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

DIPLOMSKI RAD br. 1673

# SUSTAV ZA UREĐIVANJE TESTOVA TEMELJEN NA RAZVOJNOM OKRUŽENJU ECLIPSE

Hrvoje Slaviček

Zagreb, rujan 2007.

## Sažetak

*U ovom diplomskom radu opisan je XML jezik za definiranje ispitne okoline i testova koji se izvode na Lusca Test Framework-u, okviru za automatizirano ispitivanje aplikacija. Opisivanje testova jezikom XML dosta je zahtjevno, pa izrada primjerenog grafičkog korisničkog sučelja značajno pojednostavnjuje njihovo definiranje. Prema specifikaciji XML jezika predloženo je i izgrađeno grafičko korisničko sučelje implementirano kao dodatak razvojnom okruženju Eclipse. U korisničko sučelje ugrađen je podsustav za bilježenje aktivnosti korisnika, koji na osnovu zapisa dobivenih tijekom korištenja sučelja daje mjeru efikasnosti korištenja sučelja. Opisana je implementacija i sve korištene tehnologije.*

## Abstract

*This diploma thesis describes language for defining tests and test environment, which are executed by Lusca Test Framework, framework for automated testing of applications. Test definition with XML language is difficult task, and graphical user interface would simplify this task. According to XML language, appropriate graphical user interface is proposed and implemented as Eclipse plugin. Subsystem for logging of user activity is embedded in graphical user interface. Based on data collected during usage of user interface, this subsystem gives a measure of usage efficiency. Used technologies and implementation are also described in this work.*

# Sadržaj

# 1. Uvod

Mrežne aplikacije postaju programski složenije, a broja njihovih korisnika raste, što postavlja veće zahtjeve na pouzdanost mrežnih aplikacija. Većina aplikacija koristi se na javnoj globalnoj mreži Internet i kao takva izložena je velikom broju potencijalnih napada. Zbog toga su propusti u mrežnim aplikacijama kritični i nedopustivi. Kako bi se izbjegli propusti važno je mrežne aplikacije temeljito ispitati. Postupak ispitivanja pokazao se jednako složen i vremenski zahtjevan kao i sam razvoj aplikacije.

Ispitivanje mrežne aplikacije obično zahtjeva više računala, mrežnih konfiguracija, ispitivanu aplikaciju i razne dodatne datoteke koje je potrebno distribuirati na različite lokacije, dok je pokretanje aplikacije potrebno vršiti na više računala. Ovakav postupak zahtjeva dosta vremena, koje se može posvetiti razvoju ili detaljnijem ispitivanju. Nepostojanje sustava koji bi omogućio univerzalnu automatizaciju ovog problema potaknuo je razvoj Lusce, sustava za ispitivanje mrežnih aplikacija.

Osnova Lusce je jezik kojim se definira opis testova. Jezik je ostvaren kao XML jezik i pruža veliku ekspresivnost u opisu ispitnog okruženja i ispitnih slučajeva. Opisivanje testova jezikom XML dosta je zahtjevan postupak. Osoba koja ga piše mora dobro poznavati gramatiku jezika, koja je definirana XML shemom. Cilj diplomskog rada je predložiti i implementirati grafičko korisničko sučelje koje korisniku na intuitivan način prikazuje ispitnu deklaraciju testa, olakšava unos testova i ne zahtjeva od korisnika poznavanje gramatike Lusca ispitne deklaracije, već samo način na koji se izvode testovi u Lusca sustavu.

Za izgradnju grafičkog korisničkog sučelja odabrana je Eclipse Platforma, zbog otvorenog koda, proširivost, velikog broja postojećih alata i njihove međusobne integracije. Eclipse se također nameće kao vodeća tehnologija zbog odlične podrške zajednice i velikih kompanija.

Nadogradnja Eclipse platforme omogućit će izradu okruženja za definiranje testova preko grafičkog korisničkog sučelja, integraciju s dodatkom koji omogućuje upravljanje Lusca ispitnom okruženju i ostalim Eclipse alatima.

Diplomski rad podijeljen je u više poglavlja. Drugo poglavlje opisuje Lusca okvir i definiranje ispitnih paketa. U trećem poglavlju ukratko su prikazane korištene tehnologije pri implementaciji Lusca korisničkog sučelja za izradu ispitnih paketa. Eclipse platforma opisana je u četvrtom poglavlju. Implementacija Lusca korisničkog sučelja za izradu ispitnih paketa opisana je u petom poglavlju, a korištenje izrađenog sučelja prikazano je u šestom. Zaključci diplomskog rada navedeni su u sedmom poglavlju.

# 2. Lusca

Pri razvoju mrežnih aplikacija javljaju se mnogi problemi. Jedan od glavnih problema za programera mrežne aplikacije je ispitivanje i traženje grešaka u takvim aplikacija. Glavni razlog ovog problema je način ispitivanja aplikacije koji zahtjeva posebno prilagođavanje ispitne okoline i dodatne programe kojima se vrši ispitivanje. Zbog prirode mrežnih aplikacija, koja je najčešće klijent-poslužitelj, za njihovo ispitivanje potrebno je imati više aplikacija. Za ispitivanje poslužitelja potrebno je imati i klijent aplikaciju, koju je pri ispitivanju poslužitelja, isto potrebno pokrenuti. Nakon toga potrebno je provjeriti rezultate koje su generirali poslužitelj i klijent, a zatim i mrežni promet između klijenta i poslužitelja. Često izvođenje ovog procesa, pri svakom ispitivanju uvedenih promjena, zahtjeva mnogo vremena koje programer gubi, a može bolje iskoristiti za razvoj aplikacije.

Lusca okvir za ispitivanje rješava taj problem. Lusca je zadužena za pripremu sklopovske i programske podrške, dohvaćanje aplikacija s različitih izvora u izvršnom ili izvornom kodu. Izvor može biti i neki od sustava za nadzor izvornog koda (engl. *Source Control Management – SCM*), trenutna implementacija podržava Subversion. Kod dohvaćanja aplikacije iz izvornog koda aplikacija se prevodi u binarnu datoteku i postavlja na odgovarajuće računalo. Osim aplikacija, Lusca može distribuirati datoteke na više računala. Lusca nakon postavljanja sklopovske opreme, datoteka i aplikacija pokreće aplikacije i vrši nadzor nad njima, a u odgovarajućim trenucima daje potrebne ulaze, reagira na izlaze koje aplikacije generiraju i na temelju njih određuje koju slijedeću akciju mora provesti. Nakon završetka izvođenja Lusca skuplja rezultate, koje su generirale pojedine aplikacije, sprema ih na centralno mjesto tako da programer na jednostavan način može doći do bilo kojeg rezultata. Nakon završetka ispitivanja Lusca čisti ispitnu okolinu od svih dostavljenih aplikacija i privremenih datoteka koje su generirane tijekom izvođenja testa kako bi, nakon izvođenja, stanje ispitne okoline bilo jednako stanju ispitne okoline prije izvođenja testova.

Korištenjem sustava za nadzor izvornog koda, kao izvor za dohvat aplikacije, omogućuje Lusca sustavu kontinuirano ispitivanje (engl. *regression testing*) aplikacije te obavještavanje programera koja je promjena izvornog koda prouzročila grešku i onemogućila uspješno izvršavanje testova.

Razvojni ciklus aplikacija prema unificiranom razvojnom procesu (engl. *Unified Process*) svodi se na slijedeće faze:

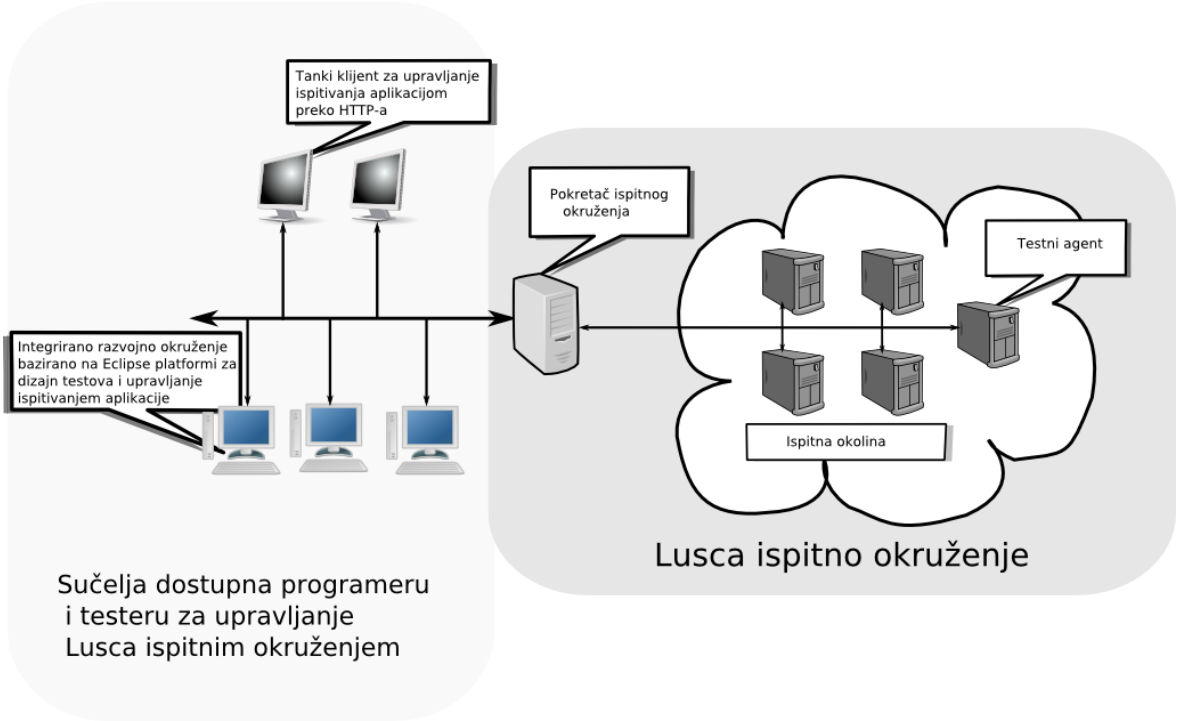
- određivanje zahtjeva aplikacije (engl. *Requirements*)
- analiza i dizajn sustava (engl. *Analysis and design*)
- implementacija (engl. *Implementation*)
- ispitivanje (engl. *Testing*)
- isporuka i postavljanje aplikacije (engl. *Deployment*)

Lusca se uključuje u ovaj iterativni ciklus u fazama implementacije i ispitivanja. Osoba koja vrši ispitivanje će na temelju zahtjeva aplikacije napisati Lusca testove i omogućiti programeru da ispituje aplikaciju u svakoj fazi razvoja. Programer pomoću Lusca ispitnog okvira može ispitivati aplikaciju. Lusca mu pruža informacije o napretku u razvoju, greškama koje su se možda pojavile između dvije faze razvoja, uzroku grešaka, a može

dobiti i stanje cijelog mrežnog sustava u kojem se pojavila greška. Programer pri ispravljanju greške ne mora gubiti vrijeme na ostvarivanje uvjeta koji su prouzročili grešku već može samo pokrenuti test koji ju je uzrokovao.

## 2.1. Arhitektura Lusca okruženja za ispitivanje mrežnih aplikacija

Kako bi se moglo služiti Lusca sustavom potrebno je poznavati njegovu arhitekturu. Lusca sustav dijelimo na dva dijela: sučelja prema korisniku i ispitno okruženje. Slika 2.1. prikazuje ta dva dijela i njihovu internu arhitekturu.




Slika 2.1. Arhitektura Lusca sustava za ispitivanje mrežnih aplikacija

### 2.1.1. Lusca sučelja prema korisniku

Lusca sučelje prema korisniku služi za upravljanje i nadzor Lusca ispitnog okruženja, te pisanje Lusca ispitnih paketa. Lusca omogućuje dva pristupa za upravljanje sustavom. Oni se razlikuju po svojim zahtjevima i mogućnostima.

