ZAVOD ZA ELEKTRONIKU, MIKROELEKTRONIKU, RAČUNALNE I INTELIGENTNE SUSTAVE
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
SVEUČILIŠTE U ZAGREBU

IZRADA DODATAKA ZA ECLIPSE IDE

Hrvoje Slaviček

SEMINARSKI RAD

Zagreb, 2007.

Sadržaj

1. Uvod

Eclipse je programsko okruženje (engl. *software framework*) otvorenog koda, neovisno je o operacijskom sustavu, te je bazirana na programskom jeziku Java. Namjena Eclipse-a je pomoć pri izradi složenih aplikacija kako ih projekt naziva "rich-client aplikacijama" kao suprotnost web aplikacijama. Eclipse pruža proširivu razvojnu platformu dizajniranu za razvoj integriranih razvojnih okruženja IDE(engl. *Integrated development environment*), raznih alata, aplikacijskih okruženja (engl. *Aplication framewroks*), te bilo kakvih RCP aplikacija. Eclipse Software Development Kit (SDK) je kombinacija nekoliko Eclipse projekata, uključujući Platform, Java Development Tools (JDT), i Plug-in Development Environment (PDE). Eclipse SDK je integrirano razvojno okruženje za razvoj Java aplikacija i izgradnju produkata baziranih na Eclipse Platformi. Ova platforma, nezavisna od proizvođača, sastoji se od jezgre(engl. *core*) i raznih dodataka (engl. *Plugin*) . Jezgra pruža usluge za upravljanje dodacima, a dodaci pružaju stvarnu funkcionalnost.

Eclipse predstavlja pouzdanu i skalabilnu tehnologiju, nad kojom svatko može pridonijeti svoj dio obogaćujući postojeće komponente ili dodajući novu funkcionalnost.

Eclipse je originalno bio razvijen od strane IBM kao nasljednik VisualAge porodice alata, a danas Eclipse-om upravlja Eclipse Foundation, nezavisni i neprofitabilni konzorcij koji se sastoji od vodećih softverskih kompanija. Veliki broj značajnih softverskih kompanija prihvatio Eclipse kao budući okvir (engl. *framework*) za njihova integrirana razvojna okruženja,te kompanije poput IBM, Borland Software, Ericsson, Hewlett-Packard, Intel, Oracle, Red Hat, SAP razvijaju komercijalne dodatke za Eclipse platformu.

Eclipse platforma dolazi s dodacima koji olakšavaju Java razvoj, skup tih dodataka nalazi se u paketu pod imenom Java Development Tools (JDT). JDT doprinosi s Java specifičnim ponašanjem u ovoj generičkoj platformi tako što dodaje Java specifične alate poput editor, prevoditelja i debugera i grafičko korisničko sučelje. JDT podržava proširivanje tako da ga korisnik u svakom trenutku može proširiti. JDT je samo jedan primjer od raznih proširivih elemenata Eclipse platforme.

Plug-in Development Environment (PDE) tj. okruženje za razvoj dodataka, dolazi kao dio Eclipse SDK paketa, je skup alata dizajniranih da pomažu Eclipse programeru u razvoju, testiranju, ispitivanju, izgradnji izvršnih arhiva i isporuki Eclipse dodataka, fragmenata, mogućnosti(engl. *features*) i RCP aplikacija.

Iako je pisanje Eclipse dodataka je dobro definiran proces, on može bit dosta složen. Proces pisanja dodataka svodi se na stvaranje i uređivanje manifest datoteke, pisanje Java izvornog koda, prevođenje izvornog Java koda u Java izvršni kod, testiranje dodatka i pakiranje dodatka u oblik pogodan za isporuku. PDE ovaj zadatak pojednostavljuje, tako da se integrira u radno mjesto (engl. *Workbench*) dodajući razne alate Eclipse platformi.

Eclipse je proširiva platforma, kao takva pruža osnovni skup servisa koji kontroliraju dodatke da bi oni međusobno mogli funkcionirati kao jedna cjelina. Osnovni mehanizam koji Eclipse pruža je davanje mogućnosti da svaki dodatak bude proširiv. Na taj način moguće je graditi integrirano razvojno okruženje s dijelovima od nekog već postojećeg razvojnog okruženja, što znatno skraćuje trajanje razvoja.

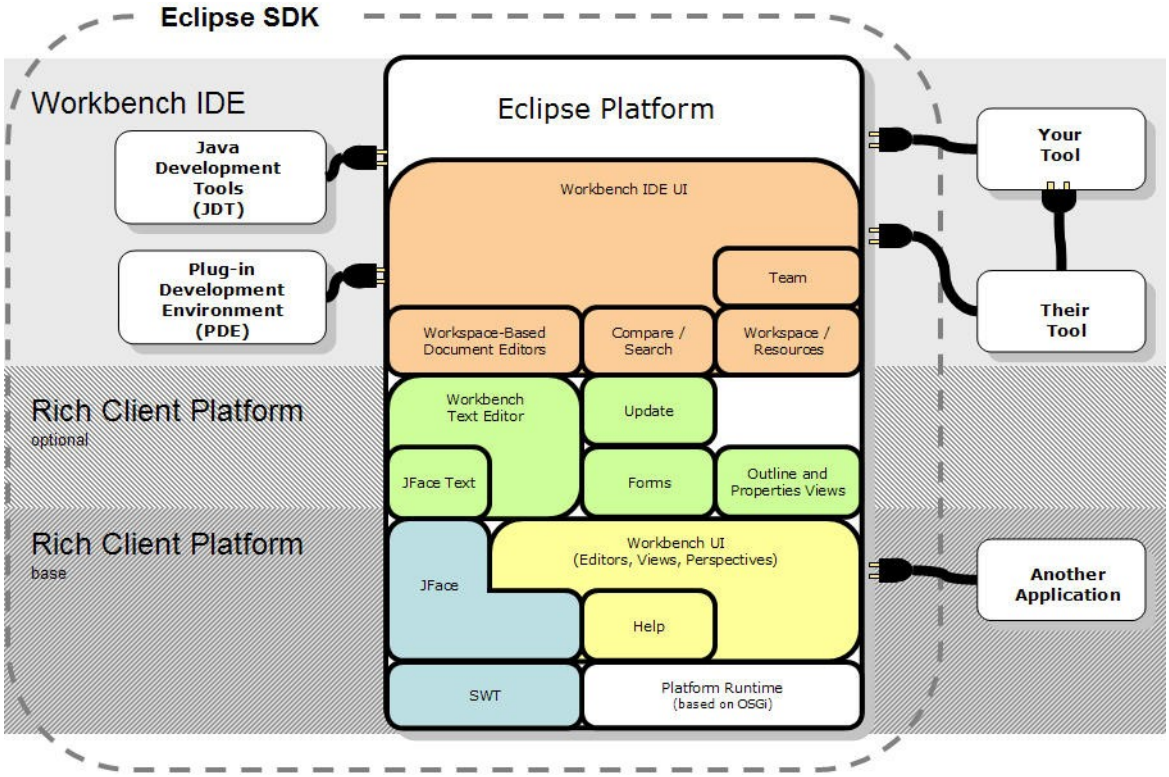
Iako je Eclipse platforma specijalizirana za izgradnju integriranih razvojnih okruženja, Eclipse platforma može se iskoristiti za izgradnju "rich-client aplikacijama" - RCP koje kao svoj osnovnu jezgru koriste Eclipse runtime i grade se kao skup Eclipse dodataka. RCP aplikacije isporučuju se kao Eclipse runtime i samo oni dodaci koji su potrebni da bi aplikacija funkcionirala.

2. Arhitektura Eclipse platforme

2.1. Elementi Eclipse arhitekture

Eclipse SDK (Software Developer Kit) sastoji se od komponenti koje su proizašla iz 3 Eclipse potprojekta: (Platform, JDT - Java Development Tools, i PDE - Plug-in Development Environment).

Slika 2.1. prikazuje na je koji način izgrađen Eclipse SDK i kako se može proširivati. Eclipse platform možemo razgraničiti u dva osnovna dijela: Rich Client Platform i Workbench (radno mjesto).



Slika 2.1: Arhitektura Eclipse SDK-a

Osnovni dio RCP -a je Platform Runtime koji je zadužen za pokretanje i održavanje svih dodataka za vrijeme ciklusa njihovog izvođenja. Za izgradnju grafičkog korisničko sučelja RCP pruža SWT (Standard Widget Toolkit) grafičke elemente. SWT elementi implementirani su tako da koriste izgled grafičkih elemenata operacijskog sustava na kojem se koriste. JFace su složeniji elementi korisničkog sučelja, te su oni izgrađeni pomoću SWT-a. Pomoću JFace i SWT gradi se cijelo grafičko korisničko sučelje Eclipse-a, te se ono dijeli u veće komponente koje se ponovno mogu iskoristiti u nekom drugom dodatku. Na RCP-u je izgrađeno Eclipse radno mjesto (engl. *Workbench*). Eclipse radno mjesto osigurava ujednačeni izgled sučelja, i svim dodacima omogućuje univerzalni razvojni proces tako što im pruža alate za traženje, timski rad, jedinstven sustav pomoći, radni prostor(engl. *Workspace*) u koji se smještaju projekti, i datoteke, sustav za ažuriranje dodataka. Eclipse radno mjesto proširuju JDT pružajući okolinu za Java razvoj i PDE pružajući okolinu za razvoj dodataka.

Dodatak može proširiti Eclipse platformu na bilo kojem mjestu te na taj način omogućuje nadogradnju od najniže do najviše razine apstrakcije. Svaki dodatak opet može biti proširen, te je na taj način dobivena je beskonačno proširiva platforma.

2.1.1. Runtime jezgra

Runtime jezgra (engl. *core*) platforme implementira runtime engine koji pokreče osnovni dio platforme i dinamički otkriva i pokreće dodatke. Dodatak je strukturirana komponenta koja opisuje sebe samom sustavu koristeći OSGi manifest (`MANIFEST.MF`) datoteku i manifest dodatka (`plugin.xml`). Platforma održava registar instaliranih dodataka i funkcionalnosti koju oni pružaju.

2.1.2. Upravljanje sredstvima

Sredstva (engl. *Resource*) u Eclipse-u su projekti, direktoriji, i datoteke Eclipse radnog prostora(engl. *Workspace*). Dodatak `org.eclipse.core.resurces` pruža potrebnu funkcionalnost za pristupanje i upravljanjem sredstvima bez direktnog pristupa datotečnom sustavu. Na taj način je omogućeno omogućiti mogućnosti koje obično datotečni sustav ne pruža. Sustav upravljanja sredstvima omogućuje stvaranje i modificiranje projekata, direktorija i datoteka te njihovu organizaciju u internu Eclipse strukturu.

2.1.3. Korisničko sučelje radnog mjesta

Korisničko sučelje radnog mjesta (engl. *Workbench User interface*) definira veći broj točaka proširenja (engl. *Extension Points*) koji omogućuju drugim dodacima da dodaju izbornike, alatne trake, drag&drop operacije, dijaloške okvire, čarobnjake i prilagođene poglede(engl. *Views*) i uređivače (engl *Editor*).

Workbench dodatka omogućava cjelokupnu strukturu i predstavlja nadogradivo korisničko sučelje. Radno mjesto sagrađeno je pomoću SWT – Standard Widget Toolkit koji se osigurava izgled grafičkih elemenata koji odgovaraju izvornom izgledu grafičkog sučelja koje pruža operacijski sustav i pomoću JFace – okvira(engl. *Framework*) za izgradnju složenijih elemenata korisničkog sučelja kao što su dijaloški okvir(engl. *Dialog*), čarobnjaka(engl. *Wizard*), radnji(engl. *Actions*) korisničkih postavki(engl. *User preferences*), implementirananog upotrebom SWT koji pojednostavnjuje uobičajene programske zadatke izgradnje korisničkog sučelja.

JFace se sastoji od uobičajenih komponenti koje zahtjeva složeno korisničko sučelje,dialoški okviri, postavke(engl. *Preferences*), čarobnjaci (engl. W*izards*) i izvještaja o napredovanju za dugotrajne operacije (engl. *Progres bar*). Jedan od korisnijih obilježja JFace su radnje (engl. *Actions*). Sustav radnji dopušta raznim radnjama da budu definirane nezavisno od njihovih točnih pozicija u korisničkom sučelju. Radnja predstavlja akciju koja može biti pokrenuta od strane korisnika putem tipke, elementa u izborniku ili elementa alatnoj traci. Svaka radnja zna svoju oznaku korisničkog sučelja (labela, ikona, tool tip, itd.) koji se koriste za konstrukciju pogodnih grafičkih elemenata kako bi se radnja mogla prezentirati. Ovo razdvajanje dopušta jednoj radnji da bude korištena na više mjesta u UI-ju, što znaci da je lako mijenjati poziciju gdje je radnja prisutna u korisničkom sučelju bez potrebe mijenjanja koda za samu radnju.

Dodatak Workbech izgrađen je pomoću SWT i JFace elemenata grafičkog sučelja, ali i komponenata višeg sloja kao što su pogledi , uređivač teksta (engl. *Text editors*), forme (engl. *Forms*). Njihova funkcionalnost podijeljena je u više dodataka :

```
org.eclipse.ui
org.eclipse.swt
org.eclipse.text
org.eclipse.jface
org.eclipse.jface.text
org.eclipse.ui.views
org.eclipse.workbench
org.eclipse.workbench.texteditor
org.eclipse.ui.ide
org.eclipse.ui.editors
org.eclipse.ui.forms
```

Eclipse radno mjesto (engl. *Workbench*) implementirano je pomoću tih dodataka. Korištenjem gore navedenih dodataka u vlastitoj aplikaciji dobiva se kvalitetno korisničko sučelja s relativno malo truda. Više o elementima Eclipse korisničkog sučelja i njihovoj implementaciji opisano je u poglavlju 5.

2.1.4. Podrška za timski rad

Skup dodataka za timski rad omogućuje drugim dodacima da definiraju vlastitu implementaciju za timski rad, pristup repozitorijima i kontrolu verzija. Eclipse SDK dolazi s jednim takvim dodatkom za CVS.

2.1.5. Sustav pomoći

Sustava za pomoć Eclipse Platforme dopušta svakom dodatku da svoju dokumentaciju doda u Eclipse sustav pomoći, kao jednu ili više tematski cjelina. Uobičajeno je da dodataka svoju dokumentaciju definira kao dvije tematske cjeline, jednu kao vodič za korisnike a drugu kao API dokumentaciju za njegovo proširivanje, ako je moguće njegovo proširivanje.

