia možnosti úpravy bunky SCM Revision.

```
col = createTableViewerColumn(titles[3], bounds[3], 3);
col.setLabelProvider(new CellLabelProvider() {
  @Override
  public void update(ViewerCell cell) {
    cell.setText(((MyDep) cell.getElement()).getScmRev());
    }
  });
col.setEditingSupport(new SCMRevEdittingSupport(viewer));
```

Listing 6: Ukážka nastavenia pre úpravu bunky

Podobne bol naprogramovaný každý stĺpec aby korektne zobrazoval jemu určené dáta.

Pre hľadší beh programu bolo vytvorené Dependency Analysis tlačítko pre spúšťanie analýzy závislostí. Vďaka nemu plugin funguje tak, že najskôr užívateľovi zobrazí závislosti jeho POM súboru a po stlačení tlačítka vypíše súvisiace verzie z analýzy. Takto plugin nevolá Dependency Analysis server hneď pri spustení a vyhne sa tak možnému erroru. Obrázok 10 prezentuje zobrazenie analýzu závislostí v tabuľke.



Obrázok 10: Zobrazenie závislostí

Zadávanie hodnôt do sprievodcu

Aby sa dala vytvoriť nová build configuration, užívateľ musí zadať do tabuľky SCM Revision a SCM Repository, ktoré sa uložia do premennej keždej instancie triedy `MyDep`, pre ktorú užívateľ zadal tieto údaje. Ako posledné musí zadať do textového pola build script. Napríklad: `mvn clean deploy -DskipTests=true`, ktorý bude pre každú build configuration rovnaký.

4.8 Vytvorenie a spustenie novej build configuration

Ak užívateľ klikne na tlačítko Trigger build(s) 9, vytvorí sa zoznam s tými konfiguráciami, pre ktoré užívateľ zadal potrebné údaje a vyvolá sa spustenie tejto konfigurácie. Zároveň sa vytvorí nový autentizačný token a odošle sa s konfiguráciami. Je

to tak preto, aby nebol token po expirácii, ak by užívateľ spustil žiadosť na server po dlhšej dobe.

Vytovenie konfigurácie

Pre tento účel bola naprogramovaná trieda `BuildConfiguration`. Každá nová konfigurácia musí obsahovať údaje o projekte, do ktorého bude patriť. Tento projekt však musí byť už vopred vytvorený. Projekt musí mať povinne zadaný názov. Názov sa pre každý projekt nastaví ako artifactId závislosti, pre ktorú je konfigurácia vytvorená. Tento údaj sa uloží do JSON Object a odošle na server. Server odpovie buď, že projekt vytvoril, alebo, že daný objekt s takým názvom už existuje, čo je v poriadku. Znamená to, sa dá použiť pretože bol vytvorený užívateľom už predtým.

Keď sa projekt nastaví, odošle sa for cyklom na server žiadosť o vytvorenie konfigurácie pre každú `BuildConfiguration` uloženú v zozname. Server potvrdí vytvorenie odoslaním Id konfigurácie, ktorú sa uloží do premennej v `BuildConfiguration`. Ukážka konštruktoru `BuildConfiguration` 7.

```
public ContentForNewBC(List<MyDep> myDeps, String buildScript){
    for(MyDep item : myDeps){
      if(item.getScmURL() != "" && item.getScmRev() != ""){
        BuildConfiguration separateBC = new
            BuildConfiguration(item.getArtifactId(), item.getScmURL(),
            item.getScmRev(), buildScript, item);
        BCs.add(separateBC);
      }
    }
  }
```

Listing 7: Ukážka koštruktoru triedy BuildConfiguration

Spustenie konfigurácie

Triedou `TriggerBuild` sa zavolá `HttpPost` na URL obsahujúcu ID danej konfigurácie, ktorá sa má spustiť. Tento proces môže trvať pár sekúnd, ale aj niekoľko hodín.

4.9 Kontrola výsledku

Pre kotrolu výsledku slúži trieda `BuildRecord`, ktorá pomocou `HttpGet` kontroluje výsledok konfigurácie. Ak je odpoveď prázdny JSON String znamená to, že zostavenie ešte prebieha. Preto bol naprogramovaný Thread, ktorý kontroluje výsledok každých 10 sekúnd.