Prvi način upravljanja je preko web sučelja za što je dovoljan web preglednik. Ovaj način omogućuje jednostavan i brz pristup Lusca ispitnom okruženju bez potrebe za instalacijom dodatne programske podrške. Moguće je pokrenuti, zaustaviti sustav, pokrenuti izvođenje svih ili samo određenog testa, pregledati rezultate testova, poslati novi ispitni paket na izvršavanje. Ovo sučelje ne omogućuje izradu Lusca ispitnih paketa.

Drugi način upravljanja je iz integriranog razvojnog okruženja (engl. *integrated development environment*). Za izradu integriranog razvojnog okruženja odabran je Eclipse platforma zbog velikog broja već postojećih dodataka (engl. *plugin*) različitih namjena koji

omogućuju uporabu različitih programskih jezika, različitih vanjskih alata (npr. sustav za upravljanje izvornim kodom), a omogućuju korištenje različitih programerskih metodologija, dok opet omogućuje jednostavno integraciju novih dodataka u jedinstvenu cjelinu. Iz Eclipse integriranog razvojnog okruženja moguće je raditi sve poslove koji se mogu raditi iz web sučelja i izrađivati Lusca ispitne pakete. Eclipse platforma omogućuje integraciju bilo kojeg Eclipse baziranog dodatka s grafičkim sučeljem za izradu Lusca testova, što se koristi kod integracije s alatima za pisanje skripti, koje se koriste u ispitnim paketima. Integracija omogućuje da se, pri razvoju aplikacije koja se ispituje, direktno radi i s Lusca ispitnim okruženjem, kako bi se nakon izvršenih promjena nad izvornim kodom moglo pokrenuti ispitivanje aplikacije s tim promjenama, prije nego li se promjene potvrde sustavu za kontrolu izvornog koda kao nova revizija. Korištenje integriranog razvojnog okruženja zahtjeva prethodnu instalaciju Eclipse SDK i Java JRE programskih paketa, što web sučelje ne zahtjeva.

### 2.1.2. Lusca ispitno okruženje

Sustav koji izvodi testove je Lusca ispitno okruženje, sastoji se od dva dijela: pokretača ispitnog okruženja (engl. *test engine*) i od jednog ili više Lusca test agenata. Test agenti i računala na kojima se nalaze čine ispitnu okolinu.

Pokretač ispitnog okruženja zadužen je da na temelju podataka, koji su mu dostupni iz njegovih konfiguracijskih datoteka, zaključi koja sredstva ima na raspolaganju (stvarna računala i njihova svojstva, mrežna oprema, mogućnosti generiranja virtualnih računala), generira sva potrebna sredstva koja su potrebna za uspješno izvršavanje testova. U slučaju da za određeni test nema dovoljno potrebnih sredstava koja su potrebna za uspješno izvođenje testova sustav šalje poruku putem elektroničke pošte osobi koja je izradila test. Nakon uspješnog postavljanja ispitne okoline na računala se smještaju test agenti. Tada može krenuti izvođenje testova. Pokretač ispitnog okruženja komunicira s test agentima i daje im naredbe koje oni izvršavaju.

Test agenti primaju naredbe od pokretača ispitnog okruženja. Naredbe mogu biti zahtjev za pokretanjem ispitne aplikacije, slanje stanja računala na kojem se nalaze, slanje stanja ispitivane aplikacije, slanje rezultata, postavljanje aplikacija i datoteka na računalo test agenta, ili bilo kakvo drugo upravljanje računalom na kojem se nalazi test agent.

Ispitnu okolinu preporuča se postaviti na računala na izoliranom mrežnom dijelu zbog sigurnosnih razloga, a svaki test agent na posebno stvarno računalo. Ako to sredstva ne dozvoljavaju, za izvođenje je dovoljno i samo jedno računalo na kojem se može staviti i pokretač ispitne okoline i više test agenta. Druga mogućnost je na jednom stvarnom računalu napraviti više virtualnih računala pomoću programske podrške za virtualizaciju i na taj način napraviti ispitnu okolinu. Lusca je prilagodljiva s obzirom na računalna i mrežna sredstva koja su dostupna.

## 2.2. Lusca ispitni paket

Lusca ispitni paket je skup datoteka zapakiranih u komprimiranoj arhivu s ekstenzijom datoteke ltp. Zadaća jednog ispitnog paketa je sadržavati opis ispitne okoline (potrebna računala i njihove postavke), stanje ispitne okoline (potrebne aplikacije, datoteke, konfiguracije aplikacija), verzija aplikacija koje se ispituju, testova i njima pripadajućih

skripti koje se mogu izvoditi na tako pripremljenoj okolini. Nakon što pokretač primi ispitni paket on će prema specifikacijama ispitnog paketa pripremiti ispitnu okolinu i na njoj izvoditi zadane testove. Nakon što se izvedu svi testovi sustav vraća ispitnu okolinu u prvobitno stanje.

Tipična struktura otpakiranog Lusca ispitnog paketa je:

```
scripts
confs
doctest_package.ltd
tfsch.xsd
.config
```

U Lusca ispitnom paketu obavezno se moraju naći datoteke *ime.ltd*, *tfsch.xsd*, *.confg*, a direktoriji *scripts*, *confs*, *doc* su opcionalni, a se koriste za smještanje skripta, dokumentacije i konfiguracijskih datoteka.

Datoteka *ime.ltd* (gdje je ime proizvoljno ime datoteke) sadrži Lusca ispitnu deklaraciju u XML-u kojom se opisuju ispitni slučajevi i svi potrebni elementi kako bi se mogli izvoditi.

Datoteka *tfsch.xsd* sadrži XML Schemu (vidi Dodatak B) koja definira valjani XML dokument smiješten u datoteci *ime.ltd*, dok je u datoteci *.config* zapisano ime datoteke koja sadrži Lusca ispitnu deklaraciju tj. *ime.ltd*.

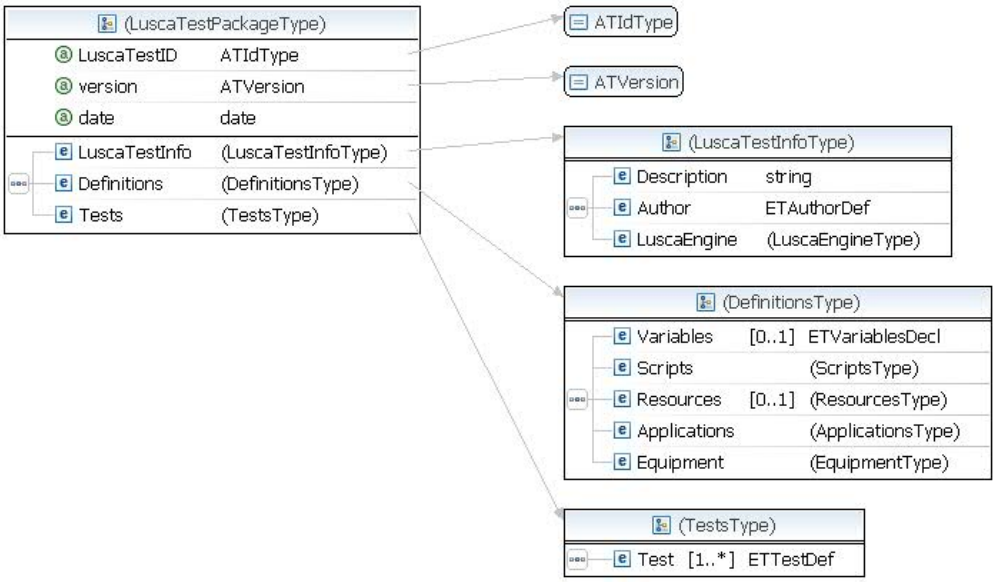
Zadaća Lusca ispitnog paket je skupiti skripte, sve potrebne datoteke na jedno mjesto i pomoću ispitne deklaracije opisati njihovo značenje kako bi Lusca sustav mogao na ispravni način prirediti okolinu za ispitivanje.

## 2.3. Elementi Lusca ispitne deklaracija

Osnovni dio Lusca ispitivanja je Lusca ispitne deklaracija, to je XML dokument opisan XML Shemom (Dodatak B). Detaljan opis svakog pojedinog elementa naveden je u literaturi [2].

Lusca ispitna deklaracija je XML dokument podijeljen u tri dijela kako je prikazano slikom 2.2.:

- informacije o ispitnom paketu
- definicije (varijabli, računala, aplikacija, skripata, datoteka ili direktorija)
- popis ispitnih slučaja

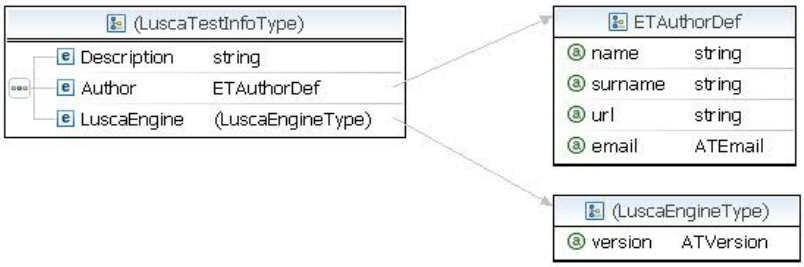



Slika 2.2. Prikaz osnovnih elemenata XML-a koji opisuje Lusca test deklaracije

Ispitna deklaracija sadrži još i podatke kad je napravljena ispitna deklaracija, koja je verzija i jedinstveni identifikator samog ispitnog paketa.

## 2.3.1. Informacije o ispitnom paketu

U XML elementu `LuscaTestInfo` nalaze se informacije o autoru ispitnog paketa, kao što su opis ispitnog paketa, podaci o autoru i verzija Lusca ispitnog okruženja za koje je ovaj test namijenjen. Podaci o autoru koriste se u slučaju greške kako bi osoba, odgovorna za ispitni paket, mogla biti obaviještena elektroničkom poštom. XML s informacijama koje opisuju Lusca ispitni paket prikazan je slikom 2.3.




Slika 2.3. Informacije o Lusca ispitnom paketu

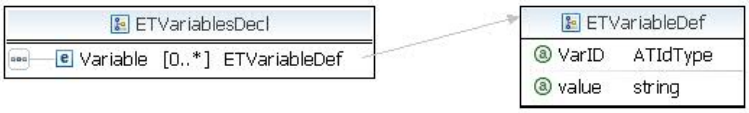
### 2.3.2. Definicije Lusca ispitne deklaracije

U definiciji ispitne deklaracije opisuju se elementi koji su potrebni za opisivanje ispitnog okruženja i ispitnih slučaja. Identifikatori elementa opisanih u definicijama kasnije se refernciraju u ispitnim slučajevima.. Elementi koji se ovdje opisuju su:

- varijable
- skripte
- resursi (datoteke i direktoriji)
- aplikacije
- računalna oprema

Varijable služe kao pomoćni spremnici vrijednosti u sustavu ispitnih slučaja ili kao parametri za prijenos podataka između skripti, ulazni parametri skripta, ili primaju rezultat koji vraća neka skripta.

Varijable se definiraju svojim identifikatorom tj. imenom (`VarID`) i početnom vrijednošću (`value`), definicija varijabli u XML prikazana je slikom 2.4.