Sam sadržaj pomoći piše se u HTML obliku, dok se navigacijske strukture definiraju u XML datotekama. Prednost ovakvog pristupa je da već postojeća, ili automatski generirana(npr. JavaDoc) dokumentacija bude integrirana u sustav pomoći bez potrebe za prethodnom modifikacijom.

XML navigacijska datoteka i HTML sadržajna datoteka su pohranjeni u nekom od direktorija dodatka. Mali dodaci obično se obično isporučuju zajedno s dokumentacijom, dok veliki dodaci imaju posebni dodatak u kojem se nalazi samo dokumentacija za njih. Eclipse sustav pomoći upotrebljava vlastiti ugrađenog web server za pribavljanje aktualnih stranica pomoći iz dodataka. Ovakav pristup dopušta Eclipse-u razrješavanje specijalnih poveznica između dodataka i otpakiravanje HTML dokumenata iz ZIP arhiva. Isto tako sustav pomoći moguće je pregledavati pomoću vanjskog web preglednika .

2.1.6. Java development tools (JDT)

Java Development Tools (JDT) je skup dodataka koji proširuju Eclipse radno mjesto pružajući mogućnosti za uređivanje, pregledavanje, prevođenje i pokretanje Java koda. JDT je najbolji i najpotpuniji primjer proširivanja Eclipse platforme, te njegova integraciji u Eclipse platformu omogućuje jednostavan razvoj Eclipse dodataka.

2.1.7. Plug-in Development Environment (PDE)

Okruženje za razvoj dodataka (PDE) je skup alata koji automatiziraju postupak izrade dodataka. PDE i njegovo korištenje u izradi dodataka detaljno je opisano u 6. poglavlju.

2.1.8. Radni prostor (Workspace)

Velik broj dodataka za Eclipse Platform manipuliraju datotekama. Radi lakšeg korištenja one se organiziraju u radnom prostoru(engl. *Workspace*). U radnom prostoru podrazumijevaju se tri vrste resursa:

- Projekti – Projekt je korijenski čvor resursnog stabla, on sadrži direktorije i datoteke
- Direktoriji – Direktoriji sadrži datoteke i druge direktorije
- Datoteke – Datoteke su završni čvorovi resursnog stabla, ne mogu sadržavat druge resurse

Eclipse koristi datotečni sustav da bi spremio direktorije i datoteke projekata. Svaki projekt sadržava datoteke koje je kreirao i njima manipulirao korisnik. Sve datoteke u radnom prostoru dostupne su u datotečnom sustavu. Eclipse pruža API za upravljanje resursima radnog prostora.

2.2. Dodaci

Jedina zadaća male Eclipse jezgre je učitavanje i izvršavanje dodataka, sva ostala funkcionalnost Eclipse Platforme sadržana je u dodacima. U većini slučajeva, dodatak sastoji se od Java arhive, ali može sadržavati i druge datoteke. Svaki dodatak obavezno mora sadržavati manifest dodatka datoteku `plugin.xml` koja opisuje konfiguraciju dodataka. `Plugin.xml` objašnjeni je u 3. poglavlju.

Dodatak je najmanji dio Eclipse Platforme koji se može razviti i isporučiti zasebno. Obično mali alati su napisani kao jedan dodatak, dok složeniji alati svoju funkcionalnost dijele na nekoliko dodataka. Izuzev male jezgre (engl. *Kernel*) poznate kao *Platform Runtime* sva funkcionalnost Eclipse Platforme smještena je u dodacima.

Dodaci se pišu u programskom jeziku Java. Tipični dodatak sastoji se od Java koda i JAR biblioteci, skupa datoteka koji se samo čitaju i ostalih resursa kao što su slike web predlošci, skupine poruka, programskih biblioteka itd. Neki dodaci ne moraju uopće sadržavati Java kod, primjer takvoga dodatka je sustav pomoći koji se sastoji od HTML stranica.

Svaki dodatak mora imati manifest datoteku u kojem se definiraju poveznice s ostalim dodacima. Model povezivanja dodatak je jednostavan: svaki dodatak definira određeni broj imenovanih **točaka proširenja**, te određeni broj **proširenja**(engl. *Extension)* prema jednoj ili više točka proširenja drugih dodataka.

Točka proširenja dodatka može biti proširena od nekog drugog dodatka. Na primjer Workbench dodatak definira točku proširenja za korisničke postavke (engl. *user preferences*), svaki dodatak može može dodati svoje korisničke postavke tako da proširi točku proširenja korisničke postavke dodatka Workbench.

Točka proširenja može imati i pripadajuće API sučelje, preko kojega drugi dodaci proširuju tu točku proširenja. Svaki dodatak ima slobodu definirati svoje točke proširenja i svoj novi API koji drugi dodaci mogu koristiti.

Pri pokretanju Platform Runtime otkriva skup dostupnih dodataka, čita njihove manifest datoteke i u memoriji izgrađuje registar (engl. *registry*) dodataka. Platfrom Runtime uparuje deklaraciju proširenja po imenu i pripadajuću točku proširenja. Svi problemi pri pokretanju, kao proširenje na nepostojećoj točki proširenja, se detektiraju i bilježe u log datoteku. Dobivenom registru dodataka moguće je pristupiti preko Platform API-a. Dodaci mogu biti dodani, zamijenjeni ili obrisani nakon startup-a.

Manifest dodatka sastoji se od nekoliko datoteka. Datoteka `manifest.mf` je manifest OSGi paketa te opisuje o čemu sve ovisi dodatak (runtime dependencies). Druga datoteka je `plugin.xml`, ona je XML datoteka koja sadrži opise proširenja koja dodatak pruža te, točaka proširenja koje mogu biti proširene.

Točka proširenja može definirati dodatne specijalizirane XML elemente za korištenje s proširenjem. Ovaj način dozvoljava dodatku koji daje proširenje da komunicira proizvoljnim informacijama s dodatkom koji definira točku proširenja. Štoviše informacije o proširenju i točki proširenja tada su dostupna iz registra dodataka bez potrebe aktiviranja tog dodatka ili učitavanja pripadajućeg koda. Ovo svojstvo omogućuje veliku bazu dodataka koji mogu biti istovremeno instalirani, ali ne moraju istovremeno biti učitani ako se ne koriste što rezultira bržim startup-om i manjom potrošnjom memorije.

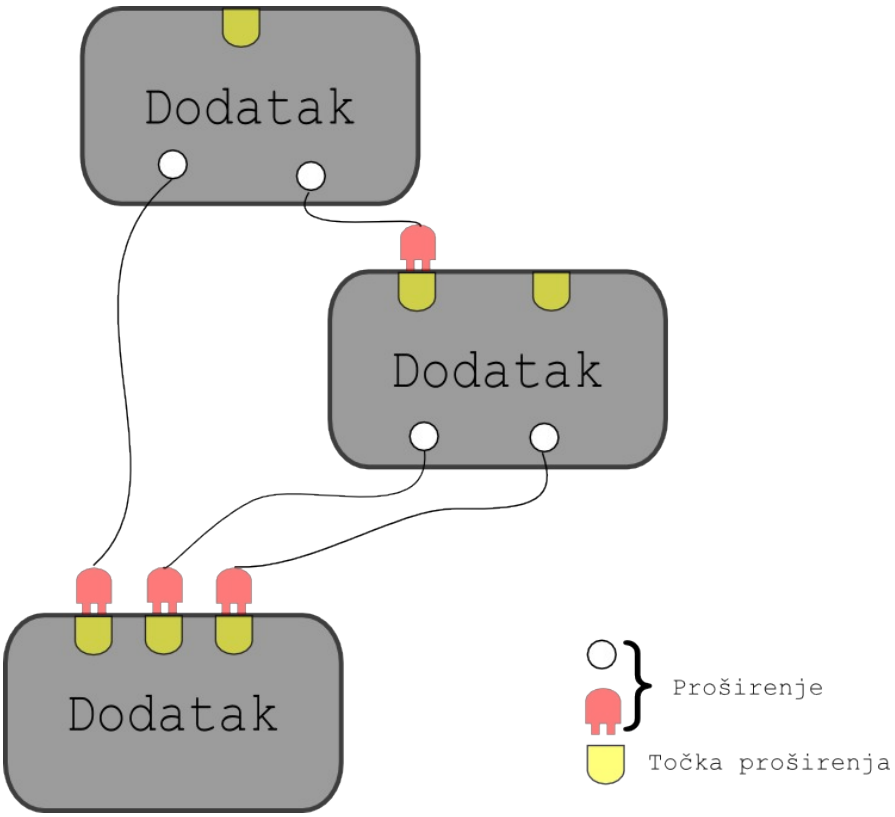
Korištenje manifesta baziranom na XML- u omogućuje jednostavniju izradu alata za izradu dodataka, primjer takvog alata je PDE (Plug-In Development Environment) koji je uključen u Eclipse SDK.

Dodatak se aktivira onda kad izvršni kod stvarno treba biti pokrenut. Jednom pokrenut dodatak koristi registar dodataka da otkrije i pristupi onim proširenjima koji koriste njegove točke proširenja. Jednom aktiviran dodatak ostaje aktivan do njegove eksplicitne dekativacije ili gašenja Eclipse Platforme .

2.3. Točke proširenja

Točke proširenja su glavni koncept arhitekture dodataka. Model povezivanja dodatak je jednostavan: svaki dodatak definira određeni broj imenovanih **točaka proširenja**, te određeni broj **proširenja** prema jednoj ili više točka proširenja drugih dodataka.

Točka proširenja dodatka može biti proširena od nekog drugog dodatka. Na primjer Workbench dodatak definira točku proširenja za korisničke postavke (engl. *user preferences*), svaki dodatak može može dodati svoje korisničke postavke tako da proširi točku proširenja korisničke postavke dodatka Workbench. Primjer proširivanja točaka proširenja dan je slikom 2.2. gdje je prikazan model povezivanja točaka proširenja i proširenja tri dodatak koji se međusobno proširuju da bi dobili željenu funkcionalnost iskorištavajući usluge nekog drugog dodataka.



Slika 2.2: Mehanizam točaka proširenja i proširenja

Točka proširenja može imati i pripadajuće API sučelje, preko kojega drugi dodaci proširuju tu točku proširenja. Svaki dodatak ima slobodu definirati svoje točke proširenja i svoj novi API koji drugi dodaci mogu koristiti.

Točke proširenja i proširenja definiraju se u manifest datoteci `plugin.xml` te svaki dodatak opisuje na koje će se postojeće točke dodatak spojiti i koje nove točke proširenja želi pružiti Eclipse platformi.

2.4. Kasno učitavanje

Kako je cijeli Eclipse IDE izgrađen od malog runtime engine-a i velikog broja dodataka Eclipse platforma je dizajnirana tako da se smanji potrošnja memorije. Omogućavanje instalacije neograničenog broja dodatak i njihovo učitavanje povećalo je potrošnju memorije, zbog toga je u uveden mehanizam kasnog učitavanja (engl. *Lazy loading*). Mehanizmom kasnog učitavanja omogućuje učitavanje Java klasa tek kad su one potrebne. Eclipse platforma gradi registar dodataka iz datoteke `plugin.xml` svakog dodatka. Pomoću Eclipse class loader-a učitavaju se samo potrebni dodaci(npr. Workbench). Korisničko sučelje se učitava tako da Workbench dodatak iz registra dodataka vidi koji dodaci su proširili njegove točke proširenja te na temelju registra dodaje elemente poput izbornika,gumbi, alatnih traka i slično. Stvarni kod koji se izvod kod aktiviranja radnji učitava se prilikom prve aktivacije. Na taj način moguće je imati neograničen broj dodataka u jednoj instalaciji Eclipse-a, i koristi samo one koji su u jednom trenutku potrebni.

3. Manifest dodatka

Manifest dodatka raspodijeljeni je u dvije datoteke `Manifest.mf` i `plugin.xml`. Ovdje je bitno napomenuti da se manifest promijenio u odnosu na Eclipse verziju 3.0 te da se ovaj opis odnosi na trenutno aktualnu verziju veće od 3.10. Stare verzije dodataka mogu se i dalje koristi jer Eclipse je zadržao kompatibilnost s starim dodacima, ali se preporuča preći na novi model. Da bi se verzije manifesta razlikovali u datoteku `plugin.xml` na početku se stavlja tag <?eclipse version?> npr:`<?eclipse version="3.2"?>`

Manifest dodatka može se ručno uređivati pomoću tekst editora pošto je je `manifest.mf` tekstualna datoteka, a `plugin.xml` XML datoteka te su čitljive čovjeku. Korištenjem XML baziranog manifesta omogućeno je pisanje alata koji podržavaju kreiranje dodataka, jedan od takvih alata je PDE koji je uključen u Eclipse SDK. PDE omogućava jednostavniji način uređivanja datoteka koje čine manifest dodatka je pomoću grafičkog manifest editora. Uređivanje manifest datoteke iz grafičkog sučelja puno je lakše, postoje mnogi čarobnjaci koji olakšavaju posao tako da pružaju unaprijed definirane obrasce koje programer ispunjava umjesto direktnog pisanja XML-a, na taj način programer ne mora znati sve XML atribute i elemente koji su definirani za svaku točku proširenja.