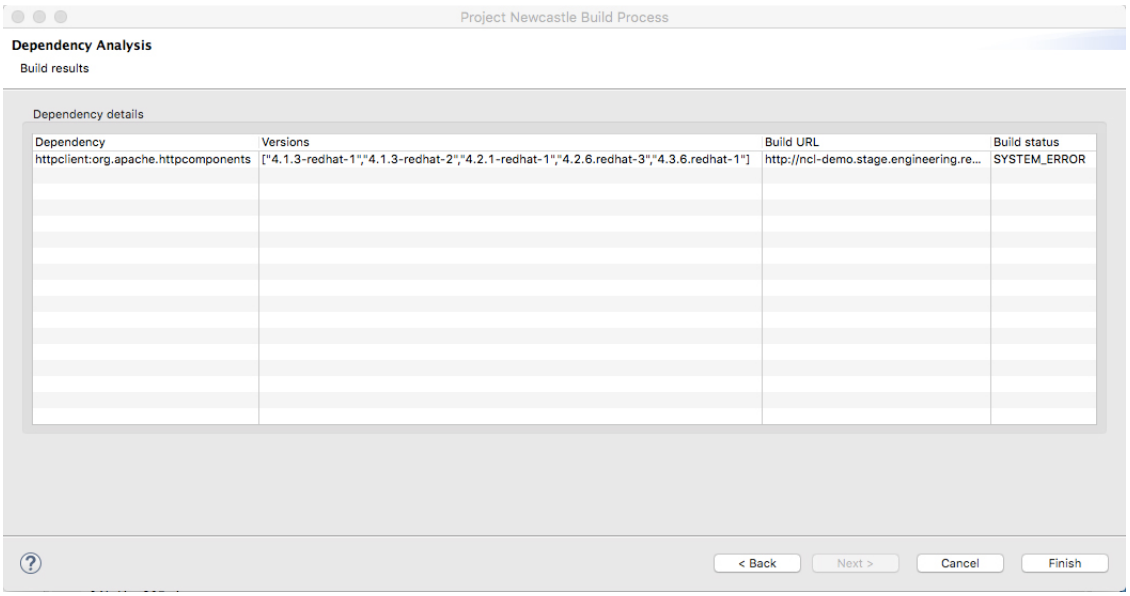
Ak je zostavenie ukončené, trieda `BuildRecord` uloží údaje o statuse výsledku a jeho ID do premennej `BuildConfiguration`.

Zobrazenie výsledku

Nová trieda `BuildResult` obsahuje premenné dependencyName, versions, buildConfiguration a buildStatus. Názov závislosti je nastavený na groupId spojené s artifactId z príslušnej instancie triedy `MyDep`, ktorá bola uložená do každej instancie triedy `BuildConfiguration`. Z triedy `BuildConfiguration` sa ešte prekopíruje premenná status.

Ešte pred finálnym zobrazením výsledku sa však musí znova zavolať analýza závislostí, aby tabuľka zobrazila nové závislosti vytvorené spustením novej konfigurácie zostavenia (v prípade ak bol výsledok úspešný). Pre každý `BuildResult` sa zavolá už spomínaná `DAJsonHttp.http` a výsledok sa uloží do premennej.

Vytvorené instancie triedy `BuildResult` sa uložia do ArrayList a rovnako ako v prvej tabuľke sa nastavia metódou `setContentProvider()` ako zdroj infromácií pre stĺpce tabuľky. Výsledok je vidieť na obrázku 11.



Obrázok 11: Vyhodnotenie zostavenia

5 Záver

Na úvod práce boli rozobraté a vysvetlené súvisiace technológie a pojmy. V ďalšej kapitole bol predstavený návrh riešenia na základe zadania a analýzy požiadavkov. V poslednej kapitole bola opísaná implementácia Eclipse pluginu. Plugin bol naprogramovaný ako sprievodca a dokáže spúšťať zostavenia na vzdialenom serveri podľa požiadavkov užívateľa. Všetky kódy obsahujú javadoc dokumentáciu. Vďaka tomu je kód zrozumiteľnejší a ľahšie sa bude v budúcnosti upravovať. Plugin bol nahraný na GitHub repozitár a je voľne dostupný na stiahnutie a úpravy.