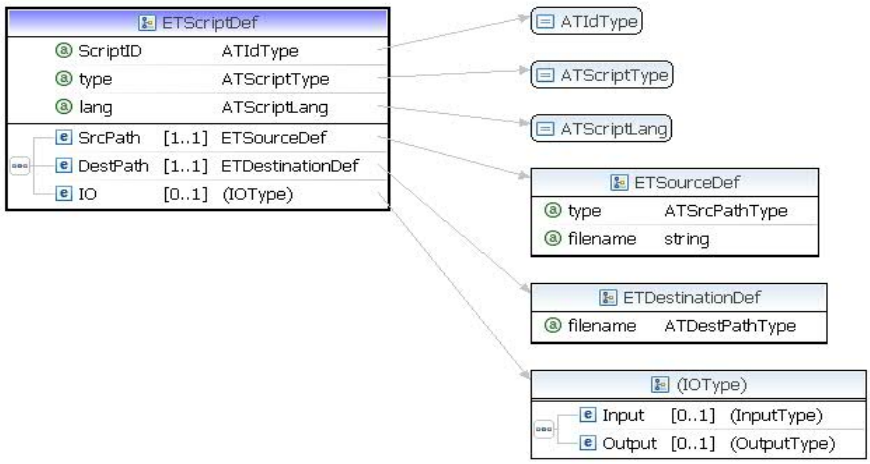

Slika 2.4. Prikaz definicije varijable

Varijabli se može definirati niti jedna ili neograničeno mnogo. Kako bi varijabla imala smisla potrebno ju je povezati s nekom skriptom, povezivanje se vrši pri izvršavanju skripti. Varijable omogućuju postizanje veće općenitosti ispitnih slučaja, jer se pomoću njih mogu definirati parametri koji se mogu mijenjati tijekom ciklusa ispitivanja.

Skripte vrše stvarni posao ispitivanja, one su izvršitelj svih akcija. Mogu biti napisane u nekom od podržanih skriptnih programskih jezika (`sh`, `python`, `perl`). Slika 2.5. prikazuje sve potrebne XML elemente i svojstva kako bi se definirala skripta. Kako bi ispitna deklaracija bila valjana mora biti definirana barem jedna skripta, jer se u protivnom nema što izvoditi, dok je broj skripti koje se mogu definirati neograničen.

Skripte se moraju prijaviti Lusca sustavu, a prijavljuju se u dijelu definicija, gdje je potrebno svakoj odrediti jedinstveni identifikator (`ScriptID`) i tip akcije koji vrši skripta (type). Tip akcije može biti računanje (`calculate`), parsiranje podataka (`parse`), ili izvršavanje (`execution`). Pomoću identifikatora na skripte se referencira kasnije kod izvođenja, a tip govori Lusca sustavu na koji način treba izvesti skriptu i kakvo ponašanje očekivati od nje.

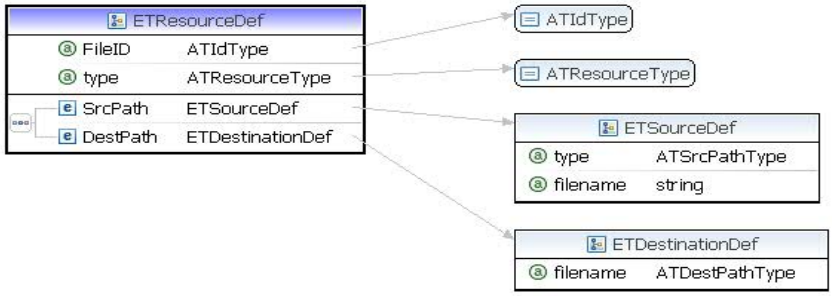
Potrebno je definirati mjesto gdje se nalazi skripta (`local`, `http`, `ftp`), putanju do nje i lokaciju gdje će se kopirati na računalo na koje će biti distribuirano.




Slika 2.5. Podaci o skripti

Skripte mogu imati ulaze i izlaze koje se koriste za komunikaciju s drugom skriptom ili kao parametrizaciju skripte. Ulazi i izlazi u `IOType` dijelu kao ulaz (`input`) ili izlaz (`output`). Svaki ulaz/ izlaz određen je svojim identifikatorom (`ParID`, `RetID`) pomoću kojeg se kasnije referencira i indeksom koji označava koji je to parametar u skripti.

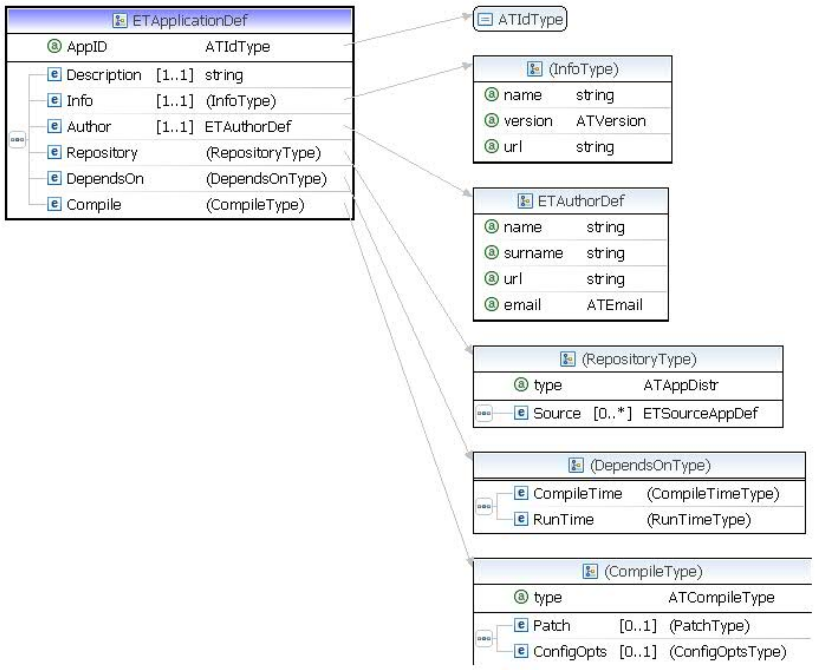
Osim skripti kao dinamičkih dijelova moguće je imati i statičke dijelove, resurse koji mogu biti datoteke i direktorij. Resurs kao i sve ostalo u definicijama mora imati svoj identifikator, tip (`type`) koji određuje da li se radi o datoteci (`file`) ili direktoriju (`directory`), gdje se nalazi i gdje se treba smjestiti na računalo agenta. Potrebni elementi pri definiranju resursa prikazani su slikom 2.6.




Slika 2.6. Opis resursa

Potrebno je definirati aplikacije koje će se ispitivati. Kako bi se definirala aplikacija potrebno je definirati njen identifikator (`AppID`), opis aplikacije (`Description`). Unose se podaci o aplikaciji kao što su ime aplikacije, verzija i njezinu web adresu. Podaci o

autoru, s njegovom adresom elektroničke poste adresom, navode se kako bi ga Lusca mogla obavijestiti u slučaju pogreške. Potrebni podaci o aplikaciji prikazani su slikom 2.7.




Slika 2.7. Prikaz svi potrebnih elementa za definiranje aplikacije

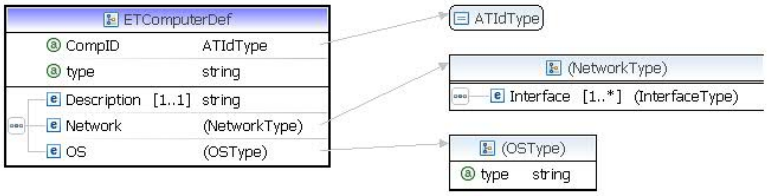
Kako bi aplikacija bila potpuno definirala potrebno je definirati još tri dodatne stvari: repozitori, njene ovisnosti i tip prevođenja kao što je i vidljivo iz slike 2.7.

Repozitori definira način dohvata aplikacije i tip distribucije aplikacije na agente koja se može distribuirati kao izvorni kod (`source`) ili izvršna datoteka (`binary`), za repozitorij je potrebno definirati i izvorište (kojih može biti više), a izvorište određuje tip izvorišta (`http`, `ftp`, `subversion`, `local`), broj revizije aplikacije i putanju na izvoru.

Ovisnosti o drugim programskim paketima definiraju listu biblioteka koje moraju postojati tijekom prevođenja aplikacije (`CompileTime`) i tijekom izvršavanja aplikacije (`RunTime`).

Tip prevođenja određuje kada će se aplikacija prevesti. Prevođenja se može izvesti prije distribucije na računala domaćine (`beforeDistribution`), tada se prevodi na pokretaču ispitnog okruženja i nakon distiribucije (`afterDistribution`), tada se izvodi na računalu domaćinu Kod prevođenja moguće je dodati dodatne parametre koje se predaju prevoditelju (`ConfigOpts`).

Računalna oprema sastoji se od računala i njihovih mrežnih sučelja. Ovdje se definira ime računala (`CompID`) na kojeg kasnije može referencirati kao mjesto na koje će se isporučiti aplikacija, neki od resursa ili skripta, isto tako se definira računalo izvoditelj skripta. Računala su domaćini ispitivanih aplikacija. Kako bi ispitna deklaracija bila valjana mora postojati barem jedno računalo, inače se testovi nemaju gdje izvoditi. Svi potrebni elementi kako bi se definiralo jedno računalo prikazani su slikom 2.8.




Slika 2.8. Definicija računala

Za računalo se definira identifikator računala (`CompID`) i tip računala (`type`). Kako bi se odredila svrha računala ono se kratko opisuje (`Description`) i računalu se definiraju mrežna sučelja (`Network`). Mrežno sučelje (`Interface`) sadrži informacije (`address`, `netmask`, `gateway`) potrebne za aktiviranje mrežnog sučelja na računalu čije ime odgovara standardnim nazivima koji se koriste prilikom aktivacije mrežnog sučelja.

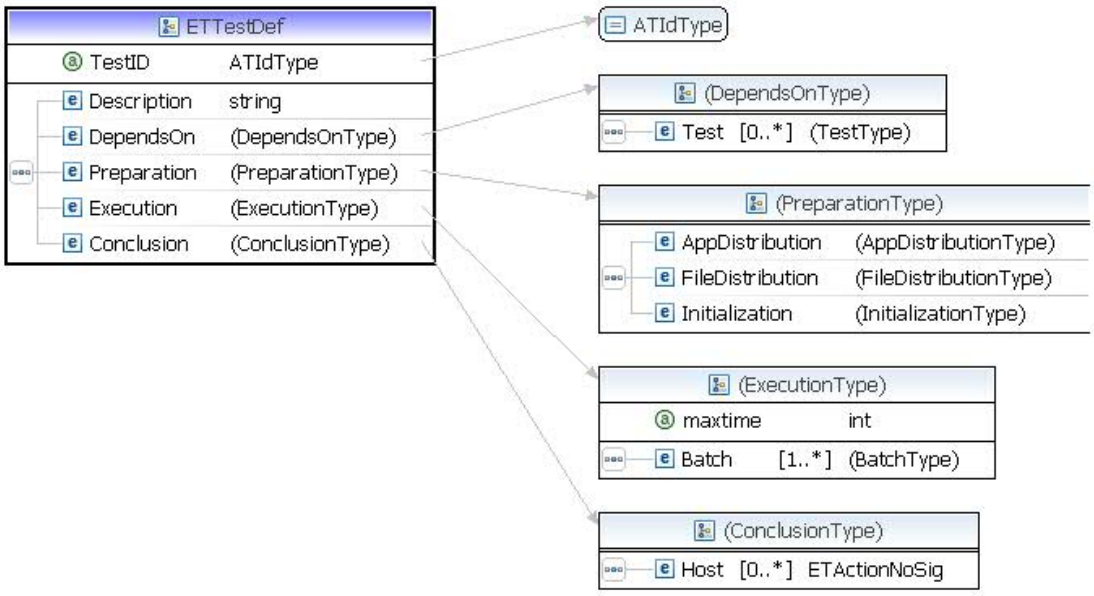
Moguće je definirati operacijski sustav (`OS`), no trenutna implementacija ograničena je na GNU/Linux operacijski sustav pa deklaracija operacijskog sustava nema nikakav utjecaj.

### 2.3.3. Ispitni slučajevi Lusca ispitne deklaracija

`Tests` element navodi više elemenata `test`, od kojih svaki predstavlja jednu cjelinu, ispitni slučaj (engl. *test case*) aplikacije.