3.1. Manifest.mf

Za svaki dodatak definira se manifest paketa(engl. *Bundle manifest*), pomoću kojeg se definira njegov identifikator, verzija, osnovni razred za pokretanje dodataka (engl. *Activator*), te tko je izradio dodataka(engl. *Provider*). Ti elementi se smještaju u datoteku `manifest.mf`. Primjer `manifest.mf` datoteke dan je ispod:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Test Plug-in
Bundle-SymbolicName: org.eclipse-testPlugin; singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: test.Activator
Bundle-Localization: plugin
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.core.resources,
 org.eclipse.ui.ide,
 org.eclipse.ui.editors,
 org.eclipse.ui.workbench.texteditor
Eclipse-LazyStart: true
Bundle-Vendor: Autor Ovog dodatka
```

Primjer Manifest.mf datoteke

Većina zapisa u datoteci `manifest.mf` definirani su kao "*svojstvo:vrijednost;*"

U svojstva koja definiraju opće podatke o dodatku su:

- Bundle-Name: ime dodatka koji će se prikazivati u Help -> About Eclipse SDK
- Bundle-SymbolicName: simboličko ime slično Java namespace-ovima koje definira dodatak

- Bundle-Version: verzija dodatka, verzije se označavaju koristeći tri broja odvojena točkama, npr. 1.0.10
- Bundle-Activator: ime Java razreda koji definira startanje dodatka
- Bundle-Vendor: ime proizvođača odnosno autora dodatka

U `manifest.mf` u Bundle-ClassPath definira se gdje se nalaze biblioteke(jar datoteke), više putanja do datoteka odvaja se zarezom.

Da bi se dodatak uspješno učitao, prije nego što se on učita provjerava se da li postoje dodaci koje on proširuje, u slučaju da ne postoje prijavljuje se greška i ne nastavlja se učitavanje. Zbog toga je potrebno definirati koji sve dodaci moraju postojati prije nego se naš dodatak učita. To određujemo svojstvom Require-Bundle: iza čega navodimo listu dodataka odvojenih zarezom koji moraju postojati. Osim dodatak koji mora postojati možemo definirati i koja točno verzija mora postojati. Verzije se označavaju slično matematičkom zapisu za intervale, verzije se stavljaju u zagrade gdje uglate zagrade [] predstavljaju **uključujući**, a obične zagrade() **isključujući** verziju, a unutar zagrada se stavljaju dvije verzije odvojene zarezom, gdje prva označava od koje verzije,a druga do koje verzije. Npr. ako želimo da imamo pakete iz verzija od verzije 3.0.0, do 3.2.0, ali bez verzije 3.0.0:

Require-Bundle: org.eclipse.ui, org.eclipse.core.runtime;bundle-version="(3.0.0,3.2.0]"

Ako ne želimo definirati granicu s obje strane možemo reći izostaviti drugu stranu npr. Sve verzije veće od 3.1.0 “[3.1.0,—“. Ako želimo navesti točno određenu verziju za koju je dodatak namjenjen možemo pisati “[3.2.0,3.2.0]”.

Ako neki dodatak izostankom nekog drugog dodatak i dalje radi ali mu nedostaje samo neka funkcionalnost možemo ga označiti opcionalnim tako da dodamo ;resolution:=optional npr:

Require-Bundle: org.eclipse.ui, org.eclipse.core.runtime;resolution:=optional

3.2. Plugin.xml

Datoteka `plugin.xml` sadrži opise proširenja i točaka proširenja u XML obliku. Definicija tog XML dana je s slijedećom DTD definicijom:

```
<?xml encoding="US-ASCII"?>
<?eclipse version="3.2"?>

<!ELEMENT plugin (extension-point*, extension*)>

<!ELEMENT extension-point EMPTY>
<!ATTLIST extension-point
 name                CDATA #REQUIRED
 id                  CDATA #REQUIRED
 schema              CDATA #IMPLIED
>

<!ELEMENT extension ANY>
<!ATTLIST extension
 point               CDATA #REQUIRED
 name                CDATA #IMPLIED
 id                  CDATA #IMPLIED
>
```

XML 3.1: DTD za plugin.xml

Iz ovog DTD-a slijedi jednostavan primjer `plugin.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension-point id="Id"
         name="Point name" schema="schema/point.exsd"/>

   <extension point="org.eclipse.xxx.yyy">
        ...
   </extension>
</plugin>
```

XML 3.2: Primjer općentiog zapisa plugin.xml

U `plugin.xml` definiraju se dvije osnovne stvari proširenja (*extension*) i točke proširenja(*extension-point*).

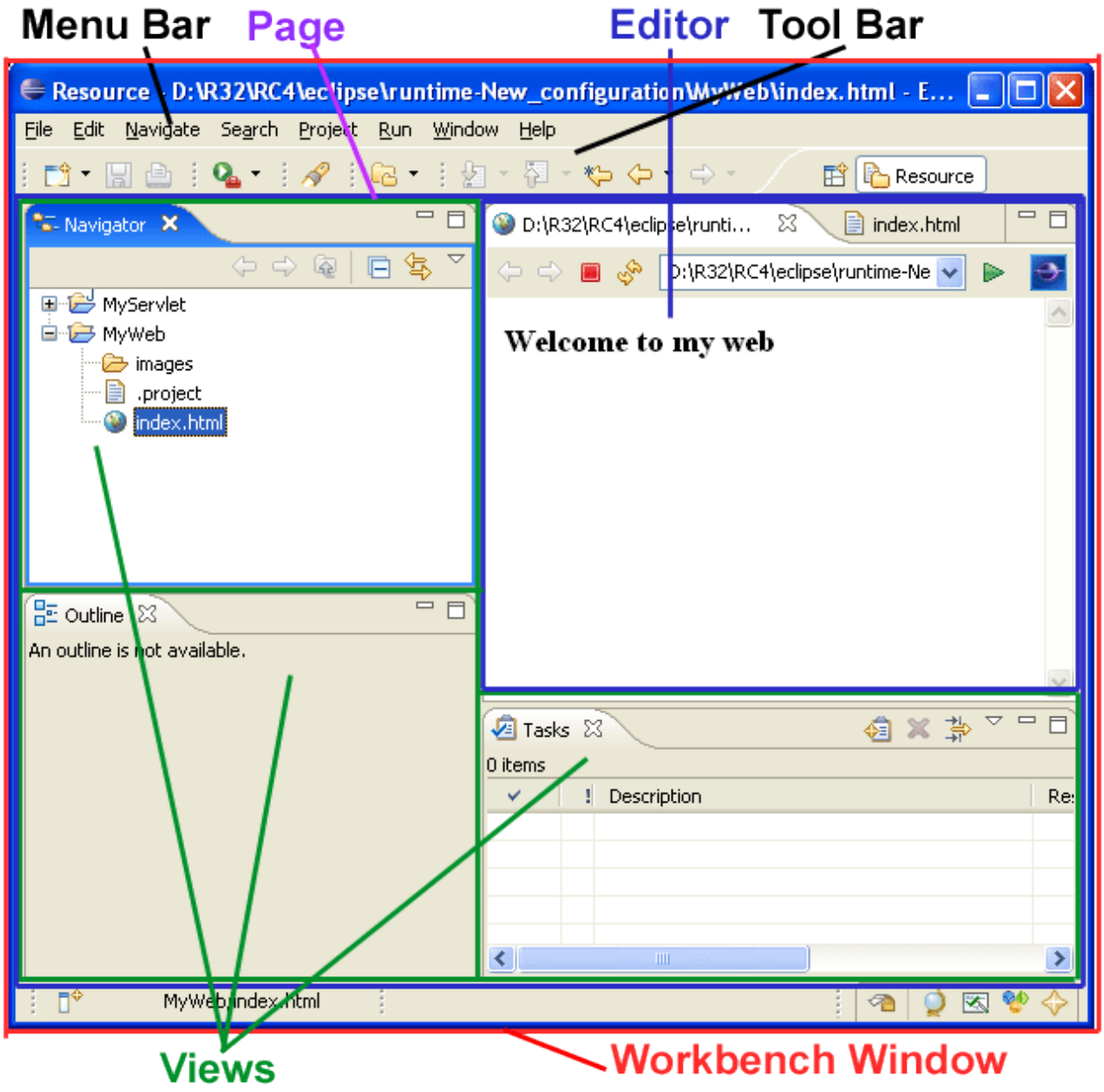
Za proširenje u XML element *extension* potrebno je staviti atribut *point* kojim se definira ime točke proširenja koju želimo proširiti, moguće je stavit atribute *Id* i *name* no oni nisu potrebni. Pod elementi se definiraju drugačije za svaku točku proširenja te svaka točka proširenja definira svoju XML shemu ovisno o podacima koje želi primiti od onoga proširenja.

Za točke proširenja u XML element *extension-point* treba definirati tri atributa *id* – jedinstveni identifikator točke proširenja, *name* – ime točke proširenja i *schema* - putanja do datoteke koja sadrži XML shemu s definicijom pod elementa za tu točku proširenja. Svaka točka proširenja može zahtijevati drugačije atribute, no obično se traži *ID* - identifikator, i *name* - ime onog što pruža točka proširenja. Pristup odvajanja podataka o proširenjima koja se implementiraju od same implementacije omogućuje smanjenu potrošnju memorije zahvaljujući kasnom učitavanju dodataka.

4. Elementi Eclipse korisničkog sučelja

4.1. Pregled osnovnih elemenata

Eclipse korisničko sučelje možemo podijeliti u nekoliko glavnih dijelova, te su oni prikazani na slici 4.1.



Slika 4.1: Elementi Eclipse korisničkog sučelja

Glavni prozor Eclipse okruženja je Workbench window – prozor radno mjesto, u njemu su smješteni ostali elementi sučelja. Radno mjesto pruža velik broj razreda i sučelja za izgradnju složenog korisničkog sučelja, no opet omogućuju jednostavno nadogradnju korisničkog sučelja.

Radno mjesto(*workbench*) je osnovni prozor u kojem su sadržani alatne trake s gumbima, izbornicima, statusna liniju, stranice (engl. Page). Radno mjesto implementira `IWorkbenchWindow` sučelje. Osnovni, korisniku važni elementi sučelja u radnom mjestu su pogledi i uređivači.

Korisničko sučelje Eclipse Platform temelji se na pogledima, uređvačima i izgledima. S korisnikove strane, prozor workbench-a vizualno se sastoji od pogleda i uređivača, Dok su izgledi zapravo trenutno prikazani pogledi i uređivači na radnoj površini i njihov međusobni raspored. Dodaci također mogu proširiti postojeće uređivače, poglede i izglede.

4.1.1. Izgledi

Izgledi (engl. *Perspectives*) pružaju dodatni sloj organizacije unutar radnog mjesta. Izgled definira skup pogleda, te njihov razmještaj, dozvoljena djelovanja za određeni zadatak. Korisnik može mijenjati izglede ovisno o zadatku kojeg žele rješavati. Sa strane implementacije trenutno aktivan izgled kontrolira koji će pogledi biti prikazani na radnom mjestu njihove pozicije i veličine. Uređivači nisu obuhvaćeni promjenom izgleda.

Prozor radno mjesto ima više odvojenih izgleda, od kojih je samo jedan vidljiva u danom trenutku. Svaki izgled ima svoje vlastite poglede i uređivače koji su uređeni (tiled, stacked ili detached) za prezentaciju na ekranu (neki, prema potrebi, mogu biti skriveni).

4.1.2. Pogledi i uređivači

Pogled (engl. *View*) i uređivač (engl. *Editor*) su osnovni elementi Eclipse sučelja, te se kroz njih vrši obrada i prezentacija podataka koji se pomoću njih obrađuju ili prezentiraju.

Pogled nude informacije o objektima na kojima korisnik radi u radnom prostoru, se obično koristi za navigaciju kroz hijerarhiju informacija, za otvaranje uređivača, prikaz svojstva elementa iz aktivnog uređivača. Pogled može pomagati uređivaču davanjem dodatnih informacija o dokumentima koji se uređuju. Na primjer, pogled **properties** prikazuje informacije o objektima u trenutno aktivnom uređivaču.

Uređivač se obično koristi za uređivanje ili pregled dokument, on dopušta korisniku da otvori, obradi i spremi objekte koje uređuje. Glavno svojstvo uređivača da se ponaša po otvori-spremi-zatvori modelu, te je njegovo ponašanje isto standardnim uređivačima teksta koji obrađuju tekstualne datoteke. Eclipse Platform osigurava standardni editor za tekstualne resurse, dok su specijalizirani uređivači implementirani drugim dodacima.

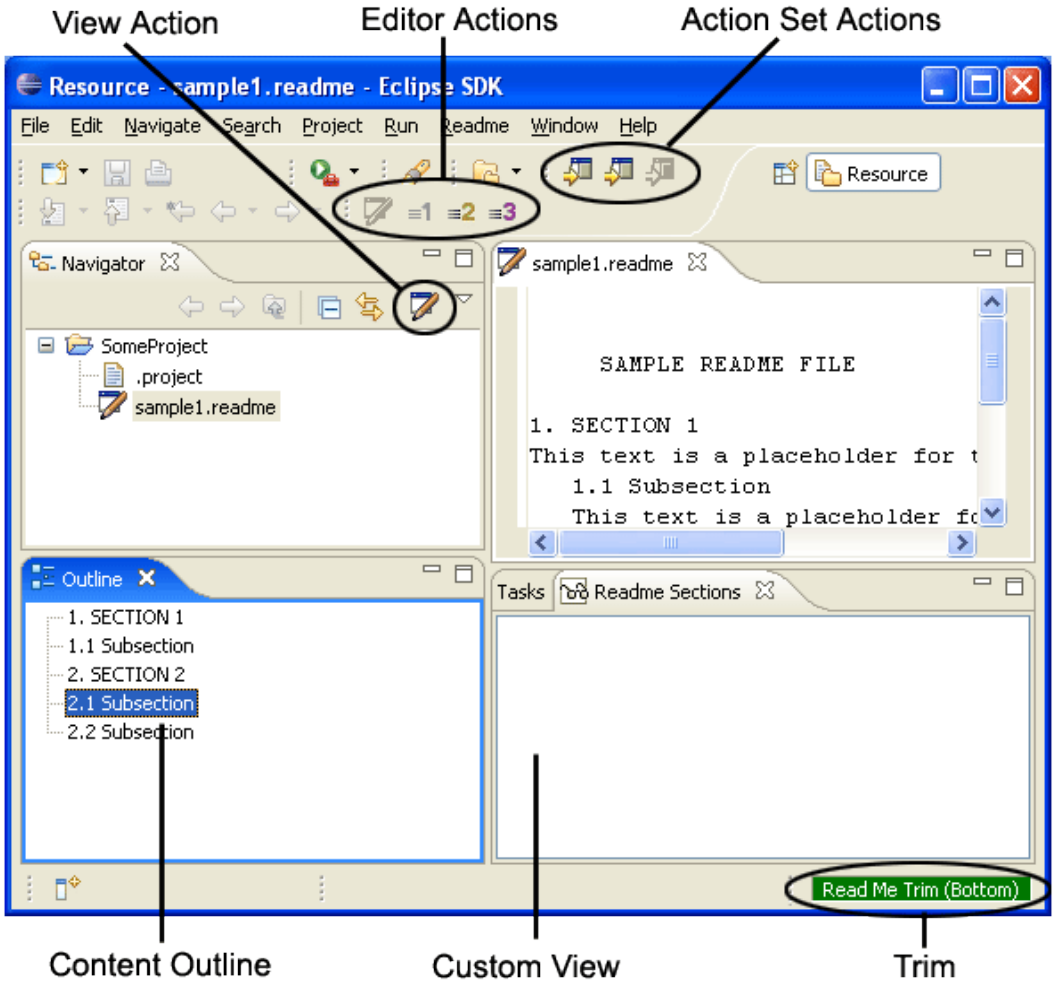
Životni ciklus jednog pogleda ili uređivača svodi se na sljedeće korake:

- nasljeđivanje odgovarajućeg nadrazreda (ViewPart ili EditorPart)
- implementacija metode `createPartControl` koja stvara SWT widgete koji predstavljaaju vizualne komponente sučelja, ovdje se vrše sva ostala alokacija potrebnih resursa
- implementacija metode `setFocus` koja definira ponašanje kad naš pogled ili uređivač dobije fokus
- implementacija metode `dispose` koja definira ponošenje kod zatvaranja pogleda ili uređivača, ovdje je potrebno osloboditi resurse koje smo ručno alocirali(npr. Dodatne fontove, kursore, grafičke elemente poput slika i slično)

4.2. Radnje

Radnje (engl. *Actions*) kao i sve ostalo definirano je kroz nekoliko točaka proširenja tako da se nove radnje mogu jednostavno dodavati. Radnje se javljaju na nekoliko različitih mjesta u Eclipse IDE-u: izbornicima, alatnim trakama, context izbornicima. Radnje se isto tako mogu vezati za određeni pogled (*View Actions*) ili uređivač(*Editor Actions*) kako je prikazano na slici 4.2.