Do budúcna by sa však dalo upraviť niekoľko nedokonalostí. Bolo by dobré, aby vedel plugin čítať závislosti z viacerých POM súborov, pretože väčšie projekty spravidla obsaujú väčší počet POM súborov, ktoré si navzájom zdieľajú informácie a čerpajú ich od seba. Tento plugin s ostatnými POM súbormi nepočíta. Pretože zostavovanie môže prebiehať niekoľko hodín, ďalšia verzia pluginu by mala mať možnosť behu na pozadí, aby mal užívateľ možnosť práce v Eclipse na inej práci. Napokon by bol užívateľ upozornený o konci kontroly notifikáciou.

6 Literatúra

Anicas, M. *An Introduction to OAuth 2*. In: Digital Ocean [online]. New York: Creative Commons Attribution-NonCommercial-ShareAlike 4.0, 2014 [cit. 2016-05-17]. Dostupné z: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2.

The Apache Software Foundation. *What is Maven?*. In: Apache Maven Project [online]. The Apache Software Foundation, 2016 [cit. 2016-05-17]. Dostupné z: https://maven.apache.org/what-is-maven.html.

Danciu, D. *Continuous Delivery*. In: Today software magazine [online]. Cluj: Today Software Magazine, 2014 [cit. 2016-05-18]. Dostupné z: http://www.todaysoftmag.com/article/1068/continuous-delivery.

Deitel, P.J., Deitel, H.M. *Java: how to program. 9th ed.*. Upper Saddle River, N.J.: Prentice Hall, c2012. ISBN 978-013-2575-669.

Eclipse [online]. Ottawa: The Eclipse Foundation, 2016 [cit. 2016-05-17]. Dostupné z: https://eclipse.org.

Fowler, M. *Continuous Integration*. In: Martin Fowler [online]. Chicago, 2006 [cit. 2016-05-17]. Dostupné z: http://martinfowler.com/articles/continuousIntegration.html.

Gosling, J., Joy, B., Steele, G.L., Bracha, G., Buckley, A. *The Java® language specification*. Java SE 8 edition. ISBN 978-013-3900-699.l.

J2EEBrain *Applications on Eclipse*. IIn: J2EEBrain [online]. [cit. 2016-05-18]. Dostupné z: http://www.j2eebrain.com/java-J2ee-applications-on-eclipse.html.

Keycloak Reference Guide - Overview. In: JBoss Developer [online]. Apache ASL 2.0, 2016 [cit. 2016-05-17]. Dostupné z: https://keycloak.github.io/docs/userguide/keycloak-server/html/index.html.

Liang, Y. D. *Introduction to Java programming: comprehensive version*. Tenth edition. ISBN 01-337-6131-2..

Mcaffer, J., Lemieux, J-M., Aniszczyk, Ch. *Eclipse Rich Client Platform* 2nd ed. Upper Saddle River, NJ: Addison-Wesley, c2010. Eclipse series. ISBN 978-032-1603-784..

Munif, A. *What is Maven? Why should I use it?*. In: Abdulmuneverlose [online]. Indonézia: abdulmuneverlose, 2013 [cit. 2016-05-18]. Dostupné z: https://abdulmuneverlose.wordpress.com/2013/12/25/what-is-maven-why-should-i-use-it.

Scarpino, M. *SWT/JFace in action*. Greenwich, CT: Manning, 2005. ISBN 19-323-9427-3..

VMware Inc. *OSGi Concepts*. IEclipse [online]. Virgo, 2011 [cit. 2016-05-18]. Dostupné z: https://www.eclipse.org/virgo/documentation/virgo-documentation-3.6.0.M01/docs/virgo-user-guide/html/ch02s02.html.