Za pojedini ispitni slučaj definira se njegov identifikator (`TestID`), opis ispitivane funkcionalnosti (`Description`) i lista identifikatora o kojima zavisi izvođenje ovog testa (`DependsOn`). Prilikom izvođenja cjelokupnog testa gradi se hijerarhija ispitnih podslučaja koje treba izvršiti prije ovog zadanog ispitnog slučaja. Ukoliko se neki od ispitnih slučaja o kojima zavisi ne uspije ispravno izvršiti, tada se ne izvodi taj ispitni slučaj, a korisniku se dojavljuje status.

Ispitni slučaj se izvodi u tri faze: priprema, izvršavanje, završetak, slika 2.9. Ovisno o fazi u kojoj se ispitni slučaj nalazi vrše se pripadajuće radnje. Kako bi se pojedina radnja uspješno izvršila potrebno ju je definirati u `Test` elementu XML-a. Svi potrebni XML elementi za definiranje jednog ispitnog slučaja prikazani su slikom 2.9.

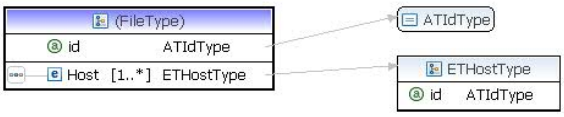



Slika 2.9. Struktura jednog ispitnog slučaja

Priprema ispitnog slučaja dijeli se u tri faze:

- distribucija datoteka i direktorija
- distribucija aplikacija
- inicijalizacija

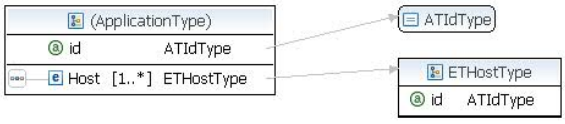
Svaki dio pripreme služi za različitu svrhu i svi se referenciraju na identifikatore unesene u definicijama. Za distribucija datoteka i direktorija potrebno je odabrati identifikator (*id*) jednog od resursa opisanoga u definicijama i listu *Host* elemenata s identifikatorima računala na koji će se datoteke ili direktoriji kopirati (slika 2.10.). Ovaj način omogućava distribuciju jedne datoteke na više računala, što može biti pogodno za kopiranje konfiguracijskih datoteka na više različitih računala koja pokreću istu aplikaciju s istim postavkama.




Slika 2.10. Distribucija datoteka i direktorija

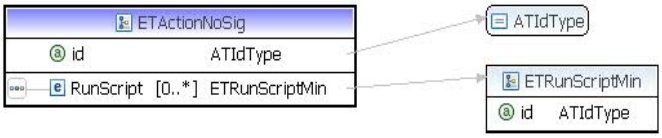
Važna zadaća Lusce je distribucija aplikacija. U fazi pripreme aplikacije se distribuiraju na sva računala. Za definiranje distribucije jedne aplikacije potrebno je navesti identifikator aplikacije koja se distriburia i listu *Host* elemenata s identifikatorima računala na koja će se smjestiti aplikacija. Aplikacije će prije distribucije biti pripremljene u skladu specifikacija

navedenim u definicijama. Elementi za definiciju distribucije aplikacije prikazani su slikom 2.11.




Slika 2.11. Distribucija aplikacija

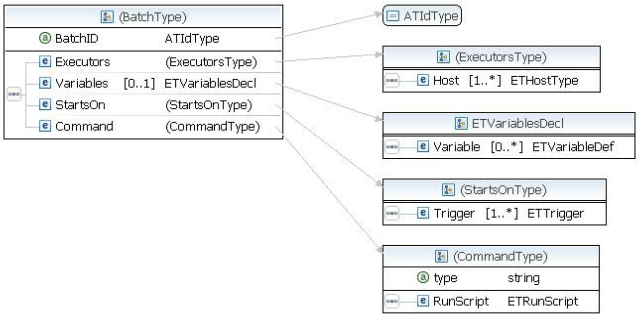
Osim distribucije resursa i aplikacija ponekad je potrebno određeno računalo dovesti u neko pretpostavljeno stanje. Postavljanje tog stanja vrše skripte, ali skripte o čijem izvođenju Lusca ne mora voditi brigu već ih samo pokrenuti bez parametara i povratnog rezultata. Tomu služi odsječak inicijalizacija. Inicijalizacija se vrši za svako računalo posebno. Inicijalizacija se definira identifikatorom računala (`id`) i listom RunScript elemenata koji sadrži identifikator skripti (`id`). Slika 2.12. prikazuje sve potrebne elemente za definiranje inicijalizacija na jednom računalu.




Slika 2.12. Pokretanje skripta na računalu u inicijalizacija i zaključku

Nakon što je završena faza pripreme i ispitna okolina postavljena u svoje početno stanje, Lusca prelazi u fazu izvršavanja. U ovoj fazi Lusca izvodi izvodi skupne obrade (engl. *batch*).

Definicije jedna skupne obrade prikazana je slikom 2.13. Jedna skupna obrada definira svoj jedinstveni identifikator, jednog ili više računala koji izvode skupnu obradu, jedan ili više okidača na koje se aktivira izvršavanje naredbe (implementacija trenutno dozvoljava samo pokretanje skripte kao naredbe).

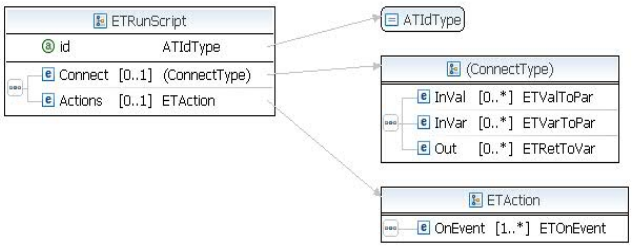



Slika 2.13. Batch izvršavanje zadataka

Svaka grupna obrada definira popis okidača koji može pokrenuti tu skupnu obradu. Okidač definira identifikator agenta (računala) od kojeg je signal primljen i identifikator signala. Ovime je omogućeno odabir točno određenog signala koji je poslan s točno određenog domaćina, ako nije važno s kojeg je domaćina signal poslan za identifikator agenta navodi se *any*.

Osim globalnih varijabli navedenih u definicijama, koje su vidljive na svim agentima, moguće je definirati lokalne varijable unutar jedne skupne obrade. Ove varijable su lokalne na svakom agentu, te su samo njemu vidljive. Definicija takvih varijabli jednaka je u bloku definicija.

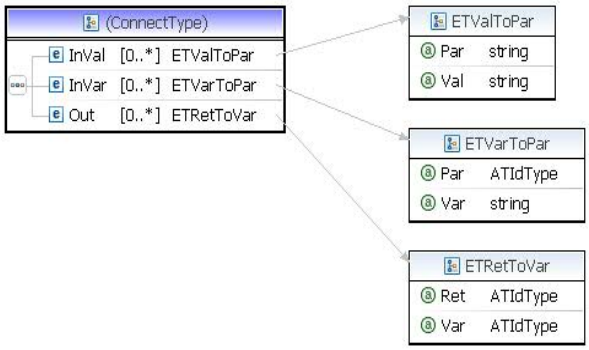
Komanda određuje pokretanje skripte koja se definira elementom prikazanim slikom 2.14. Pokretanje skripte (`RunScript`) određeno je identifikatorom skripte navedenim u definicijama. Pri pokretanju skripte potrebno je povezati njene parametre koji su definirani kao ulazni/izlazni u definicijama skripte s vrijednostima ili varijablama (Slika 2.15.). Svaka skripta definira svoj kontekst izvršavanja. Kontekst izvršavanja određen je uvjetom za njeno pokretanje i reagiranjem na događaje. Pokretanje skripte ovisi o nadelementu `RunScript` elementa, dok se se reakcija na događaja definira elementom `OnEvent`.




Slika 2.14. RunScript koji pokreće skriptu i veže njene parametre

Ulaze skripte moguće je vezati  uz neku ili fiksnu vrijednost, dok se povratna vrijednost povezuje s varijablom. Povezivanje varijabli se vrši uparivanjem identifikatora parametra i

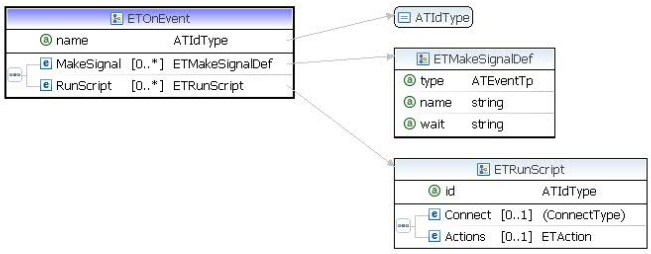
identifikatora varijable, a povezivanje s vrijednosti uparivanjem identifikatora parametra i vrijednosti.




Slika 2.15. Vezanje parametara skripte na vrijednosti i varijable

Identifikator događaja mora biti jednak identifikatorom signala na koji reagira. Nakon što se se primi signal za događaj moguće je generirati signal ili pokrenuti skriptu (Slika 2.16). Generirani signal je određenog tipa (`type`) i imena (`name`), a moguće je signal odgoditi tako da se pošalje nakon određene količine vremena (`wait`)

Pokrenutanje skriptu kad se dogodi događaj omogućava rekurzivno pokretanje skripti. Isto tako je omogućeno da određena skripta bude pokrenuta nakon što se dogodio potreban slijed događaja. Ovakav događajno orijentirani sustav omogućuje oporavak od pogreške i veliku fleksibilnost.




Slika 2.16. Opis događaja

Nakon što završi faza izvršavanja, izvršava se faza zaključka. Ova faza koristi se za izvršavanje zaključnih radnji nakon izvršavanja ispitnog slučaja. Zadaća zaključne faze je omogućiti izvršavanje skripti koje obavljaju posao završavanja ispitnog slučaja. Jedna od primjena može biti čišćenje ispitne okoline od nepotrebnih datoteka ili vršenje finalne analize rezultata ispitnog slučaja.

Definicija ove faze ekvivalentna je fazi inicijalizacije pripremne faze, a elementi koji su potrebni za definiranje ove faze prikazani su slikom 2.12.

# 3. Korištene tehnologije

## 3.1. Java

Java je programski jezik originalno razvijen od Sun Microsystems. Java aplikacije obično se prevode u međujezik (engl. *bytecode*), iako je moguće prevođenje u strojni kod. Izvršavanje međujezika izvodi se na virtualnom stroju koji ga interpretira. Java je dizajnirana s nekoliko osnovnih ciljeva:

- mora koristi objektni programski model
- mora se moći izvoditi na različitim operacijskim sustavima bez potrebe za izmjenama izvornog koda, ili potrebe za ponovnim prevođenjem izvornog koda
- treba imati ugrađenu mrežnu podršku
- treba biti jednostavna za korištenje tako da se uzmu dobra svojstva drugih objektno orijentiranih jezika

Jednostavna sintaksa jezika i podrška za različite operacijske sustave omogućili su da Java postane najrasprostranjeniji programski jezik današnjice.

## 3.2. Eclipse

Eclipse je programski okvir (engl. *software framework*) otvorenog koda, neovisn je o operacijskom sustavu i baziran je na programskom jeziku Java. Eclipse pruža proširivu razvojnu platformu dizajniranu za razvoj integriranih razvojnih okruženja IDE (engl. i*ntegrated development environment*), raznih alata, aplikacijskih okruženja (engl. *aplication frameworks)*, ali i samostalnih aplikacija (engl *Rich Client Platform - RCP*). 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 razvio IBM kao nasljednika VisualAge porodice alata, a danas Eclipse-om upravlja Eclipse Foundation, nezavisni i neprofitabilni konzorcij koji se sastoji od vodećih softverskih kompanija. Veliki broj značajnih IT kompanija prihvatio Eclipse kao budući okvir (engl. *framework*) za njihova integrirana razvojna okruženja, 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 specifično ponašanje ovoj generičkoj platformi, dodavanjem Java alate poput uređivača Java koda, prevoditelja i program za pronalaženje pogrešaka (engl. *debugger*) i grafičko korisničko

sučelje. JDT podržava proširivanje tako da korisnik može dodati novu funkcionalnost (npr. dodavanje funkcionalnosti za izračuna broja linija koda u projektu). JDT je primjer mnogobrojnih 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, ispitivanju, izgradnji izvršnih arhiva i isporuci (engl. *deployment*) Eclipse dodataka, fragmenata, mogućnosti (engl. *features*) i RCP aplikacija.