Radi lakše organizacije radnje se grupiraju u skupove radnji (engl. *ActionSets*), koji su skupine logički povezanih radnji(npr. manipulacija datotekama, pokretanje projekata,...).

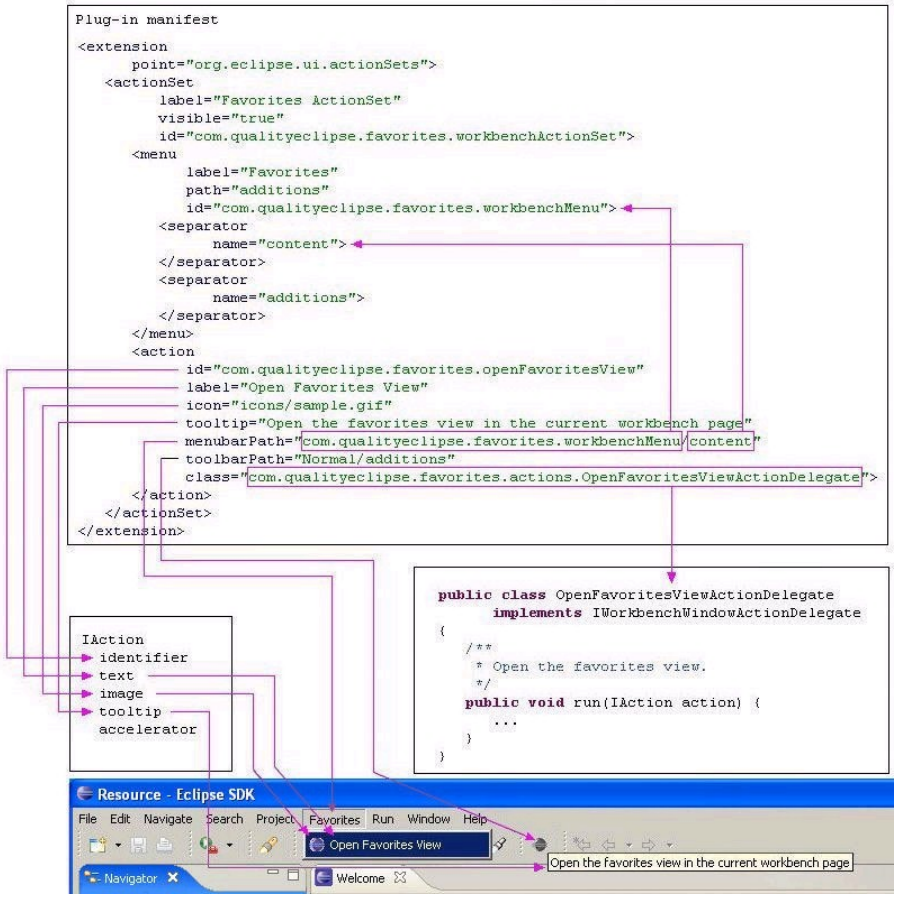


Slika 4.2: Smještanje radnji na radno mjesto

Eclipse radnja sastoji se od dva dijela XML deklaracije radnje u manifestu dodatka i razreda koji implementira `IActionDelegate` sučelje koji sadrži kod koji se izvodi kod izvršenja željene radnje. Potrebno je definirati dvije metode za `IActionDelegate` to su `run(IAction action)` gdje se nalazi kod kod aktivranja te radnje i `selectionChanged(IAction action, ISelection`

`selection)` ova metoda služi za međusobnu komunikaciju između više različitih Eclipse komponenata.

Definiranjem akcije XML dijelu manifesta dodatka Eclipse generira objekt koji implementira `IAction` sučelje te on preslikava sadržaj XML-a u taj objekt. Slika 4.3. prikazuje, na jednostavnom primjeru, povezivanje između XML manifesta Eclipse sučelja, `IAction` objekta i `IActionDelegate` željene implementacije



Slika 4.3: Vezanje XML deklaracije manifesta na IAction i IActionDelegate

Odvajanje `IAction` objekta, kojeg definira i instancira Eclipse korisničko sučelje, i same implementacije koja se mora izvesti u `IActionDelegate` objektu omogućava Eclipse-u prikaz radnje i izborniku ili alatnoj traci bez učitavanja sadržaja dodataka tako dugo dok korisnik ne odabere željenu radnju iz izbornika ili ne pritisne gumb iz alatne trake. Ovaj način se zove kasno učitavanje dodataka(engl. *lazy plug-in initialization*) i omogućava Eclipse-u korištenje velikog broja dodataka, bez povećanja potrošnje memorije ako se ne koriste svi dodaci u isto vrijeme što je najčešći slučaj.

Postoji nekoliko `IActionDelegate` podvrsta:

- `IActionDelegate2` : naprednija varijanta `IActionDelegate` sučelja i koristi se kad su potrebno dodatne informacija o životnom ciklusu dodatka, poput kad napraviti oslobađanje zauzetih resursa prije nego se dodatak oslobodi iz memorije
- `IEditorActionDelegate` : pruža dodatne informacije o događajima koji su povezani uređivačem

- `IObjectActionDelegate` : pruža dodatne informacije o događajima koji su povezani s elementima padajućih izbornika
- `IViewActionDelegate` : pruža dodatne informacije o događajima koji su povezani s pogledima
- `IWorkbenchWindowActionDelegate` : pruža dodatne informacije o događajima koji su povezani s elementima radnog mjesta kao što su izbornici i alatne trake

Primjer dodavanja radnje:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>

   <extension
         point="org.eclipse.ui.actionSets">
      <actionSet
            label="Sample Action Set"
            visible="true"
            id="testPlugin.actionSet">
         <menu
               label="Sample &amp;amp;Menu"
               id="sampleMenu">
            <separator
                  name="sampleGroup">
            </separator>
         </menu>
         <action
               class="test.actions.DoSomethigTest"
               icon="icons/sample.gif"
               id="test.actions.DoSomethigTest"
               label="&amp;amp;Sample Action"
               menubarPath="sampleMenu/sampleGroup"
               style="radio"
               toolbarPath="sampleGroup"
               tooltip="Hello, Eclipse world">
         </action>
      </actionSet>
</plugin>
```

XML 4.1: Primjer jednostavnog dodatka koji dodaje jednu radnju u izbornik

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Test Plug-in
Bundle-SymbolicName: testPlugin; singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: test.Activator
Bundle-Localization: plugin
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.core.resources,
 org.eclipse.ui.ide
Eclipse-LazyStart: true
Bundle-Vendor: Autor Ovog dodatka
```

```
package test.actions;

import ...

/**
 * Primejr korištenja IWorkbenchWindowActionDelegate za jednostavnu radnju
 */
public class DoSomethigTest implements IWorkbenchWindowActionDelegate {
	private IWorkbenchWindow window;
	/**     * Konstruktor       */
	public DoSomethigTest() {
	}

	/**
	 * Radnja je aktivirana. Ovdje se definira stvarni posao
	 * koji treba izvrsiti ovdje se ispisuje dialog s Hello World
	 */
	public void run(IAction action) {

		MessageDialog.openInformation(
			window.getShell(),
			"Test Plug-in",
			"Hello, Eclipse world");
	}

	/**    dio koji obrađuje promjenu selekcije     */
	public void selectionChanged(IAction action, ISelection selection) {
	}

	/** Ako je potrebno osloboditi neki resurs to radimo ovdje    */
	public void dispose() {
	}

	/**
	 * Ovdje se prosljeđujemo sučelje glavnog prozora radnog mjesta
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}
}
```

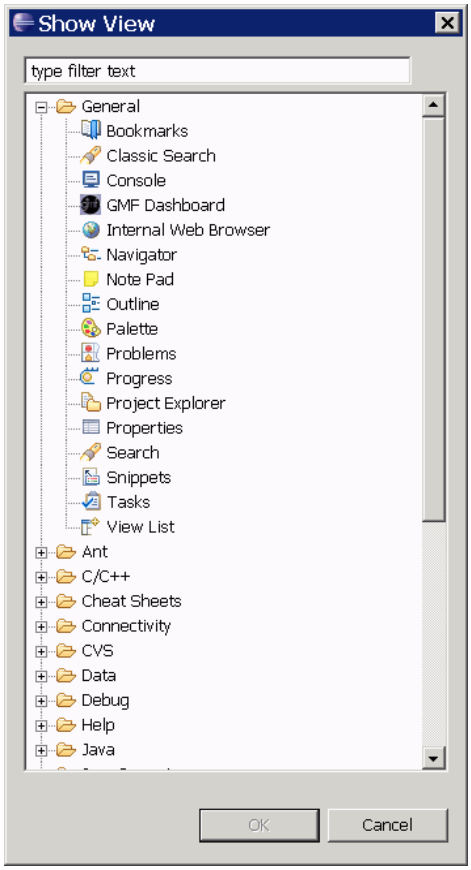
Java 4.1: Primjer dodavanja radnje

4.3. Pogledi

Pogledi su jedan od glavnih načina proširivanja Eclipse korisničkog sučelja jer služe za prikaz informacija. Ovo poglavlje opisuje kako dodati novi pogled u Eclipse korisničko sučelje.

Pogledi dijele skup zajedničkih ponašanja s uređivačima implementiranih preko nadrazreda `org.eclipse.ui.part.WorkbenchPart` i sučelja `org.eclipse.ui IWorkbenchPart`. No postoje i razlike između uređivača i pogleda. Kod pogleda svaka izvedena radnja rezultira trenutnu promjenu stanja radnog prostora i podređenih sredstava, dok uređivači slijede klasičnu otvori-promjeni-spremi metodologiju. Uređivači se pojavljuju samo na jednom mjestu(sredini) Eclipse korisničkog sučelja te se njihova lokacija ne može promijeniti, dok pogledi su razmješteni oko uređivača te se njihova lokacija može mijenjati pomicanjem na željeno mjesto.

Kako za Eclipse radno mjesto može postojati više pogleda, a svi nisu istovremeno potrebni, pomoću dialoga Show View(slika 4.4) možemo odabrati koji pogled želimo aktivirati. Radi lakšeg snalaženja pogledi su grupirani u kategorije.



Slika 4.4: Grupiranje pogleda u kategorije - dailog Show View

Dodavanje novog pogleda sastoji se od tri koraka:

- definiranje kategorije kuda pripada pogled u manifestu dodatka
- definiranje pogleda u manifestu dodataka
- stvaranje dijela pogleda koji sadržava samu programsku implementaciju

Definiranje kategorije u svodi se na proširivanje točke proširenja `org.eclipse.ui.views` dodavanjem novog XML pod elementa category u `plugin.xml` . Kako pokazuje XML 4.2 za kategoriju je potrebno dodati dvije stvari

- id: određuje jedinstveni identifikator kategorije, te ovaj atribut mora biti jedinstven
- name: ime kategorije koje je čitljivo za čovjeka, ime će biti prikazano u izbornicima i dialoškom okviru Show View

```
<extension
      point="org.eclipse.ui.views">
   <category
         id="testPluginKategorija1"
         name="Test kategorija"/>
   <view
         allowMultiple="true"
         category="testPluginKategorija1"
         class="testplugin.views.ednostavanPogled"
         icon="icons/sample.gif"
         id="testplugin.views.JednostavanPogled1"
         name="Jednostavan Pogled">
      <description>
         Ovo je primjer jednostavnog pogleda
      </description>
   </view>
</extension>
```

XML 4.2: Primjer dodavanja pogleda i kategorije pogleda

Da bi definirali pogleda u manifestu dodataka potrebno je proširiti točku proširenja `org.eclipse.ui.views` dodavanjem novog XML pod elementa view u `plugin.xml`. Kako XML 4.2. prikazuje, potrebno je dodati XML element view s sljedećim obaveznim atributima:

- id: jedinstveni identifikator pogleda, mora biti jedinstven
- name: ime pogleda koje će biti prikazano korisniku u Eclipse sučelju
- class: ime i namespace Java razreda koji implementira funkcionalnost pogleda
- category: id kategorije kojoj pogled pripada, u slučaju nepostojeće kategorije pogled neće biti prikazan

i slijedećim opcionalnim:

- allowMultiple: da li je omogućeno otvaranje više instanci istog pogleda
- icon: putanja do ikone koja se pridružuje pogledu, te će biti prikazana u Eclipse sučelju za pogled
- description(element): opis namjene pogleda

Nakon definiranja XML za pogled potrebno je izraditi Java razred koji implementira `org.eclipse.ui.part.IViewPart` sučelje. Da bi implementirali ovo sučelje potrebno je ostvariti slijedeće metode:

- `createPartControl(Composite)`: ova metoda stvara SWT elemente korisničkog sučelja te određuje izgled pogleda.
- `Dispose()`: metoda se poziva kod uništavanja pogleda te je ona zadužena za oslobađanje sredstava koji su ručno zauzeti od strane pogleda
- `saveState(IMemento)`: ova metoda se poziva kod spremanja stanja pogleda. Stanje pogleda može biti trenutni odabir, sortiranje po određenom ključu, trenutno uključeni filtar.
- `SetFocus()`: ova metoda je postavlja fokus na odgovarajuću SWT kontrolu kad pogled dobije fokus.