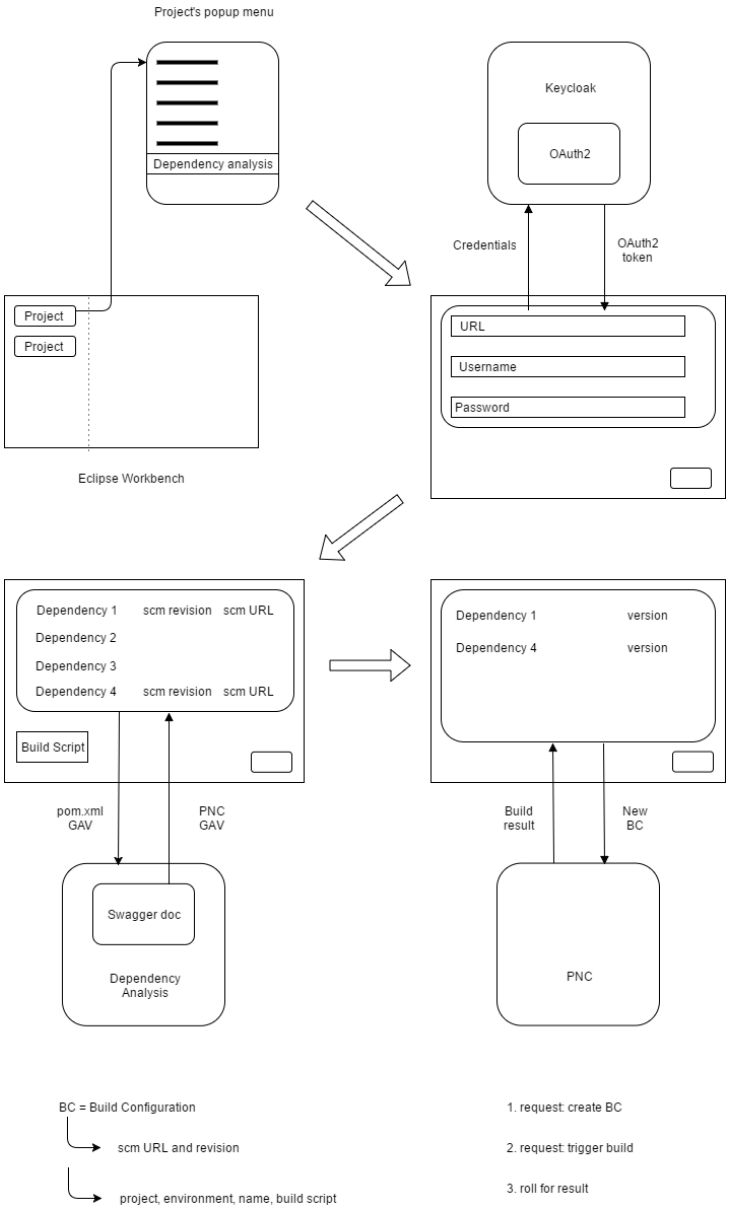
Vogel, L.*Eclipse Tycho for building Eclipse Plug-ins, OSGi bundles and RCP applications - Tutorial*. In: Vogella [online]. Hamburg: vogella GmbH, 2016 [cit. 2016-05-18]. Dostupné z: http://www.vogella.com/tutorials/EclipseTycho/article.html.

Vogel, L.*Maven for building Java applications - Tutorial*. In: Vogella [online]. Hamburg: vogella GmbH, 2015 [cit. 2016-05-17]. Dostupné z: http://www.vogella.com/tutorials/ApacheMaven/article.html.

Vogel, L.*SWT - Tutorial*. In: Vogella [online]. Hamburg: vogella GmbH, 2016 [cit. 2016-05-17]. Dostupné z: http://www.vogella.com/tutorials/SWT/article.html#swt-overview.