Pisanje Eclipse dodataka 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, ispitivanje dodatka i pakiranje dodatka u oblik pogodan za isporuku. PDE pojednostavljuje ovaj zadatak, integracijom u radno mjesto (engl. Workbench) dodajući čarobnjake za izradu pojedinih dijelova dodataka i povezivanjem s JDT kao Java razvojnim okruženjem.

Eclipse je proširiva platforma, pruža osnovni skup servisa koji kontroliraju dodatke kako 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.

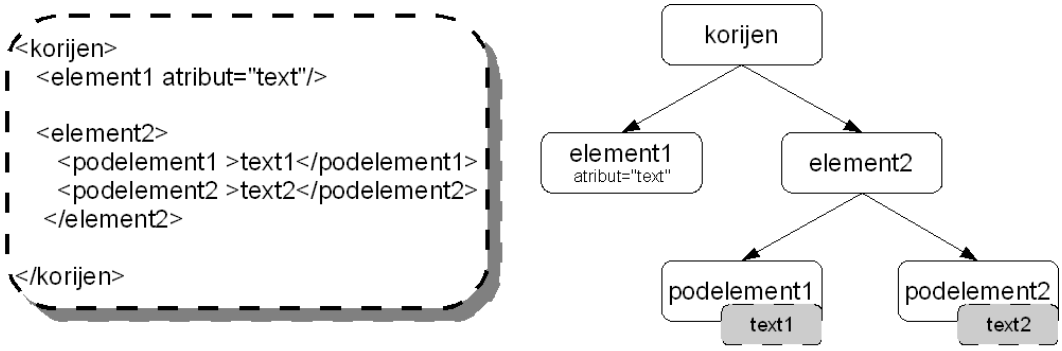
Iako je Eclipse platforma specijalizirana za izgradnju integriranih razvojnih okruženja, Eclipse platforma može se iskoristiti za izgradnju RCP aplikacijama, koje kao svoj osnovnu jezgru koriste Eclipse Runtime i grade se kao skup Eclipse dodataka. Eclipse Runtime jer skup servisa za učitavanje razreda u virtualni stroj i upravljanje ovisnostima između pojedinih dijelova aplikacije. RCP aplikacije isporučuju se kao Eclipse Runtime i skup potrebnih dodataka da bi aplikacija funkcionirala.

## 3.3. XML

### 3.3.1. XML

XML (engl. eXtensible Markup Language) je jezik opće namjene zasnovan na oznakama. XML je nastao pojednostavljenjem jezika SGML (engl. Standard Generalized Markup Language), a osnovna namjena mu je olakšati razmjenu podataka između različitih informacijskih sustava. Jezik XML koristi tekstualni zapis podataka neovisan o računalnoj platformi, dizajniran tako da bude čitljiv čovjeku. XML je opće primjenjiv jezik, jer dodavanjem semantičkih ograničenja na XML dobiva je novi jezik za rješavanje određenog problema. Danas postoje mnogi formalno definirani jezici zasnovani na oznakama bazirani na XML-u, između ostaloga i XHTML (pokušaj pojednostavljivanja HTML-a, koji je baziran na SGML-u), RSS, MathML, GraphML, SVG, MusicXML i drugi.

Jezikom XML se u tekstualnom obliku opisuju stablaste strukture podataka. Slika 3.1 prikazuje primjer XML dokumenta i pripadajućeg stabla.




a) XML dokument b) Podatkovno stablo XML dokumenta

Slika 3.1. Primjer XML dokumenta i pripadajućeg podatkovnog stabla

XML dokumenti sastoje se od elemenata. Elementi započinju i završavaju sa oznakama i mogu imati sadržaj i svojstva (engl. *attributes*). Primjer elementa s definiranim sadržajem i svojstvom jest `<elem attr="vrijesnost_svojstva">sadrzaj</elem>`. Oznake se unose između znakova "<" i ">". Početna oznaka sadrži i svojstvo elementa označeno nizom `attr="vrijednost_svojstva"`, oznaka može imati neograničeni broj svojstva. Sadržaj elementa se unosi između oznaka. Završna oznaka se stavlja između znakova "</" i ">". XML elementi mogu imati proizvoljan broj podelemenata. Ako element nema podelemenata niti sadržaj, onda se takav element naziva praznim elementom i označava se s `<element/>`.

XML raspoznaje dvije razine ispravnosti dokumenta:

- uređenost (engl. *well - formed* ) - uređen je dokument koji se zadovoljava pravila XML sintakse. Provjera uređenosti provodi se upotrebom XML parsera.
- valjanost (engl. *valid*) – dokument je valjan ako se pridržava semantičkih pravila koja su definirana posebno za svaku primjena XML posebno. Pravila se definiraju u posebnom dijelu XML dokumenta ili u vanjskoj datoteci kao DTD (engl. Document Type Definition) ili XML Schemu. Kako bi XML dokument bio valjan prvo mora biti uređen. Provjera valjanosti vrši se posebnim alatima koji nakon parsiranja XML dokumenta provjeravaju da li svi elementi i svojstva zadovoljavaju pravila definirana za taj dokument.

Prednosti XML-a su čitljivost od strane računala i čovjeka, neovisnost o računalnoj platformi, jednostavna sintaksa i općenitost. XML dokumenti su tekstualni dokumenti s podržanim Unicode standardom koji omogućava zapis gotovo svih pisama u računalni oblik. Jednostavna sintaksa omogućava razvoj efikasnih i jednostavnih XML parsera. Jezikom XML je moguće prikazati strukture podataka proizvoljne složenosti. Hijerarhijska struktura XML je pogodna za gotovo sve tipove dokumenata, a otvorenost pogoduje za izradu bilo kakvih dokumenta za razliku od vlasničkih formata koji to ne omogućuju.

Najveći nedostatak jezika XML je veličina XML dokumenata. Povećana količina podataka zahtjeva veće memorijske spremnike i dodatno se povećavaju opterećenja tijekom obrade podataka.

### 3.3.2. XML Schema

XML gramatika je skup pravila i ograničenja koja se odnose na strukturu i sadržaj XML dokumenta. Najčešće korištena XML gramatika jest XML Schema, a još se često koriste DTD gramatika koja je manje ekspresivna i RELAX NG gramatika koja je kasnije prihvaćena od standardizacijskih tijela. Glavna prednost XML Scheme je njen zapis, koji je isto XML dokument pa se čitanje može vršiti XML parserom. XML Schema daje opis tipova XML dokumenta, obično kao skup ograničenja koja se odnose ne strukturu i sadržaj elemenata XML dokumenta kako bi taj XML bio valjan. XML Schema Definition (XSD) je instanca XML Scheme pisana u XML. XSD određuje tip XML definirajući tipove elementa koji se mogu pojavljivati, ograničenja tipova i svojstva, te odnose između pojedinih elemenata.

XSD definira slijedeće stvari:

- elemente i atribute koje se mogu pojaviti u dokumentu
- elementi moraju imati ime i tip
- elementi mogu sadržavati jednostavne unaprijed definirane tipove (string, integer, decimal, booelan, date,duration,URI)
- svaki element definira broj svojih pojavljivanja (jednom, jednom ili nijednom, nijednom ili više puta, jednom ili više puta,odrešen broj puta)
- elementi mogu sadržavati druge elemente, ili u sebi mogu definirati strukture koje opisuju elemente
- elementi u sebi mogu sadržavati kompleksne tipove definirane u definiciji elementa
- tipovi elementa definiraju se posebno i pridružuju elementu, ili se definiraju unutar definicije elementa.

XML shema podržava podatkovne tipove i na taj način nam omogućava opis dozvoljenog sadržaja dokumenta, opis dozvoljenog sadržaja dokumenta i postavljanje ograničenja. Primjer XML Scheme dan je odsječkom XML 3.1, dok je XML dokumenti koji zadovoljava tu XML Schemu dan odsječkom XML 3.2.

XML 3.1: Primjer XML Scheme

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:complexType name="Osoba">
      <xs:sequence>
         <xs:element name="ime" type="xs:string"/>
         <xs:element name="prezime" type="xs:string"/>
         <xs:element name="grad" type="Grad"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="Adresa">
      <xs:attribute name="grad" type="xs:string"/>
      <xs:attribute name="postanski_broj" type="xs:int"/>
   </xs:complexType>

   <xs:element name="Osoba" type="Osoba"/>
</xs:schema>
```

XML 3.2: Primjer XML koji zadovoljava XML Scheme-u iz primjera 3.1

```
<?xml version="1.0" encoding="UTF-8"?>
<osobe:Osoba xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:noNamespaceSchemaLocation="osobe.xsd">
  <ime>Ivan</ime>
  <prezime>Horvat</prezime>
  <grad grad="Zagreb" postanski_broj="10000"/>
</osobe:Osoba>
```

### 3.3.3. Simple XML serijalizacijski okvir

Simple XML je okvir za serijalizaciju Java objekata visokih performansi. Osnovi cilj ovog okvira brzi razvoj aplikacija u Javi koje koriste XML, tako da se pojednostavni čitanje i pisanje XML-a. Simple pruža mogućnosti koje su slične C# XML serijalizaciji na Java platformi, ali pruža i dodatne mogućnosti za nadzor pojava grešaka i manipulaciju objektima.

Sustav Simple XML jednostavan je za korištenje i temelji se na Java napomena (engl. *annotation*) i jednom *persistent* – postojanom objektu koji čita i piše sadržaj objekata u XML. Stroj za postojanost (engl. *persistence engine*) može obrađivati složene objektne strukture i na taj način omogućuje serijalizaciju i deserijalizaciju rekurzivnih struktura.

Za razliku od ostalih XML Java okvira, Simple ne zahtjeva dodatne konfiguracije i pridruživanja da bi se mogla provesti serijalizacija objekata neovisno o njihovoj složenosti, a XML Schema nalazi se u Java razredima kao Java napomene.

XML dokumenti koji se generiraju prilikom serijalizacije objekata, Simple okvir generira tako da su čitljivi čovjeku u bilo kojem uređivaču teksta. Simple se brine da su svi XML elementi i atributi ispravno uvučeni tako da čovjek može razaznati XML strukturu.

Kako bi se objekt uspješno serijalizirao u XML objekt potrebno je u razred objekta na svojstvo razreda staviti napomene koje označavaju u koji dio XML će se serijalizirati to svojstvo. Podržani Simple XML elementi su:

- Root – označava korijenski element XML, svaki razred koji će biti serijaliziran mora ima ovu napomenu
- Element – označava XML element, serijalizacija se vrši rekurzivno prema pravilima koja su zadana za taj razred
- Attribute – označava XML atribut, u njega se mogu serijalizirati samo elementarni Java tipovi poput String, Integer, Date, boolean, Currency.
- Text – označava XML tekst elementa, isto kao i atribut u njega se mogu serijalizirati samo elementarni Java tipovi
- ElementList – označava listu elemenata, za listu se može definirati da li će imati nad element koji će element koji će označavati listu a u nju će biti smješteni svi elementi liste, ili se može definirati da su elementi "inline" što znači da nema nad elementa u koji se smještaju ti sadržaji liste već se sve smješta unutar trenutnog element liste.
- ElementArray – označava polje XML elemenata, razlika na u odnosu na listu je da se ovdje bilježi veličina polja koja je tada fiksna

Jednostavan primjer razreda koji se može serijalizirati prikazan je Java odsječkom 3.1. U njemu možemo primijetiti tri različite Java napomene Root koji označava da razred `Primjer` može biti korijenski element XML, svojstvo razreda `text` serijalizira kao XML podelement, a svojstvo `index` kao XML atribut.

*Java 3.1: Primjer korištenja Simple XML-a*