Umjesto implementiranja ovog sučelja, možemo naslijediti i `org.eclipse.ui.part.ViewPart` razred koji implementira `IViewPart` sučelje s praznim metodama, i tada preopteretimo metode koje

moramo implementirati npr. `createPartControl(Composite)`. Nasljeđivanje razreda `ViewPart` može se vidjeti u primjeru Java 4.2.

```
package testplugin.views;

import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.part.ViewPart;

public class JednostavanPogled extends ViewPart {

	@Override
	public void createPartControl(Composite parent) {
		// TODO Ovdje dodati SWT grafičke elemente
	}

	@Override
	public void setFocus() {
		// TODO Auto-generated method stub
	}
}
```

Java 4.2: Primjer jednostavnog pogleda, koji ne sadrži nikakve grafičke elemente

4.4. Uređivač

Uređivač je osnovni mehanizam preko kojeg korisnici mijenjanju resurse (npr. tekstualne datoteke). Eclipse pruža nekoliko osnovnih uređivača za uređivanje Java izvornog koda, uređivač za obradu teksta, ali ima i složene uređivače koji se sastoje od više strana, primjer takvog uređivača je manifest uređivač PDE-a. U ovom poglavlju biti će objašnjeni kako napraviti uređivač.

Slično kao i pogledi uređivači moraju implementirati `org.eclipse.ui.IEditorPart` sučelje, no u praksi je praktičnije naslijediti razred `org.eclipse.ui.part.EditorPart` te na taj način je veći dio uobičajenog ponašanja naslijeđen, i ne treba ga ponavljati isti posao.

Uređivači se pojavljuju samo na jednom mjestu radnog mjesta i nije ih moguće pomaknuti,obično služe za uređivanje nekog sredstva (npr. tekstualne datoteke).

Dodavanje novog uređivača sastoji se od dva koraka:

- definiranje uređivača u manifestu dodataka
- pisanje programskog koda koji sadržava implementaciju uređivača

```
<extension
      point="org.eclipse.ui.editors">
   <editor
         class="testplugin.editors.MultiPageEditor"
         contributorClass="testplugin.editors.MultiPageEditorContributor"
         extensions="txt"
         icon="icons/sample.gif"
         id="testplugin.editors.MultiPageEditor"
         name="Jednostavan Multi-page Editor"/>
 </extension>
```

XML 4.3: Primjer proširivanje uređivača dodajući višestrani uređivač

Da bi definirali novi uređivač, slično kao i kod dodavanja novog pogleda proširujemo točku proširenja `org.eclipse.ui.editors` dodajem XML element editor s slijedećim atributima:

- class: ime Java razreda koji implementira uređivač implementirajući sučelje `org.eclipse.ui.IEditorPart` ili nasljedujući razred `EditorPart` u, alternativno je moguće koristiti i `MultiPageEditorPart` koji je proširenje običnog `EditorPart` s dodanom mogućnosti da imam više stranica(tabova), primjena više stranica prikazana je u primjerom.
- ContributorClass: ime Java razreda koji implementira `org.eclipse.ui.IEditorActionBarContributor` koji služi za dodavanje novih radnji koje postaju dostupne kad je uređivač aktivan. Radnje se smještaju kao izbornici, alatne trake ili gumbi u alatnim trakama
- extensions: ovdje dolazi lista ekstenzija datoteka odvojene točka-zarezom za koje će se aktivirati uređivač (npr. ”txt;csv” će otvoriti naš uređivač za dototeke s završetkom txt i csv)
- icon: određuje se putanja do ikone koja će biti prikazana u lijevom gornjem kutu uređivača
- id: jedinstven identifikator uređivača
- name: ime uređivača koje će pisati u Eclipse okruženju

Ostali atributi koji nisu korišteni u primjeru:

- command: naredba koja pokreće vanjski editor, izvršna datoteka koja se pokreće naredbom mora biti smještena u sistemskom putu (engl. *System PATH*) ili direktoriju dodatka. Atributi class, command i launcher međusobno se isključuju.
- default: ima vrijednost “true” ili “false” (prazno= false). Ako je “true” uređivač će biti korišten kao pred definirani za određeni tip datoteke. Ovo je bitno samo kad je za određenu vrstu datoteke definirano više uređivača. Ako uređivač nije pred definirani on se i dalje može koristiti za taj tip datoteka pokretanjem iz padajućeg izbornika Open with za određenu datoteku
- filenames: lista datoteka odvojenih zarezom za koje se registrira uređivač za njihovo uređivanje. Primjer takve registracije je manifest uređivač PDE-a koji se registirai sa `plugin.xml, fragment.xml,manifest.mf`.
- launcher: razred koji implementira `org.eclipse.ui.IEditorLauncher` sučelje, koje ostvaruje otvaranje eksterni uređivača

Nakon definiranja XML deklaracije u *plugin.xml*, treba izraditi Java kod tako da naslijedimo EditorPart ili MultiPageEditorPart. Pošto MultiPageEditorPart nasljeđuje EditorPart sve metode koje je potrebno implementirati za EditorPart potrebno je implementirati i u MultiPageEditorPart.

Za EditorPart potrebno je implementirati slijedeće metode:

- `createPartControl(Composite)` : ova metoda stvara SWT elemente korisničkog sučelja te određuje izgled uređivača.
- `dispose()`: metoda se poziva kod uništavanja pogleda te je ona zadužena za oslobađanje sredstava koji su ručno zauzeti od strane pogleda
- `doSave(IProgressMonitor)`: ova metoda mora implementirati spremanje sadržaja uređivača. Ako je sadržaj uspješno spremljen treba generirati događaj “property changed” označavajući novo stanje da li ima promjena na uređivaču.

- `doSaveAs()` : ova metoda je opcionalna. Ona otvara Save As dialog i sprema sadržaj uređivača na novu lokaciju. Ako je sadržaj uspješno spremljen treba generirati događaj "property changed" označavajući novo stanje da li ima promjena na uređivaču.
- `gotoMarker(IMarker)`: postavlja kursor na mjesto koje označava mjesto selekcije koje je određeno danim markerom.
- `isDirty()`: vraća da li je bilo promjena od zadnjeg spremanja sadržaja
- `isSaveAsAllowed()`: vraća da li je dozvoljeno pozvati metodu Save As
- `setFocus()`: postavlja fokus na neku od kontrola na uređivaču.

MultiPageEdittorPart pruža slijedeće dodatne metode koje treba imlemenitrati:

- `addPage(Control)`: Dodaje novu stranicu koja sadrži kontrole za uređivač s više stranica
- `addPage(IEditorPart, IEditorInput)`: Stvara i dodaje novu stranicu koja sadrži uređivač. Ova metoda koristi se da bi se dodao uređivač koji je namijenjen kao jednostrani i tako omogućilo korištenje jednostavnih uređivača u nekom složenijem uređivaču
- `createPages()`: stvara stranice višestranog uređivača, obično se za svaku stranicu radi posebna metoda te se u ovoj metodi one pozivaju npr. `CreatePage0()`; `CreatePage1()`;, ova metoda zamjenjuje `createPartControl(Composite)` običnog uređivača.
- `getContainer()`: vraća composite kontrolu koja sadrži stranice više stranog uređivača. Ova metoda se korisit kod stvaranja kontrola ta pojedinu stranicu, ovaj composite treba biti roditelj za svaku stranicu koja se dodaje s `addPage(Control)`
- `setPageImage(int, Image)`: postavlja se slika za stranicu s zadanim indeksom
- `setPageText(int, String)`: postavlja se tekst za stranicu s zadanim indeksom

Java primjer višestranog uređivača je dan s Java 4.3 i Java 4.4 odsječcima. Java 4.3. prikazuje specifične detalje višestrani uređivač kao što su manipulacije stranicama, dok Java 4.4. odsječak prikazuje elemente koji se odnose općenito na uređivače na spremanje dokumenata.

```
public class MultiPageEditor extends MultiPageEditorPart{
	/** Tekstualni uređivač na stranici  0. */
	private TextEditor editor;
	/** Text widget za prikaz teksta na stranici 1. */
	private StyledText text;
	/** konstruktor */
	public MultiPageEditor() {
		super();
	}
	/** Stvara stranicu 0 koja je tekst uređivač */
	void createPage0() {
		try {
			editor = new TextEditor();
			int index = addPage(editor, getEditorInput());
			setPageText(index, editor.getTitle());
		} catch (PartInitException e) {
			ErrorDialog.openError(
				getSite().getShell(),
				"Error creating nested text editor",
				null,
				e.getStatus());
		}
	}

	/** Stvara prvu stranicu višestrukog uređivača u kojoj se
	 * prikazuje tekst u StledText elementu s određenim fontom
	 */
	void createPage1() {
		Composite composite = new Composite(getContainer(), SWT.NONE);
		FillLayout layout = new FillLayout();
		composite.setLayout(layout);
		text = new StyledText(composite, SWT.H_SCROLL | SWT.V_SCROLL);
		text.setEditable(false);
		int index = addPage(composite);
		setPageText(index, "Preview");
	}
	/** Stvara pojedine stranice višestranog uređivača. */
	protected void createPages() {
		createPage0();
		createPage1();
	}
	/*** Oslobađaju se zauzeti resursi nested editors. */
	public void dispose() {
		super.dispose();
	}
	/**
	 * Pri promjeni stranice postavlja se sadržaj za odgovarajuću stranicu
	 */
	protected void pageChange(int newPageIndex) {
		super.pageChange(newPageIndex);
		if (newPageIndex == 1) {
text.setText(editor.getDocumentProvider().getDocument(editor.getEditorInput(
)).get());
		}
	}
```

Java 4.3: Primjer višestranog uređivača manipulacija stranicama

```
	/**
	 * Spremnaje uređivanog dokumenta. Poziva se spremanje
	 * ugrađenog uređivača za obradu teksta
	 */
	public void doSave(IProgressMonitor monitor) {
		getEditor(0).doSave(monitor);
	}
	/**
	 * Sprema dokument s Save As. Poziva se spremanje
	 * ugrađenog uređivača za obradu teksta
	 */
	public void doSaveAs() {
		IEditorPart editor = getEditor(0);
		editor.doSaveAs();
		setPageText(0, editor.getTitle());
		setInput(editor.getEditorInput());
	}

	/**
	 * Spremanje dokumenta je uvjek dozvoljeno
	 */
	public boolean isSaveAsAllowed() {
		return true;
	}


}
```

Java 4.4: Primjer višestranog uređivača definicij spremanja dokumenta

4.5. Izgledi

Izgledi (engl. *Perspectives*) definiraju kako će se na radnoj površini razmjesti skup pogleda i uređivača. Izgledi se definiraju za pojedine zadaće koji se korisnik Eclipse okruženja vrši i pojedinom trenutku, za svaku zadaću definira se koji pogledi i uređivači će biti prikazani i koji je njihov razmještaj na radnoj površini. Ovisno o zadaći koja se vrši izgled možemo izgraditi cijeli novi, ili možemo nadograditi veće neki postojeći izgled.

Dodavanje izgleda svodi se na proširivanje `org.eclipse.ui.perspective` dodavanjem odgovarajućeg XML elementa u plugin.xml (primjer. XML 4.4) i generiranje odgovarajućeg Java razreda koji vrši samo raspoređivanje elemenata izgleda.

```
<extension
      point="org.eclipse.ui.perspectives">
   <perspective
         class="test.PrimjerIzgleda"
         id="test.perspective1"
         name="Izgled za testiranje"/>
 </extension>
```

XML 4.4: Primjer dodavanja novog izgleda

Nakon XML dijela sam raspored definiramo u Java kod tako da definiramo razred koji implemenitra `IPerspectiveFactory`, ovdje moramo implementirati samo jednu metodu `createInitialLayout(IPageLayout layout)` u kojoj postavljamo raspored. U ovoj metodi raspored(layout) dobivamo kao parametar metode. `IPageLayout` sučelje nam dozvoljava korištenje slijedeće metode:

- `addView(String, int, float, String)` : dodaje pogled određenog imena, na određenu poziciju relativno od nekog elementa, relativne veličine
- `addStandaloneView(String, boolean, int, float, String)` : isto kao i `addView(...)` s razlikom da se može definirati da pogled ne može biti zatvoren i njegova pozicija ne može biti promjenjena.
- `getEditorArea()` : Vraća ID uređivača čime je određena, obično se koristi da bi se pogledi mogli relativno smjestiti u odnosu na uređivač
- `addFastView(String)` :pogled s ID dodaje kao Fast View .
- `addFastView(String, float)` : pogled s ID dodaje kao Fast View i određene relativne veličine
- `setEditorAreaVisible(boolean)` : postavlja da li je uređivač vidljiv.

Slika 4.5: Primjer jednog izgleda s dva definirana pogleda i uređivačem

Primjer jednog pogleda dan je slikom 4.5. a Java kod koji implementira taj izgled dan je primjerom Java 4.5.

```
public class PrimjerIzgleda implements IPerspectiveFactory {

	private static final String BROWSER_VIEW_ID="org.test.browser";
	private static final String PROBLEMS_VIEW_ID="org.test.problems";

	public void createInitialLayout(IPageLayout layout) {
		//dohvati adresu uređivača
	   String editorArea = layout.getEditorArea();
	   //stavi BROWSER pogled na trećinu ekrana lijevo od uređivača
	   layout.addView(BROWSER_VIEW_ID,
	               IPageLayout.LEFT, 0.33f, editorArea);
	   //stavi PROBLEMS na četvrtinu ekrana ispod uređivača
	   layout.addView(PROBLEMS_VIEW_ID,
	                IPageLayout.BOTTOM, 0.25f, editorArea);

	}
}
```

Java 4.5: Primjer pogleda

5. PDE (Plug-in Development Environment)

Okruženje za razvoj dodataka - PDE (engl. *Plug-in Development Environment)* je skup alata koji omogućuju programeru razvoj, testiranje, ispitivanje, izgradnju izvršnih arhiva i isporuku Eclipse dodataka, fragmenata, mogućnosti(engl. *features*) i RCP aplikacija. PDE je izgrađen kao dodatak Eclipse IDE razvojnom okruženju, te se isporučuje kao sastavni dio Eclipse SDK-a.