Prílohy

A Návrh riešenia



Obrázok 12: Návrh implementácie pomocou diagramu

B Metóda connect triedy OAuthConnect

```
private static String[] connect(String url, Map<String, String> urlParams)
        throws ClientProtocolException, IOException{

    CloseableHttpClient httpclient = HttpClients.createDefault();
    HttpPost httpPost = new HttpPost(url);

    httpPost.setHeader("Content-Type", "application/x-www-form-urlencoded");

    List <BasicNameValuePair> urlParameters = new ArrayList <BasicNameValuePair>();
    for(String key : urlParams.keySet()) {
        urlParameters.add(new BasicNameValuePair(key, urlParams.get(key)));
    }
    httpPost.setEntity(new UrlEncodedFormEntity(urlParameters));
    CloseableHttpResponse response = httpclient.execute(httpPost);

    String refreshToken = "";
    String accessToken = "";
    try {
        BufferedReader rd = new BufferedReader(
                new InputStreamReader(response.getEntity()
                    .getContent()));

        String line = "";
        while ((line = rd.readLine()) != null) {
            if(line.contains("refresh_token")) {
                String[] respContent = line.split(",");
                for (int i = 0; i < respContent.length; i++) {
                    String split = respContent[i];
                    if(split.contains("refresh_token")) {
                        refreshToken = split.split(":")[1]
                          .substring(1,split.split(":")[1]
                          .length() -1);
                    }
                    if(split.contains("access_token")) {
                        accessToken = split.split(":")[1]
                          .substring(1,split.split(":")[1]
                          .length() -1);
                    }
                }
            }
        }
    } finally {
        response.close();
    }
    return new String[] {accessToken,refreshToken};
}
```

Listing 8: Prihlasovanie cez triedu OAuthConnect

C Tlačítko Trigger Build(s)

```
Button btnCreateBc = new Button(grpBuild, SWT.NONE);
btnCreateBc.setBounds(433, 3, 113, 28);
btnCreateBc.addSelectionListener(new SelectionAdapter() {
  @Override
  public void widgetSelected(SelectionEvent e) {
    System.out.println("Create new BC button selected!");

    ContentForNewBC newBC = new ContentForNewBC(myDeps2, text_3.getText());

    Login loginPage = (Login)getWizard().getPage("Page one");

    newBC.printInfo();
    System.out.println(loginPage.getLogin());

    try {
      newToken = OAuthConnect.getrefreshToken(
      "https://ncl-keycloak.stage.engineering.redhat.com/auth/realms/pncredhat/protocol/openid-connect/token", "pncdirect",
      loginPage.getTextUSR(), loginPage.getTextPASS());
    } catch (IOException e1) {
      // TODO Auto-generated catch block
      e1.printStackTrace();
    }

    for(BuildConfiguration item : newBC.getBCs()){
      BCProject.http(BCProject.json("name", item.getProject()).toJSONString(), newToken);
      System.out.println("project created!");
      item.setProjectId(BCProject.getId(item.getProject(), newToken));

      NewBcRequest bcRequest = new NewBcRequest();

      bcRequest.http(NewBcRequest.jsonObj(item).toJSONString(), newToken);
      item.setBcId(bcRequest.getBcId());
    }

    newBC.printInfo();

    System.out.println("All Configurations done!");
    System.out.println("Lets try to trigger them");
    for(BuildConfiguration item : newBC.getBCs()){
      TriggerBuild trigger = new TriggerBuild();

      trigger.http(item.getBcId(), TriggerBuild.triggerBuildjson(item.getBcId()).toJSONString(), newToken);
      item.setBuildTriggerId(trigger.getbuildTriggerId());
    }

    for(BuildConfiguration item : newBC.getBCs()){
      boolean done = false;
      BuildRecord br = new BuildRecord();
      while(done == false){
        try {
          System.out.println("Waiting 10 seconds to check for result");
          Thread.sleep(10000);
          done = br.http(item.getBcId(), newToken);

        } catch (InterruptedException e1) {
          System.out.println("problem with sleep");
          // TODO Auto-generated catch block
          e1.printStackTrace();
        }

      }
      item.setLatestBuildRecordId(br.getLatestBuildRecordId());
      item.setStatus(br.getStatus());
    }

    newBC.printInfo();
    toNextPage = newBC;

  }
}
```

Listing 9: Ukážka funkcionality Trigger Build(s)