```
import org.simpleframework.xml.Attribute;
import org.simpleframework.xml.Element;
import org.simpleframework.xml.Root;

@Root
public class Primjer {
          @Element
          private String text;
          @Attribute
          private int index;

          public Primjer() {
          }

          public Primjer(String text, int index) {
                    this.text = text;
                    this.index = index;
          }

          public String getText() {
                    return text;
          }

          public void setText(String text) {
                    this.text = text;
          }

          public int getIndex() {
                    return index;
          }

          public void setIndex(int index) {
                    this.index = index;
          }

}
```

Kako bi se instanca objekta `Primjer` serijalizirala potreban je objekt `Persister`, čija instanca prima objekt označen napomenama i generira XML. Primjer serijalizacije dan je odsječkom Java 3.2.

*Java 3.2: Primjer serilaizacije objekta pomoću Simple XML-a*

```
Serializer serializer = new Persister();
Primjer primjer = new Primjer("neki tekst",123);
File datoteka = new File("primjer.xml");

serializer.write(primjer, datoteka);
Primjer primjer2=serializer.read(Primjer.class, datoteka);
```

Kod deserijalizacije potrebno je navesti razred koji se deserijalizira, kako bi okvir znao pročitati napomene i prema njima ispravno popuniti polja objekta iz XML-a. Nakon uspješnog izvođenja Java odsječka 3.2.,dobiva se XML 3.3.

XML 3.3: Dobiveni XML serijalizaciom pomoću Simple XML-a

```
<Primjer index="123">
   <text>neki text</text>
</Primjer>
```

Kod liste objekata potrebno je odrediti da li je lista u liniji (engl. *inline)*, odnosno da li se lista serijalizira s XML nadelementom ili bez njega. Primjer korištenja *inline* opcije za serijalizaciju liste dan je s odsječkom Java 3.3.

*Java 3.3: Primjer korištenja inline lista*

```
@Root
public class Korijen {
          @Element
          String element1;

          @ElementList(inline=true)
          ArrayList<Element2> list;
}
@Root
class Element2{
          @Attribute
          String id;

}
```

Serijalizacijom objekta razreda `Korijen`, korištenjem *inline* svojstva, dobiti ćemo XML prikazan u odsječku XML 3.4.

XML 3.4: Dobiveni XML serijalizaciom inline liste

```
<korijen>
          <element1>text</element1>

          <element2 id="1" />
          <element2 id="2" />
          <element2 id="3" />

</korijen>
```

Ako ne bi bilo uključeno svojstvo *inline*, serijalizacija objekata razreda `Korijen` generirala bi XML prikazan odsječkom XML 3.5.

Kod korištenja "inline" lista potrebno je paziti da nema miješanja elemenata liste s drugim elementima koji ne pripadaju listi.

XML 3.5: Dobiveni XML serijalizaciom liste bez korištenja inline opcije

```
<korijen>
        <element1>text</element1>
        <lista>
                <element2 id="1" />
                <element2 id="2" />
                <element2 id="3" />
        </lista>
</korijen>
```

# 4. Eclipse

Eclipse je platforma koja je dizajnirana za izgradnju integriranih web i aplikacijskih razvojnih alata. Sam dizajn platforme ne pruža veliku krajnju funkcionalnost, ali vrijednost platforme je u poticanju brzog razvoja integriranih mogućnosti koje se baziraju na modelu dodataka (engl. *Plugi-in model*).

Jezgra Eclipse platforme je arhitektura za dinamičko otkrivanje, učitavanje i pokretanje dodataka. Sama platforma odrađuje logistiku traženja i pokretanja ispravnog koda, dok korisničko sučelje pruža standardni korisnički navigacijski model. Svaki dodatak može se orijentirati na mali broj zadatak koje mora izvršavati, zadaci mogu biti bilo što, definiranje, ispitivanje, animiranje, objavljivanje, prevođenje, izrada dijagrama ili nešto drugo, jer Eclipse ne postavlja nikakva ograničenja.

## 4.1. Arhitektura Eclipse platforme

Eclipse platforma definira otvorenu arhitekturu tako da svaki razvojni tim može fokusirati na razvoj svog dodatka, na taj način svaki se razvojni tim doprinosi sa svojim vještinama i znanjem Eclipse projektu bez opterećenja kodom drugih projekata.

Eclipse platforma koristi model zajedničkog radnog mjesta (engl. *Workbench*) u koje se integriraju svi alati koristeći dobro definirane točke proširenja (engl. *extension points*). Na taj način platforma pruža korisniku jedinstveni izgled svih alata i pruža integrirano upravljanje svih resursa (projekata, direktorija, datoteka) kojima manipuliraju ti alati.

Sama platforma izgrađena je od više slojeva dodataka od kojih svaki pruža svoja proširenja za njihovo prilagođavanje, a pružaju točke proširenja za svoju prilagodbu.

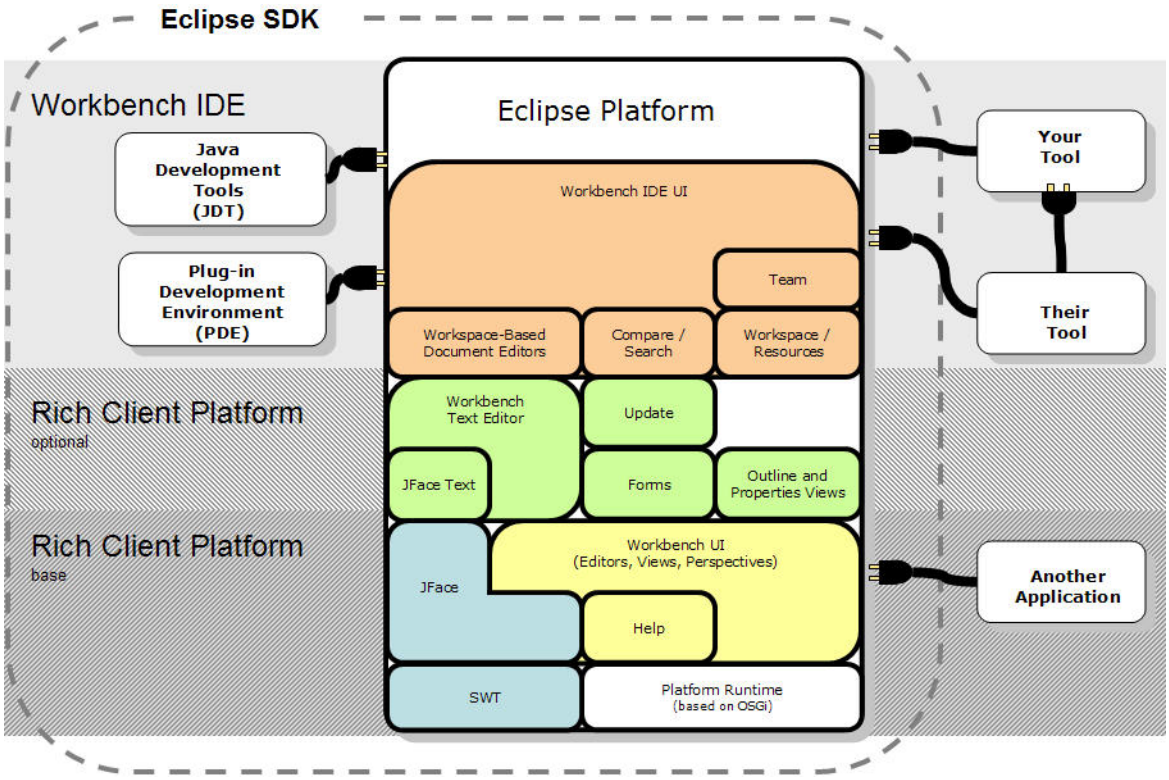
Zahvaljujući arhitekturi Eclipse platforme programer ne mora brinuti o operacijskom sustavu na kojem se izvodi i na taj način omogućava programeru da se usredotoči na problem koji rješava, a ne na implementacije na različitim sustavima.

Platforma je strukturirana u podsustave koji su implementirani kao jedan ili više dodataka, a svi podsustavi izgrađeni su na malom pokretačkom stroju (engl. *runtime engine*). Skup podsustava koji grade Eclipse prikazani su slikom 4.1. Neki od podsustava implementiranju vidljive mogućnosti platforme, dok drugi vrše ulogu biblioteka za izgradnju vidljivih dijelova platforme.

Eclipse SDK je paket (engl. *bundle*) takvih podsustava čija namjena je osim platforme pružiti i alate koji će omogućiti integrirano okruženje za razvoj Java aplikacija (Java development tools – JDT) ali i za razvoj dodataka za Eclipse platformu (Plug-in Development Environment – PDE). Na taj način Eclipse postaje razvojna platforma za samu sebe.

Eclipse platformu možemo podijeliti u tri sloja (Slika 4.1.):

- osnovni RCP dio
- opcionalni RCP dio
- radno mjesto integriranog razvojnog okruženja




Slika 4.1. Arhitektura Eclipse platforme prikazana preko podsustava

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 i oni su 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. 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.

Opcionalni RCP dio dodaje dodatne višestruko iskoristive elemente, koji su proizašli iz integriranog razvojnog okruženja ali u praksi su se pokazali pogodni i za druge aplikacijske primjene, zbog toga su izdvojeni u posebni sloj. U osnovni RCP dio nisu ugrađeni kako bi se smanjila veličina osnovnog RCP dijela. Ako RCP aplikacija nema potrebe za nekim dijelom iz opcionalnog dijela RCP on se ne mora distribuirati

Radno mjesto integriranog razvojnog okruženja pruža funkcionalnosti koje su specifična za integrirana razvojna okruženja, a njenu funkcionalnost iskorištavaju, nadograđuju i implementiraju alati popu JDT-a i PDE-a.

Dodatkom se može proširiti Eclipse platforma na bilo kojem mjestu, te je na taj način omogućena nadogradnja od najniže do najviše razine apstrakcije. Svaki dodatak opet može biti proširen i na taj način dobivena je beskonačno proširiva platforma.

## 4.2. Eclipse 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.

Dodatak je najmanji dio Eclipse platforme koji se može razviti i isporučiti zasebno. Obično su mali alati 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 i određeni broj proširenja (engl. *extension*) prema jednoj ili više točka proširenja drugih dodataka.

Točka proširenja dodatka može biti proširena od nekog drugog dodatka. 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, detektiraju se 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 (engl. *runtime dependencies*). Druga datoteka je `plugin.xml`, ona je XML datoteka koja sadrži opise proširenja koja dodatak pruža i 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 pokretanjem (engl. *startup*) i manjom potrošnjom memorije.

Korištenje manifesta baziranom na XML- u omogućuje jednostavniju izradu alata za izradu dodataka, primjer takvog alata je PDE koji je uključen u Eclipse SDK.

Dodatak se aktivira 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 deaktivacije ili gašenja Eclipse Platforme .

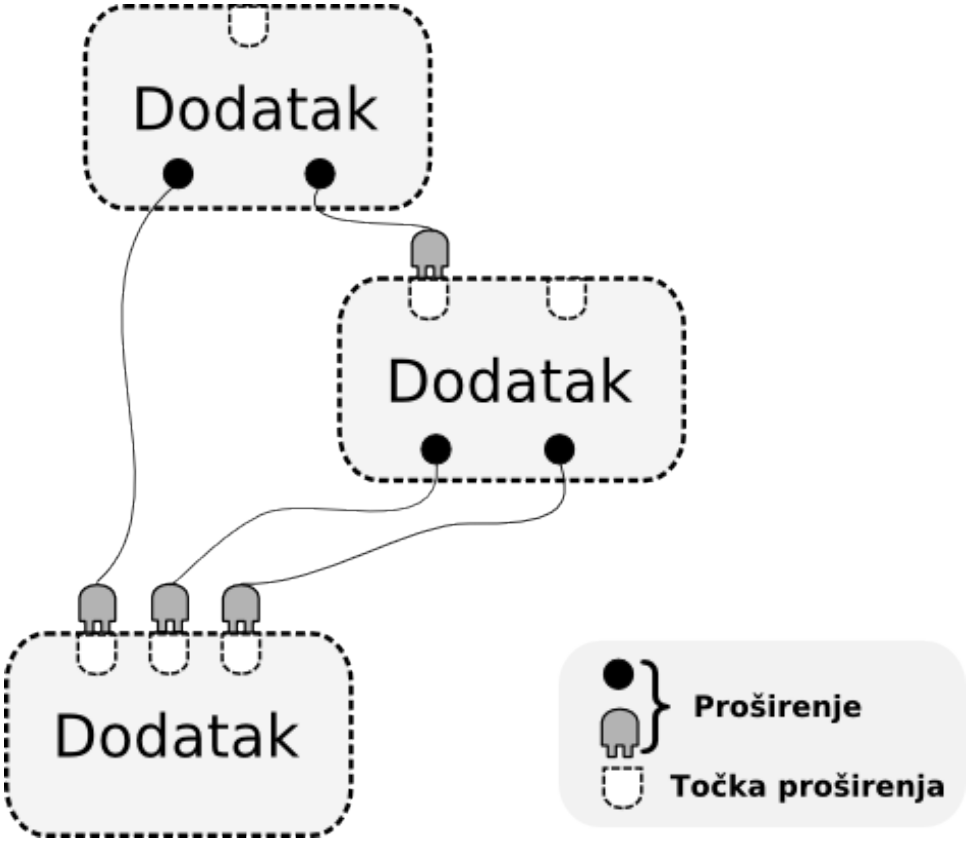
## 4.3. Proširivanje Eclipse platforme

### 4.3.1. Točke proširenja

Točke proširenja su glavni koncept arhitekture dodataka. Model povezivanja dodataka 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 4.2. gdje je prikazan model povezivanja točaka proširenja i proširenja tri dodatka, koji se međusobno proširuju, kako bi dobili željenu funkcionalnost, pri tome iskorištavajući usluge nekog drugog dodatka.

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




Slika 4.2. Mehanizam točaka proširenja i proširenja

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

### 4.3.2. Kasno učitavanje

Kako je cijeli Eclipse IDE izgrađen od malog pokretačkog stroja (engl. *runtime engine*) i velikog broja dodataka Eclipse platforma je dizajnirana tako da se smanji potrošnja memorije. Omogućavanje instalacije neograničenog broja dodataka i njihovo učitavanje povećalo je potrošnju memorije, zbog toga je u uveden mehanizam kasnog učitavanja (engl. *Lazy loading*). Mehanizam 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, a na temelju registra dodaje elemente poput izbornika, gumbi, alatnih traka i slično. Stvarni kod koji se izvodi kod aktiviranja radnji učitava se prilikom prve aktivacije. Na taj način moguće je imati neograničen broj dodataka u jednoj instalaciji Eclipse-a, a koristi samo one koji su u jednom trenutku potrebni.

### 4.3.3. Manifest dodatka

Manifest dodatka raspodijeljeni je u dvije datoteke *Manifest.mf* i *plugin.xml*. Manifest dodatka može se ručno uređivati pomoću tekst uređivača pošto je je *manifest.mf* tekstualna datoteka, a *plugin.xml* XML datoteka koja je čitljiva 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 uređivača. 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.

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 dodatak (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;".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.

Kako bi dodatak bio uspješno učitan, prije učitavanja 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 kojeg navodimo listu dodataka odvojenih zarezom koji moraju postojati. Osim dodatka koji mora postojati možemo definirati i koja točno verzija mora biti zadovoljena. 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.

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

U *plugin.xml* definiraju se dvije osnovne stvari proširenja (*extension*) i točke proširenja (*extension-point*). Datoteka *plugin.xml* sadrži opise proširenja i točaka proširenja u XML obliku. Definicija tog XML dana je s slijedećom DTD definicijom:

XML 4.1: DTD za plugin.xml