PDE je podijeljeni u dva dijela: PDE UI , PDE Build od koji je svaki zadužen za određeni dio izrade Eclipse dodataka.

PDE UI zadužen je za za korisničko sučelje koje programer treba olakšati izradu dodataka. Za taj posao služe sljedeći alati :

- Uređivač manifest datoteke bazirana na formularima – više strani editor koji se brine za sve manifest datoteke dodatka ili mogućnosti
- RCP alati – skup čarobnjaka i uređivača baziranih na formularima za definiranje, izradu brand-a, testiranje i izvoz produkata na različite operacijske sustave
- Čarobnjaci za izradu novih projekata – izrada novih dodataka, fragmenata, mogućnosti
- Čarobnjaci za uvoz projekata s datotečnog sustava
- Čarobnjaci za izvoz dodataka i izgradnju u arhive
- Launchers – pokretači koji pojednostavljuju testiranje, debugiranje Eclipse aplikacija i dodataka.
- Pogledi za jednostavniji razvoj dodataka.
- Integracija s JDT-om te mogućnost refactoringa i pretrage

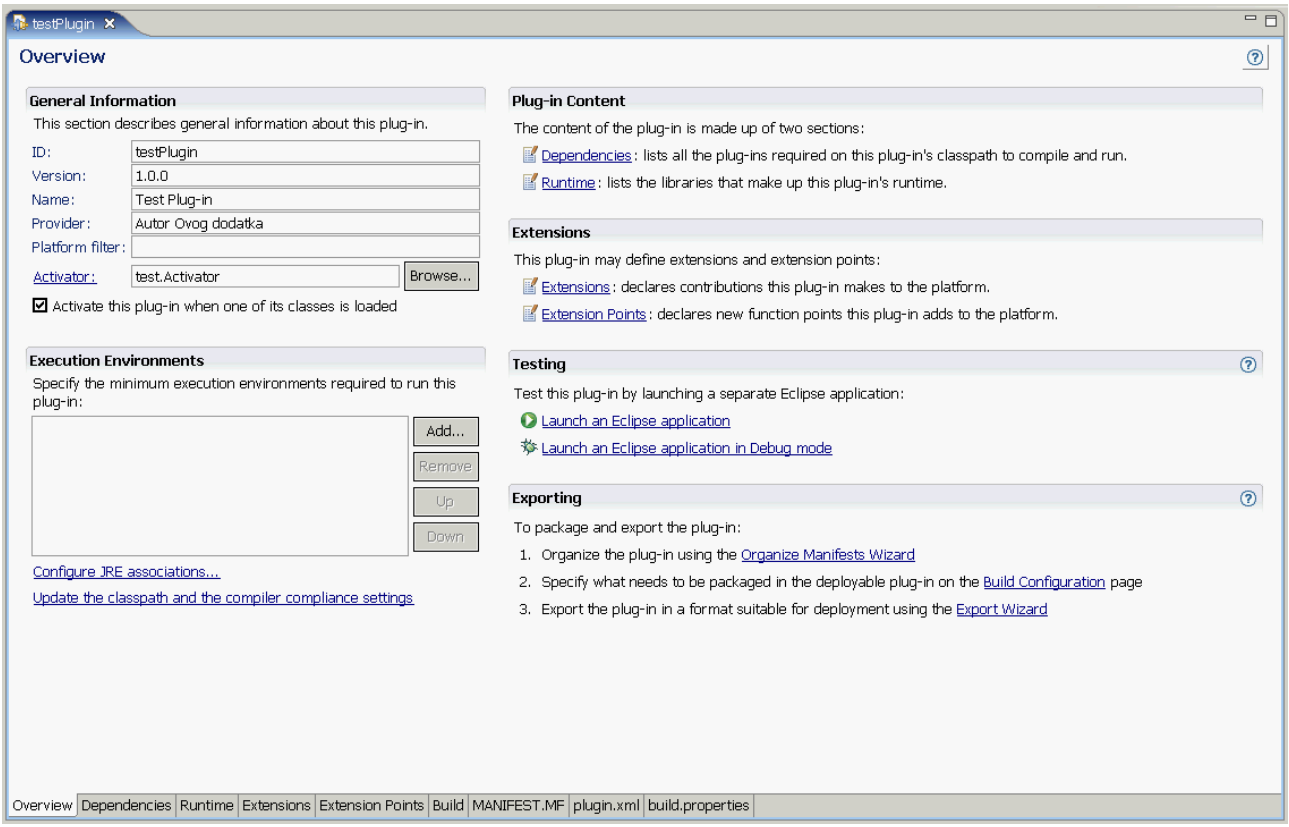
Glavna zadaća PDE Build olakšavanje procesa automatizacije procesa izgradnje (engl. *build processes*) dodataka. PDE Build generira interne ANT skripte koje na temelju informacija koje se zadaju za vrijeme razvoja dodatka, npr. iz `plugn.xml`. Generirana Ant skripta dohvaća sve relevantne izvorne kodove,vrši prevođenje, gradi jar arhive, sprema sve zajedno u formatu spremnom za isporuku.

Glavni alata PDE je manifest uređivač, on se sastoji od više strana:

Popis strana:

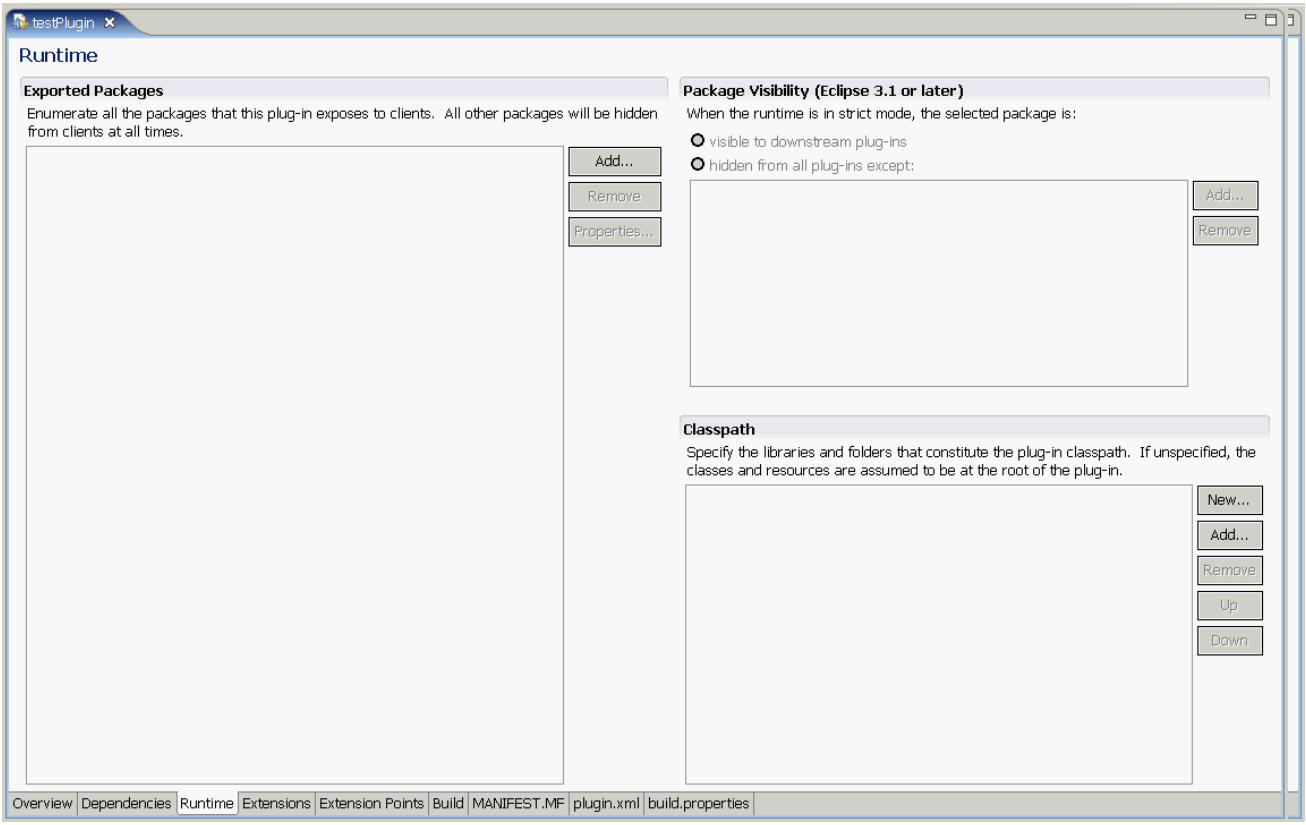
- Overview – prikazuje sažetak informacija o dodatku kao što su ime dodatak, verzija, provider (Slika 5.1)
- Dependencies – detalji o dodacima koji su potrebni da bi se mogao pokrenuti nas dodatak, njihovim međuovisnostima, ovdje možemo provjeriti da li postoje nepotrebne i kružne međuovisnosti (Slika 5.2)
- Runtime – definiranje dodatnih biblioteka, biblioteka koje se žele eksportirati (Slika 5.3)
- Extensions – dodavanje novih proširenja, uređivač parametara proširenja (Slika 5.5)
- Extensions Points – dodavenje novih točaka proširenja (Slika 5.6)
- Build – odabir datoteka koje će biti isporučene i ugrađene u paket dodatka (Slika 5.7)

- MANIFEST.MF – tekstualni prikaz manifest.mf datoteke
- plugin.xml – tekstualni prikaz XML plugin.xml datoteke
- build.proproites -tekstualni prikaz odabira iz strane Build



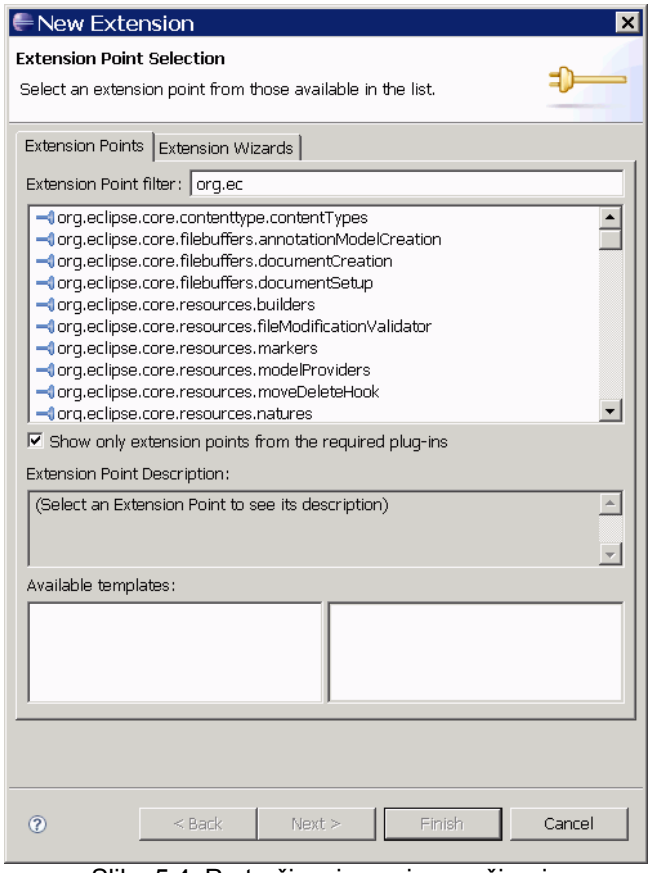
Slika 5.1: Pregled i osnovni podaci o dodatku

Lista potrebnih dodataka određuje biblioteke koji će biti u plug-in runtime class pathu. Nakon spremanja manifest datoteke, PDE automatski obnavlja class path projekta tako da sve promjene utječu na listu međuovisnosti, ako takva postoji. Lista potrebnih dodataka ne bi trebala sadržavati neiskorištene stavke. Takve stavke teže je uočiti, a one smanjuju performanse jer povećavaju potrošnju memorije, pa PDE nudi opciju Find Unused Dependencies(Slika 5.2) mogućnost koja traži stavke koje su nepotrebne te nudi mogućnost njihova micanja.



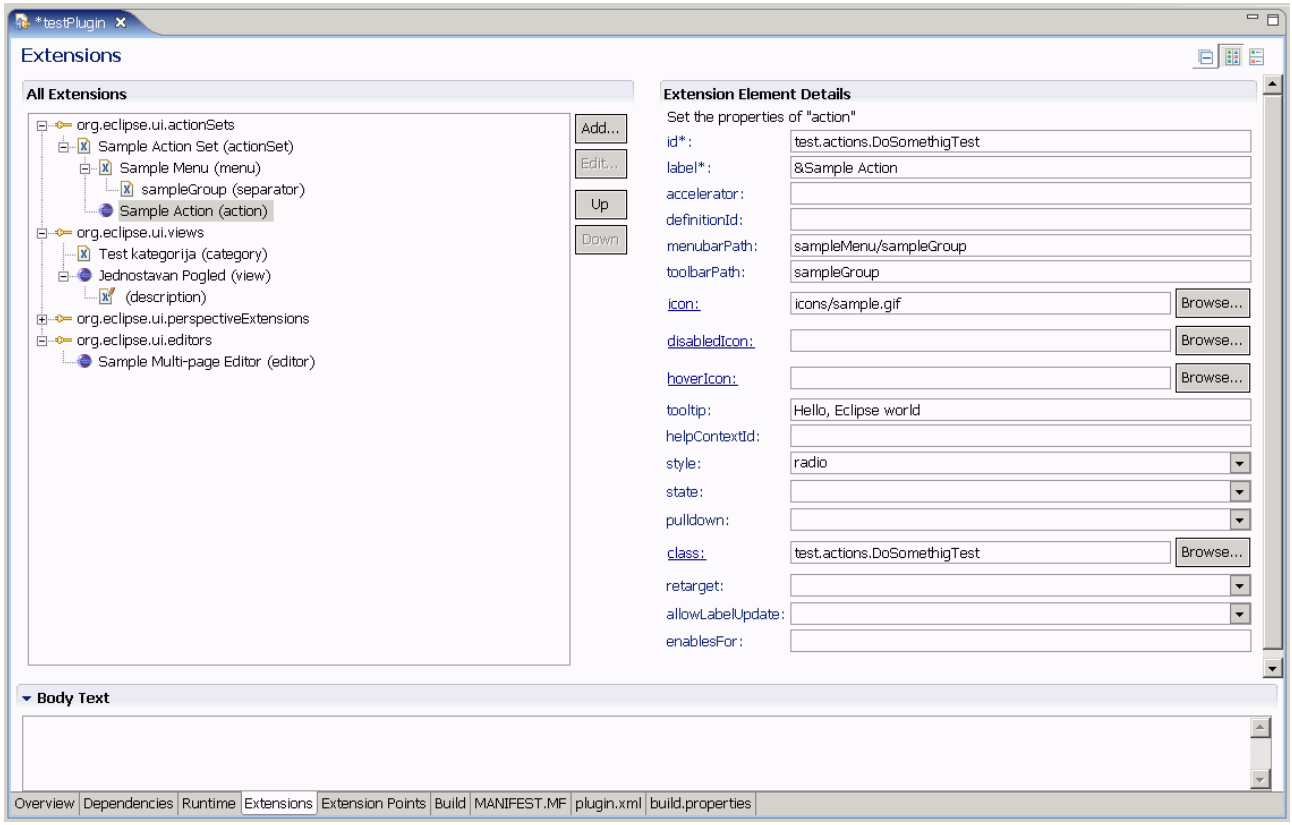
Slika 5.3: Dodavanje potrebnih biblioteka u classpath dodatka

Uređivač dodataka najkorisniji je dio manifest uređivača, on je zadužen za proširivanje točaka proširenja. U listu proširenja dodajemo proširenje, kod dodavanja otvara nam se izbornik u kojem možemo pretraživati listu proširenja i za svako dobijemo objašnjenje čemu to proširenje služi(Slika 5.4).



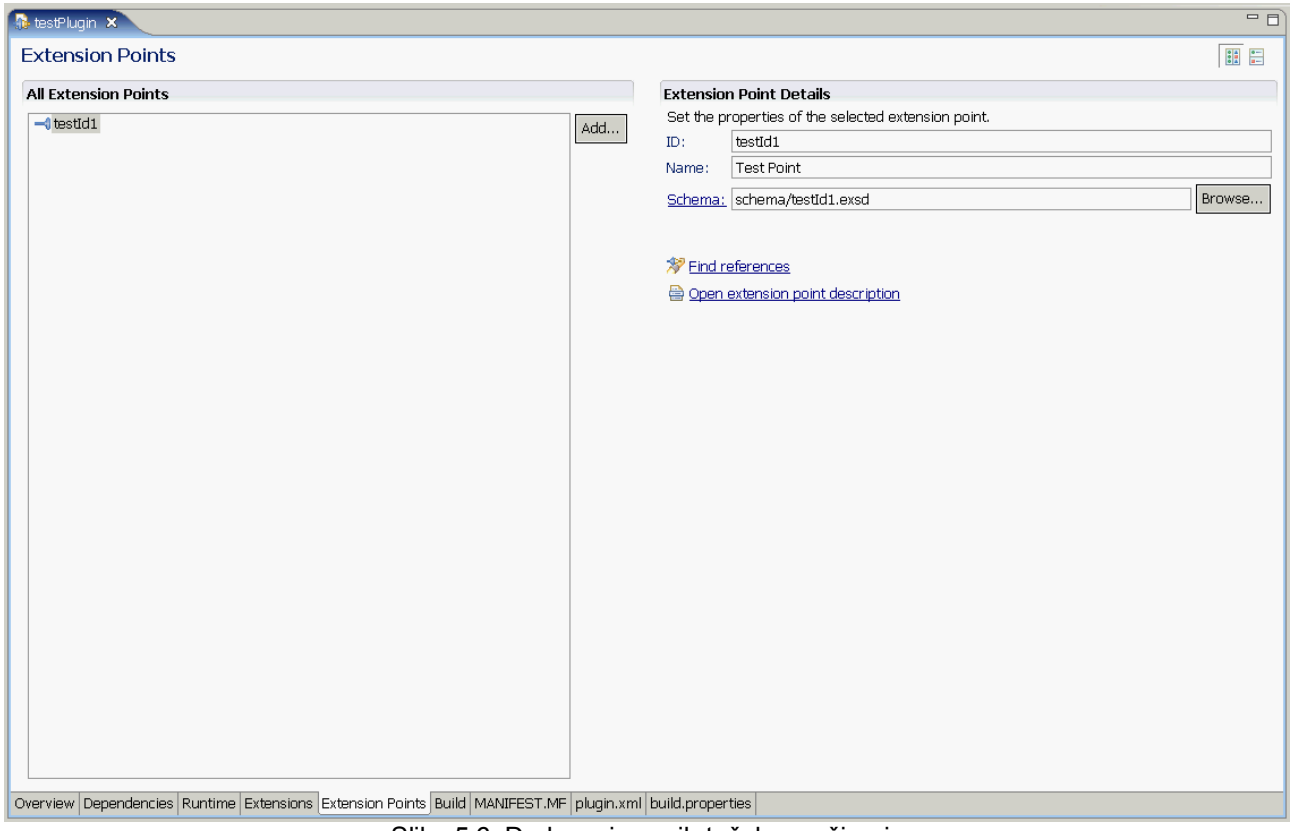
Slika 5.4: Pretraživanje popisa proširenja

Nakon što smo odabrali proširenje, ovisno o vrsti proširenja proširenju dodajemo potrebne parametre. Prednost PDE uređivača jest da programer vidi sve moguće parametre(Slika 5.5) te dobiva poveznice na Java izvorni kod koji je povezan na to proširenje. U slučaju da Java kod ne postoji otvara se dialog za generiranje novog Java razreda koji implementira ili nasljeđuje odgovarajuće Java sučelje ili razred.

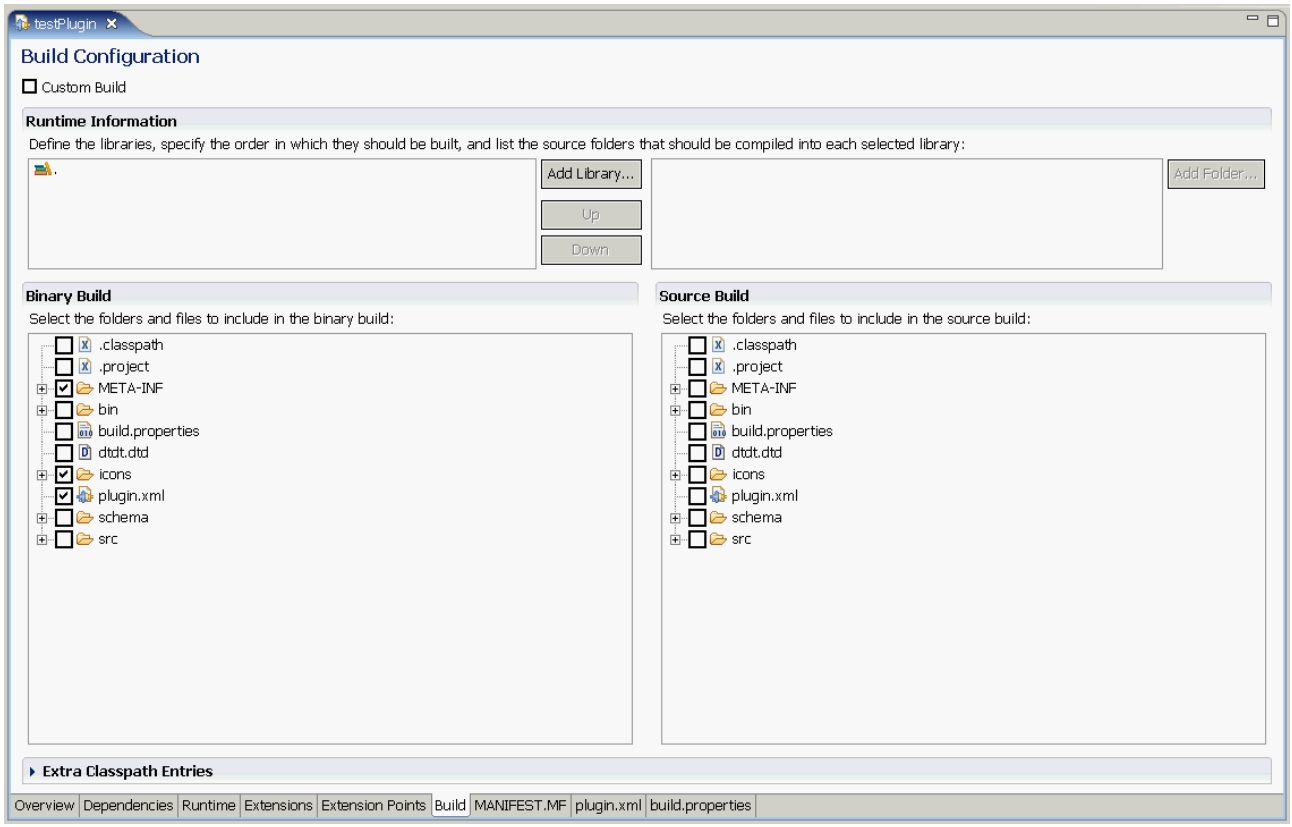


Slika 5.5: Obrada proširenja i parametara proširenja

Točke proširenja definiraju se slično proširenjima, no definiranje novih točaka proširenja zahtijeva pisanje sheme koju svi dodaci koji proširuju tu točku proširenja moraju slijediti da bi se ispravno registrirali u Eclipse okolinu. PDE nudi editor za izgradnju takve sheme. U slučaju da točke dodatak ne treba biti proširivan, ili se ne želi pružiti mogućnost proširivanja ovaj dio nije potreban.



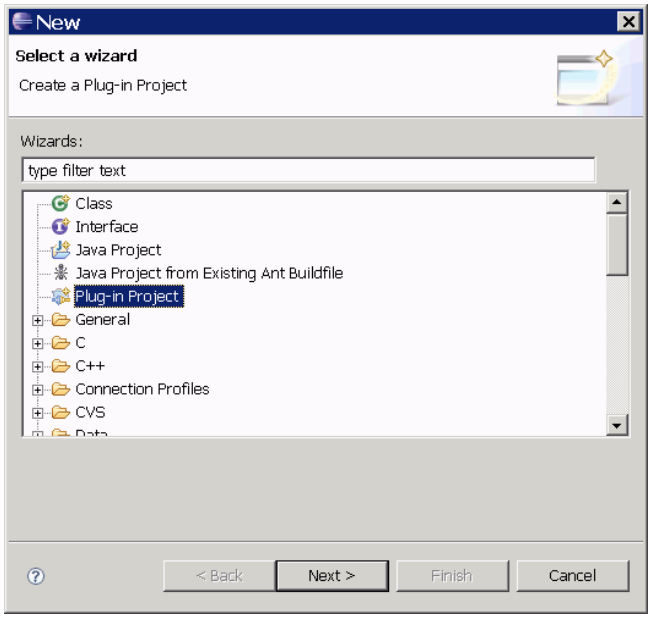
Slika 5.6: Dodavanje novih točaka proširenja



Slika 5.7: Odabir datoteka, biblioteka koje će biti integrirane u izgradnju dodatka

5.1. Kreiranje novog projekta za izradu dodatka

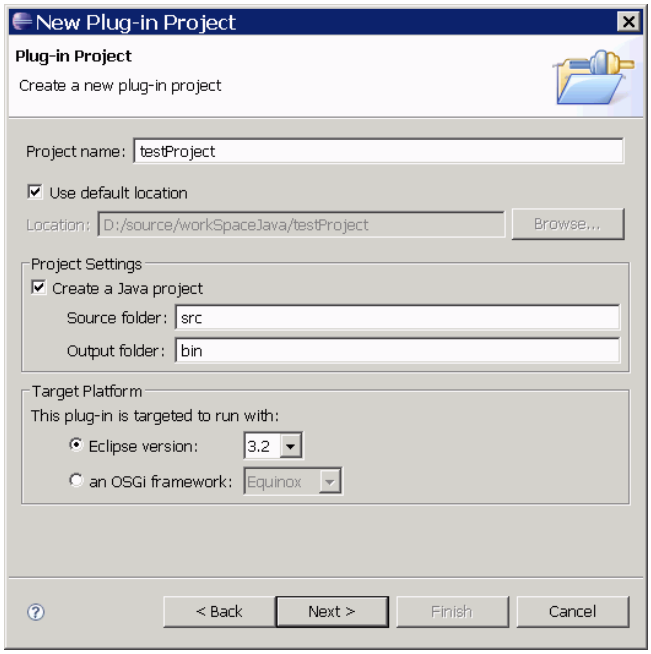
PDE sadržava čarobnjak koji pojednostavljuje izradu dodataka automatski generirajući preddefinirane datoteke plugin.xml, manifest.mf.



Slika 5.8: Čarobnjak New Plug-in Project

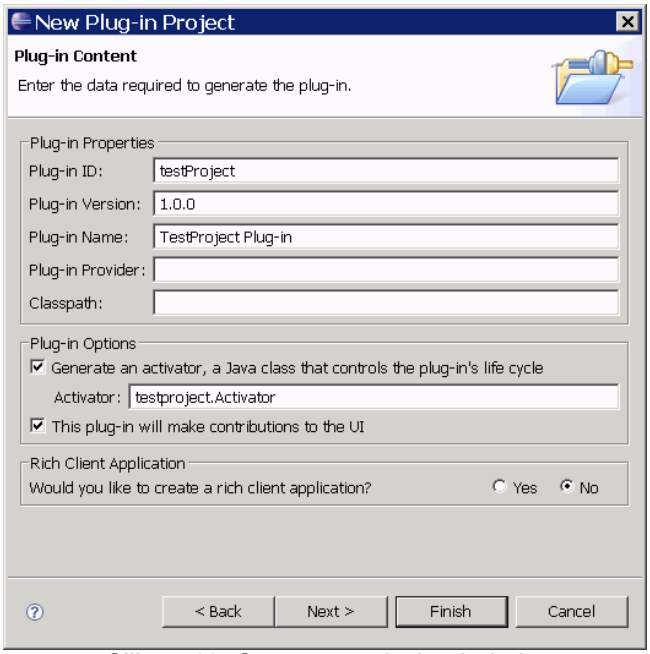
Za kreiranje novog plug-in projekta PDE nam nudi čarobnjak New Plug-in Project koji možemo naći pod kategorijom Plug-in Development / New Project (Slika 5.8).