```
<?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
>
```

Iz navednog DTD-a slijedi jednostavan primjer *plugin.xml*:

XML 4.2: Primjer općentiog zapisa 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>
```

Za proširenje u XML element *extension* potrebno je staviti atribut *point* kojim se definira ime točke proširenja koju želimo proširiti, moguće je staviti atribute *Id* i *name* no oni nisu potrebni. Pod elementi se definiraju drugačije za svaku točku proširenja a 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 zahtjevati drugačije atribute, no obično se traže *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.

## 4.4. Eclipse grafičkog korisničkog sučelja

### 4.4.1. SWT

Standard Widget Toolkit – SWT je alat elemenata grafičkog korisničkog sučelja (engl. *widget toolkit*) originalno razvijen od IBM za potrebe razvoja Eclipse razvojnog okruženja. Eclipse ga koristi kao osnovnu za izgrađuju grafičkog korisničko sučelja. SWT je nastao nakon što se pokazala potreba za još boljim gradivnim elementima za izgradnju grafičkog korisničkog sučelja za Javu. Kako AWT i Swing od Sun Microsystem-a nisu pružali dobre performanse i izvorni izgled sučelja na različitim operacijskim sustavima IBM je krenuo u razvoj svojeg alata elemenata grafičkog korisničkog sučelja koji bi na različitim operacijskim sustavima omogućio izgled grafičkih elementa koji odgovara operacijskom sustavu.

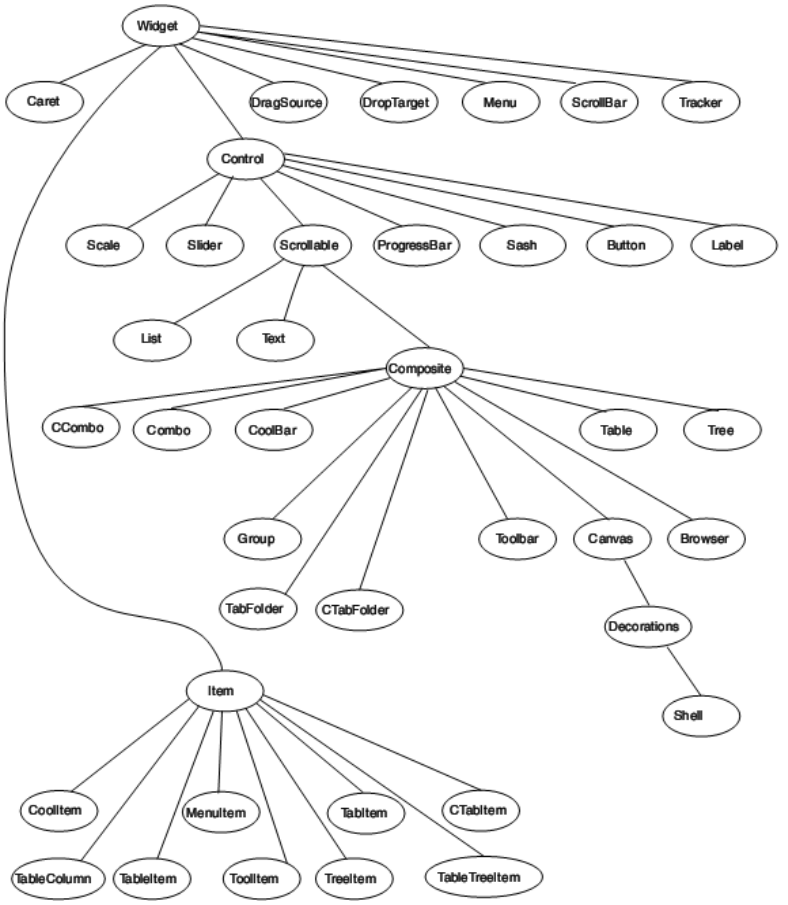
AWT se sastoji od grafičkih elementa za koje je postoji implementacija na svim operacijskim sustavima. Za razliku od AWT-a, Swing ne koristi implementaciju operacijskog sustava već sam iscrtava sve grafičke elemente. Na taj način dobiva sve grafičke elemente, no ovaj način ne daje dobre performanse i ne može dobiti izvorni izgled grafičkih elemenata operacijskog sustava.

SWT je preuzeo najbolje koncepte iz AWT i Swing-a. Ako postoji izvorna implementacija grafičkog elementa. SWT se preko JNI sučelja koristi sistemskim pozivima kako bi iscrtao taj grafički element, ako ne postoji taj grafički element, na nekom operacijskom sustavu, on ga emulira tako da se ručno iscrtava. Tako se postižu dobre performanse i izvorni izgled na operacijskim sustavima za koje je SWT podržani. SWT ne radi na svim operacijskim sustavima za koje postoji Java implementacija već samo na onim za koje je napisan, a za svaku platformu potrebno je napraviti posebnu implementaciju. Popis operacijskih sustava za koje je SWT dostupan dan je tablicom 4.1.

*Tablica 4.1. Implementacija SWT na različitim operacijskim sustavima*

| Operacijski sustav | Izvorna implementacija SWT |
|---|---|
| Microsoft Windows | Windows API |
| AIX, FreeBSD, Linux, HPUX | Motif,GTK+ |
| MacOS | Carbon |
| QNX | Photon |
| Java | SWTSwing |

Osnovni razred kojeg naslijeđuju svi grafički elementi je `Widget`, nasljeđivanjem ovog razreda svi grafički elementi dobivaju slična svojstva, a to su događaji (engl. *events*) i raspored elemenata (engl. *layout*). Nasljeđivanje svih grafičkih elemenata prikazano je slikom 4.3.




Slika 4.3. Prikaz stabla nasljeđivanja SWT grafičkih elemenata

### 4.4.2. JFace

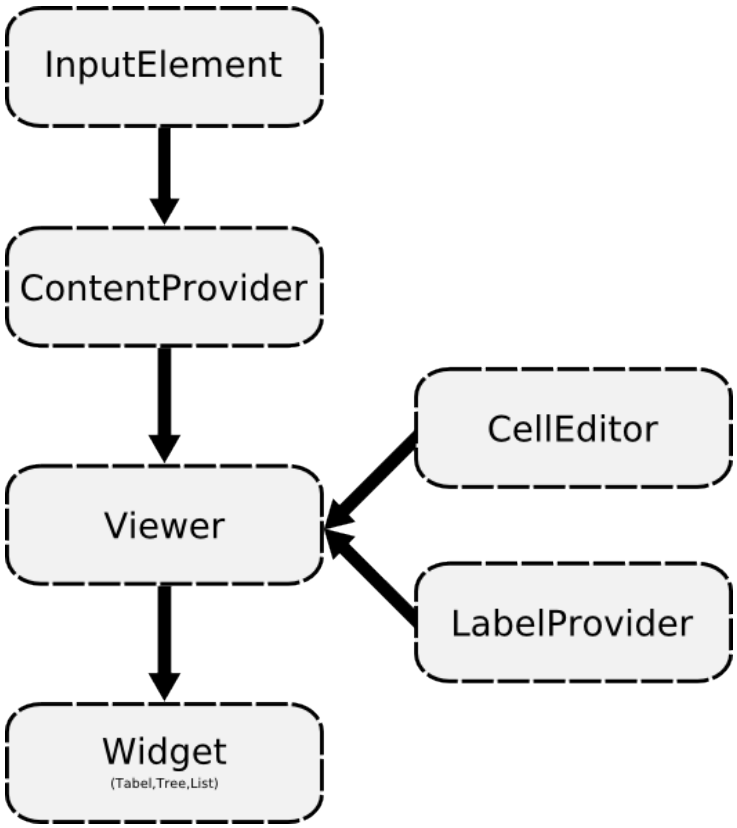
JFace API bazira se na SWT-u i pojednostavljuje programeru izradu složenijih grafičkih elemenata kao što su prikazi (engl. *viewers*), dijalozi (engl. *dialogs*), čarobnjaci (engl. *wizards*), upravitelji resursima (slike, fontovi) i ostali. Neke od JFace komponenta su specifične za Eclipse radno mjesto te kao takve su sastavni dio dodatak Eclipse radno mjesto (*workbench.jar*). No većina komponenata može se koristi nezavisno od Eclipse radnog mjesta. Najvažnija komponenta JFace su prikazi koji će biti detaljnije opisani.