Nakon odabira Plug-in Projecta postavlja se slijedeća na kojoj biramo mjesto gdje želimo smjestiti datoteke projekta, verziju Eclipse platforme za koju želimo namijeniti dodatak(Slika 5.9).



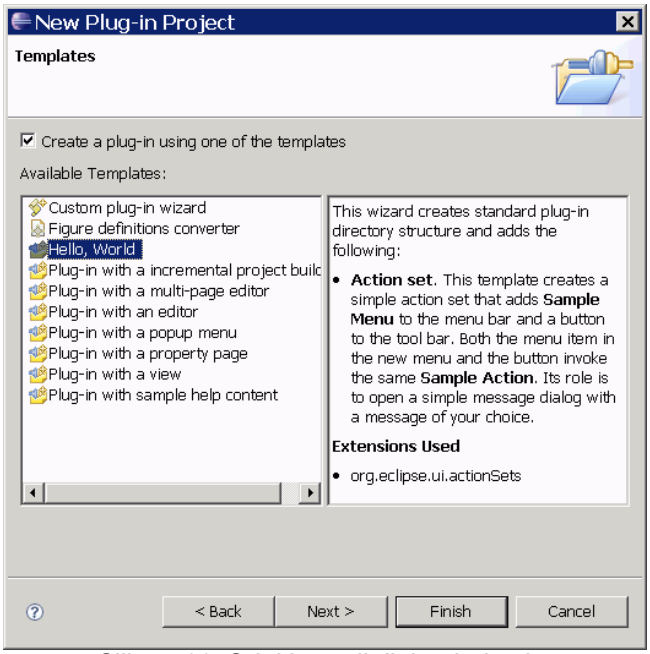
Slika 5.9: Odabir Eclipse platforme za novi dodatak

Slijedeći korak je odabir ID dodatka, verzije, imena proizvođača dodatka, ime osnovnog Java razreda dodatka. Informacije ovog dialoga rezultirati će osnovnim kosturom novog dodatak te će na temelju njega biti generirane datoteke plugin.xml i manifest.mf i osnovni Java razred koji služi za aktiviranje dodatka (Slika 5.10).



Slika 5.10: Osnovne podaci o dodatku

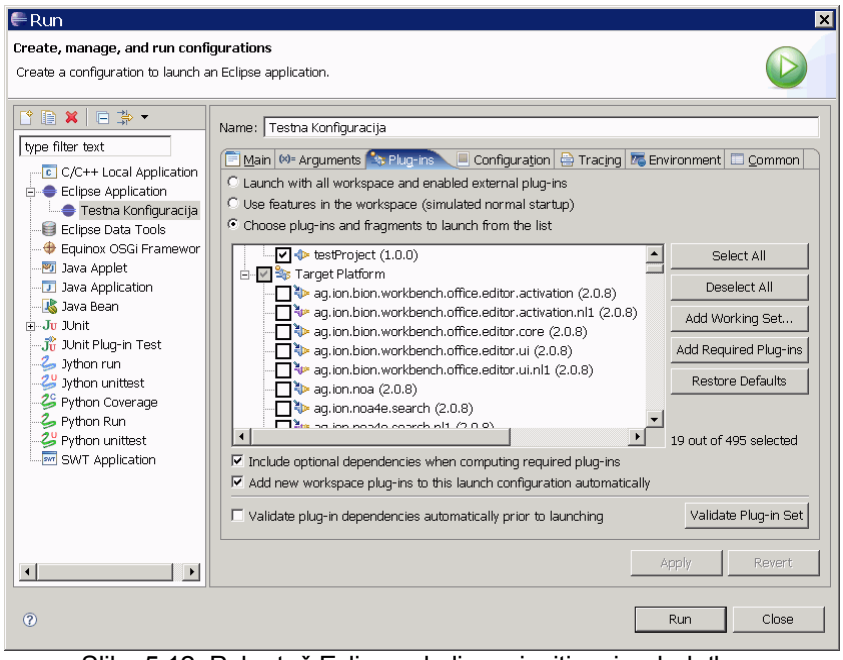
Nakon ovog koraka možemo završiti, ili odabrati neki od predložaka za brzi početak(Slika 5.11). Ovisno o odabiru aktivirati će se odgovarajući čarobnjak koje će generirati potrebana proširenja(XML elemente) i pripadajući Java kod.



Slika 5.11: Odabir predložaka dodataka

5.2. Ispitivanje i pokretanje dodataka

Da bi se pojednostavnilo ispitivanje dodataka PDE omogućuje pokretanje dodataka iz samog Eclipse razvojnog okruženja. PDE radi automatsku izgradnju dodataka, definiranje svih potrebnih parametara(JRE, VM argumenti…).



Slika 5.12: Pokretač Eclipse okoline s ispitivanim dodatkom

Čarobnjak za pokretanje omogućuje odabir dodataka koje želimo uključiti u Eclipse aplikaciju koju ćemo pokrenut. Prije pokretanja možemo provjeriti da li su svi potrebni dodaci uključeni. Pokretanje aktiviramo pritiskom na tipku *Run* nakon čega nam se pali novo Eclipse okruženje s našim dodatkom uključenim u to novo Eclipse okruženje, gdje možemo ispitati funkcionalnost našeg dodatka.

5.3. Izvoz dodataka u arhivu za isporuku

Zadnje korak izrade dodatka je pakiranje dodatka koji se može isporučiti i instalirati u Eclipse okruženje. Za sam proces pakiranja dodatka zadužene se `build.properties` konfiguracijske datoteke.

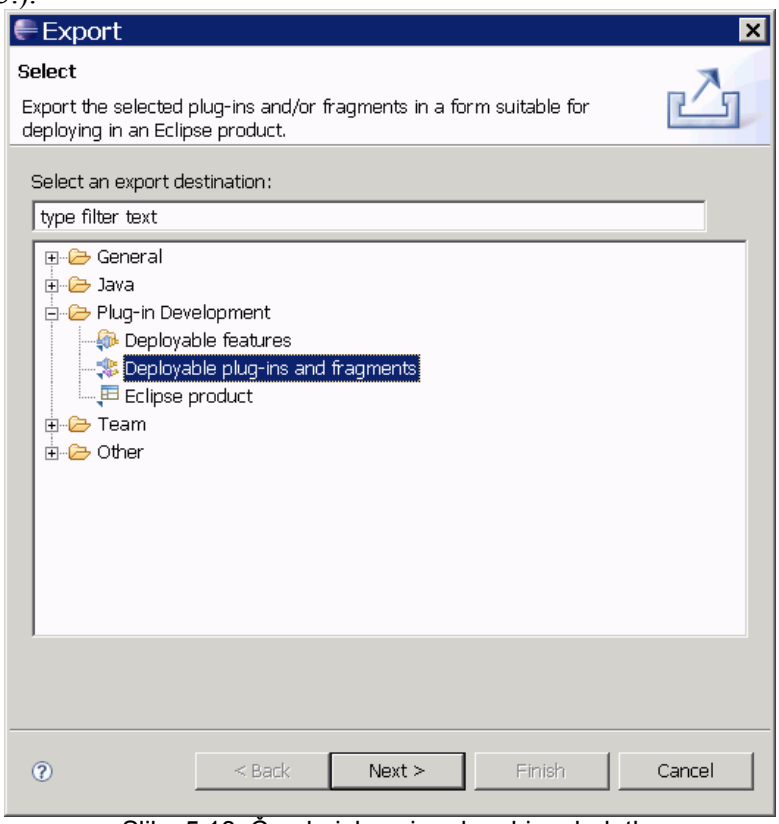
Ova je datoteka strukturirana kao skupe ključnih vrijednosti, te određuje koje je datoteke potrebno uključiti, a koje isključiti iz zapakiranoga dodatka. Umjesto ručnog uređivanja datoteke `build.properties` možemo iz manifest uređivača na Build stranici odabrati koje datoteke želimo isporučiti.

Primjer složenije datoteke `build.properties` dan je ispod:

```
source.. = src/
output.. = bin/
bin.includes = plugin.xml,\
               META-INF/,\
               .,\
               lib/hsqldb_1_8_0_7/hsqldb.jar,\
               lib/log4j-1.2.14.jar
jars.extra.classpath = lib/commons-net-1.4.1/commons-net-1.4.1.jar,\
                       lib/hsqldb_1_8_0_7/hsqldb.jar
```

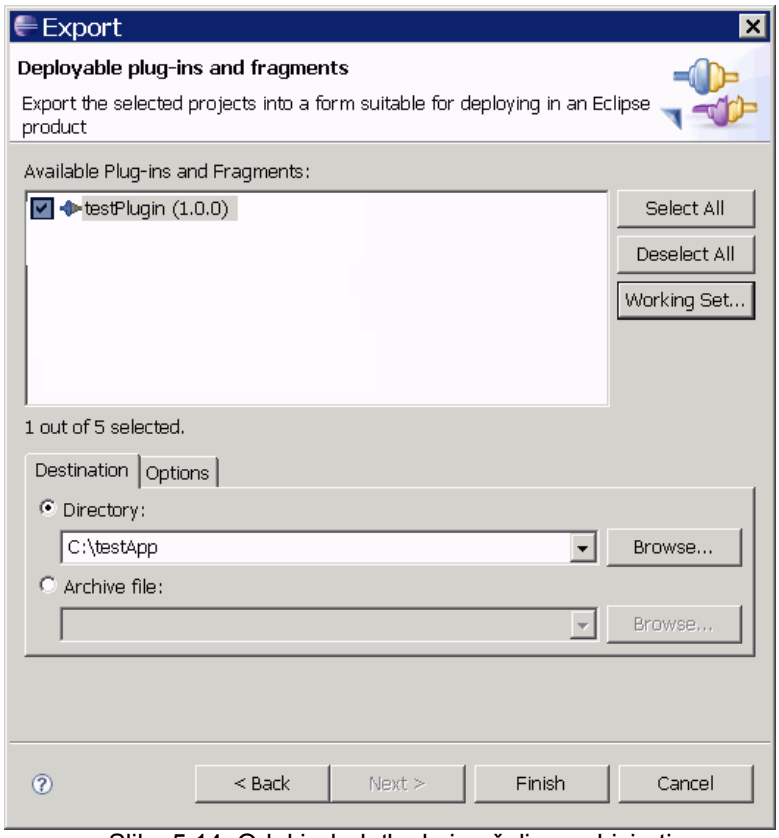
Primjer datotke build.properties

Za jednostavnije pakrianje i izradz arhive dodatka korisit se čarobnjak *Deployable plug-ins and fragments*(Slika 5.13.).



Slika 5.13: Čarobnjak za izradu arhive dodatka

Čarobnjak će nam ponuditi skup projekta koje možemo spremiti u arhivu, od kojih odaberemo onaj za koji želimo napraviti arhivu(Slika 5.14). Dodatak možemo spremiti kao direktorij ili komprimiranu arhivu te ovisno što želimo odaberemo putanju do direktorija ili arhive.



Slika 5.14: Odabir dodatka kojeg želimo arhivirati

Instalacija dodatka svodi se na kopiranje arhive ili direktorija u Eclipse pod direktorij `.\plugins`.

6. Zaključak

Eclipse platform je doista ono za što je i zamišljeno otvorena platforma za sve po malo i ništa određeno. Eclipse Platform predstavlja temelje za gradnju i rad integriranih alata za razvoj, ona daje slobodu proizvođačima alata da nezavisno razvijaju alate koje mogu kombinirati sa alatima drugih – i to toliko jednostavno da nitko ne može primijetiti gdje jedan alat završava, a drugi počinje. Svi dijelovi zajedno stvaraju bogato razvojno okruženje koje programeru omogućava učinkovito razvijanje alata koji se glatko uklapaju u Eclipse Platform.

Arhitektura Eclipse-a je skalabilna na broj dodataka, pomoću kojih svatko može pridonijeti svoj dio obogaćujući postojeće komponente ili dodajući novu funkcionalnost. Velik broj postojećih dodataka pruža mogućnost korištenja Eclipse-a kao integriranog razvojnog okruženja za većinu popularnih programskih jezika (Java,C/C++,Pyhton,Perl,PHP,Ruby).

Brz razvoj Eclipse okruženja možemo zahvaliti upravo proširivost i mogućnosti da naše dodatke netko drugi može proširiti i obogatiti njihovu funkcionalnost. S druge strane korisničko sučelje koje Eclipse pruža je unificirano, te skup različitih alata zapravo izgleda kao jedan univerzalni alata.

Obogaćivanje Eclipse platforme pojednostavnjeno je zahvaljujući PDE projektu koji se isto integrira u Eclipse okruženje, te se pomoću odličnog Eclipse integriranog okruženja mogu raditi dodaci za sam Eclipse.

Sve više se Eclipse platforma, osim za izradu integriranih razvojnih okruženja, koristi i kao Rich Client Platforma, kao okvir za izradu aplikacija, jer tada dobivamo aplikaciju s svom funkcionalnosti, mogućnostima proširenja Eclipse platforme, univerzalno korisničko sučelje, te razvoj aplikacije možemo koordinirati pomoću PDE-a.

7. Literatura

1. Eclipse 3.2 Documentation (HTML Help Center) : Platform Plug-in Developer Guide
2. E. Clayberg, D. Rubel: Eclipse: Building Commercial-Quality Plug-ins, Second Edition, Addison Wesley Professional, 2006.
3. Eclipse Platform – Technical Overview, URL:http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf (2007.)
4. K. Guclu: A First Look at Eclipse Plug-In Programming URL:http://www.developer.com/java/other/article.php/10936_3316241_1 (2007)
5. W. Melhem and D. Glozic, IBM Canada Ltd: PDE Does Plug-ins URL:http://www.eclipse.org/articles/Article-PDE-does-plugins/PDE-intro.html
6. The Java Developer's Guide to Eclipse, URL:http://www.jdg2e.com/ch08.architecture/doc/index.html(2007.)
7. Azad Bolour : Notes on the Eclipse Plug-in Architecture URL:http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html (2006.)
8. Peter Nehrer : Developing Eclipse Plug-ins URL:http://www.developer.com/lang/article.php/10924_3552481_1 (2007.)
9. Eclipse RCP: A Platform for Building Platforms URL;http://www.onjava.com/pub/a/onjava/2006/08/23/eclipse-rich-client-platform.html?page=1(2007.)
10. Eclipse Plugins Exposed, Part 1: A First Glimpse URL:http://www.onjava.com/pub/a/onjava/2005/02/09/eclipse.html?page=1(2007.)