Iako SWT pruža direktno sučelje grafičkim elementima operacijskog sustava, SWT je ograničen na korištenje elementarnih podatkovnih struktura kao što su znakovni nizovi (engl. *string*) brojeve i slike. Ovakav pristup dobar je za veliki broj aplikacija, ali otežava rada s objektno orijentiranim strukturama kad se podaci moraju prikazati na različite načine poput stabala, tablica i tekstualnih uređivača. Tome služe JFace Viewers, kako bi omogućili lakše baratanje objektno orijentiranim strukturama između pojedinih objekata i grafičkih elemenata.

JFace Viewers su najkorisniji elementi za izgradnju grafičkog sučelja temeljenoga na Eclipse platformi. Iako imaju u imenu *prikaz*, njihove mogućnosti pružaju puno više, tako svi razredi Viewer omogućuju i promjenu sadržaja koju prikazuju. Ime prikaz dolazi od

paradigme Model-Prikaz-Upravitelj (engl. *Model-Viewer-Controler*) odnosno MVC. Eclipse ostvarenje ove paradigme svodi se na samo jedno postojanje podataka (model) i više različitih prikaza tih istih podataka (prikazi), a korisničke akcije utječu na podatke (upravitelj). Ovakav *"design pattern"* omogućuje postojanje samo jedne instance objekta u memoriji koji se prikazuju na više različitih načina (tablice, stabla).

Slika 4.4 prikazuje povezivanje pojedinih dijelova JFace viewer s ostalim elementima za prikaz.




Slika 4.4. Prikaz odnosa između pojedinih dijelova JFace Viewera

Ovakav prikaz temelji se na nekoliko razreda koji međusobno surađuju. Glavni je razred prikaza (*viewer*). On vrši cijelu funkcionalnost. Na instancu objekta prikaza potrebno je pridružiti jedan od grafičkih elemenata SWT-a u kojem će se prikazivati podaci, to može biti stablo, tablica ili lista. Zatim je potrebno implementirati razred pružatelja sadržaja (*ContentProvder*) koji će iz primljenog objekt (*InputElement*) filtrirati samo one elemente sadržaja objekta koje treba prikazati. Pružatelj sadržaja isto tako će na odgovarajući način vračati elemente koje odgovaraju grafičkom elementu za koji je namijenjen, tako će za stablo biti vraćani element i njegova djeca, dok će za tablicu biti vraćani elementi po stupcima. Pružatelj sadržaja se pridružuje prikazu. Ukoliko je potrebno moguće je implementirati i pružatelj tekstualnoga prikaza (*LabelProvider*), čija je zadaća prikaz primljenih elemenata od pružatelja sadržaja prikazati kao tekst, ako nije implementiran i pridružen pružatelj tekstualnoga prikaza, tada se prikazuje sadržaj koji vraća metoda

`toString()` svakog elementa. JFace omogućuje i promjenu sadržaja objekta koji se prikazuje. Kako bi imali tu mogućnost potrebno je prikazu pridružiti instancu objekta koji implementira uređivač ćelije (*CellEditor*).

Nakon pridruživanja svih željenih mogućnosti prikazu, možemo mu predati objekt ulaza (inputElement). Slika 4.4. prikazuje put ulaznog objekta kroz JFace dijelove. Nakon što predamo ulazni objekt prikazu on se oblikuje u pružatelju sadržaja i takva se struktura predaje prikazu. Prikaz na temelju pružatelja tekstualnoga prikaza na odgovarajući način prikazuje sadržaj u grafičkom elementu. U slučaju da je omogućeno mijenjanje objekta preko uređivača ćelije, na grafički element se smještaju dodatne kontrole koje omogućuju unos i promjenu sadržaja elementa.

## 4.5. Eclipse Forms UI

### 4.5.1. RCP

Eclipse platforma je dizajnirana da služi kao otvorena platforma za izgradnju alata, arhitektura je tako osmišljena da se pomoću komponenta koje su namijenjena za nadogradnju može izgraditi bilo kakva samostalna klijentska aplikacija. Najmanji skup potreban za izgradnju bogate klijentske aplikacije naziva se bogatom klijentskom platformom – RCP (engl. *Rich Client Platform*).

Bogate aplikacije i dalje koriste Eclipse-ov model dinamičkog učitavanja dodataka, grafičko sučelje koje je izgrađeno pomoću Eclipse alata za izgradnju grafičkog sučelja, proširenja i već postojećim dodatke. Za razliku od razvijanja dodataka za Eclipse platformu, kod razvoja za Eclipse RCP programer je sam zadužen za prilagodbu Workbench dodatka i definiranja aplikacije koja sad postaje samostalna i ne ovisi o Eclipse platformi.

Glavna razlika između bogate klijentske aplikacije i Eclipse platforme je da kod RCP moramo odrediti koji će razred biti glavni razred aplikacije koji će se pokrenuti.

### 4.5.2. Nastanak Forms

Glavna prednost Eclipse platforme je da aplikacije izgrađene na njoj imaju jedinstven izgled i osjećaj korištenja (engl. *native look and feel*) kao sve druge aplikacije koje se izvršavaju na pokrenutom operacijskom sustavu. Zahvaljujući SWT svi meni, prozori, stabla, tabele, gumbi i ostali grafički elementi imaju isti izgled.

Dolaskom Eclipse verzija 3.x uvodi se koncept bogate klijentske platforme RCP, koji značajno proširuje funkcionalnost Eclipse platforme, pruža mogućnost iskorištavanja dijelova Eclipse u svrhe izgradnje aplikacija, koje nemaju nikakvu vezu s integriranim razvojnim okruženjem s čime se pojam Eclipse obično povezuje.

Za razvoj aplikacija grafički elementi SWT, nisu pružali dovoljno tako je od verzije Eclipse-a 3.0 dostupna alternativa pod nazivom Eclipse Forms UI koja pruža API za izgradnju grafički privlačnih i prilagodljivih elemenata za izgradnju grafičkog sučelja.

Eclipse Forms UI razvijen je puno prije RCP koncepta, kao riješenje na konkretan problem koji su Eclipse programeri morali riješiti. Programeri PDE-a radili su na dodatku za uređivanje manifesta dodatka. Pri razvoju dodatka nisu bili zadovoljni idejom razvoja

teksualnog uređivača zbog neprivlačnosti koji bi takav uređivač pružao programeru, čak i s bojanjem sintakse uređivanje manifesta bio je dosta zahtjevan i neprivlačan posao. Manifest datoteka bila je XML datoteka čije uređivanje čak i uz razne čarobnjake i automatsko nadopunjavanje (engl. *code completion*) zahtjevan posao. Glavna razlika u odnosu na uređivač Java koda jest da sama sintaksa XML manifest datoteke nije važna, već samo podaci koji taj XML predstavlja su bitni.

Ideja vodilja pri izradi sučelja za uređivanje manifesta bila je analogija XML manifesta i HTML. HTML i XML oboje su jezici obilježavanja, kao takvi su vrlo slični. Izvorni kod oba jezika sastoji se od ugnježdenih elemenata i pripadajući im atributima. Iako kod oba vidimo njihovu sintaksnu strukturu, teško je vizualizirati što ona predstavlja. WYSIWYG (engl. *What you see, is what you get*) alati pružaju drugačiji pogled na isti sadržaj pomoću kojeg su tada vidljive tablice, boje, tekst, slike i ostali elementi, rad na ovakav način pruža veću produktivnost. Analogija je primijenjena i na manifest uređivač, iako je još uvijek postojalo pitanje kakva bi trebala biti alternativna prezentacija XML manifesta. Odlučeno je da će se svaki dio manifesta prikazivati na način koji je najbolje odgovara sadržaju koji predstavlja, tako su osnovne informacije koje treba uređivati smještene u tekst elemente, dok su hijerarhijske strukture smještene u stabla. Informacije su prikazane na različitim stranama uređivača i u različitim sekcijama ovisno o njihovim značenjima a ne mjestu gdje se nalaze u XML izvornom kodu. Strane uređivača su zadržale osjećaj uređivanja dokumenta, a opet omogućuju miješanje hiperlinkova, slika i teksta na jednom mjestu.

Nakon što su programeri riješili problem PDE mafinest uređivača, ostali programeri željeli su iste mogućnosti za rješavanje ostalih problema, tako da je s verzijom Eclipse-a 3.0 postali dostupni Forms UI API.

### 4.5.3. Zadaća Forms UI-a

Eclipse Forms UI je opcionalni dodataka za RCP bazirana na SWT i JFace koji omogućuje izgradnju web-olikog sučelja u bilo kojoj kategoriji Eclipse korisničkog sučelja, tako da kontrolira stilove elemenata, boje fontove grafičkih elemenata kako bi se postigao web-oliki izgled i ponašanje elemenata. Zadaća Eclipse Forms UI-a nije bila zamijeniti SWT i JFace već nadopuniti njihovu funkcionalnost.

Sam Forms UI dodaje samo nekoliko probranih grafičkih elemenata, rasporeda elemenata i pomoćnih razreda kako bi ostvarili svoju zadaću, a kako su zapravo proširenje SWT-a i JFace dostupni su na svim platforma gdje i SWT.

Kako bi postigli web-olik izgled sučelja Forms UI uvodi slijedeće koncepte:

- koncept "forme" kao elementa koji se smješta na mjesta za prikaz ili obradu podataka kao što su (pogledi, uređivač), forma kao spremnik (engl. *container*) za ostale grafičke elemente koji čine jednu cjelinu, a kao takvi trebaju imati zajednička grafička i ponašajna svojstva
- alat (engl. *toolkit*) koji kontrolira boje, hiperlinkove i ostale aspekte forme, a služi kao tvornica (engl. *Factory pattern*) za SWT grafičke elemente koji poprimaju svojstva forme
- dva nova načina raspoređivanja (engl. *layout*) čiji algoritam razmještanja grafičkih elemenata na površinu oponaša algoritam smještanja tablica HTML-a

- skup novih grafičkih kontrola dizajniranih da se uklope u form-u kao što su hiperlinkovi, grafički hiperlinkovi, klizne prostore (engl. *scrollable composite*) i odlomke (engl. *section*)
- višestrane uređivače gdje je svaka strana forma (npr. PDE manifest uređivač)

Grafički elementi Forms UI-a zapravo su SWT grafički elementi prilagođenog izgleda, oni se kreiraju pomoću posebnog razreda tvornice (FormToolkit), instanca tog razreda brine se o izgledu svih elementa generiranih pomoću njega.

### 4.5.4. Napredni Forms UI elementi

Gospodar/Detalji (engl. *Master/Details*) je česta šablona (engl. *pattern*) programiranja korisničkih sučelja. Sastoji se od liste ili stabla (gospodara) i skupa svojstva (detalja) koja se mijenjaju promjenom elementa liste ili stabla ( gospodara).

Kako bi se olakšalo korištenje ove šablone Eclipse Forms pruža korisne gradivne elemente s slijedećim svojstvima:

- dio s detaljima treba se implementirati, dok se gospodar dio mora naslijediti i implementirati potrebne metode
- gospodar i detalji oboje nasljeđuju sash form tako da se njihov međusobni odnos širina može jenjati pomicanjem linije između njih
- pošto gospodar i detalji nasljeđuju sash forum, mogu postaviti horizontalno ili vertikalno.

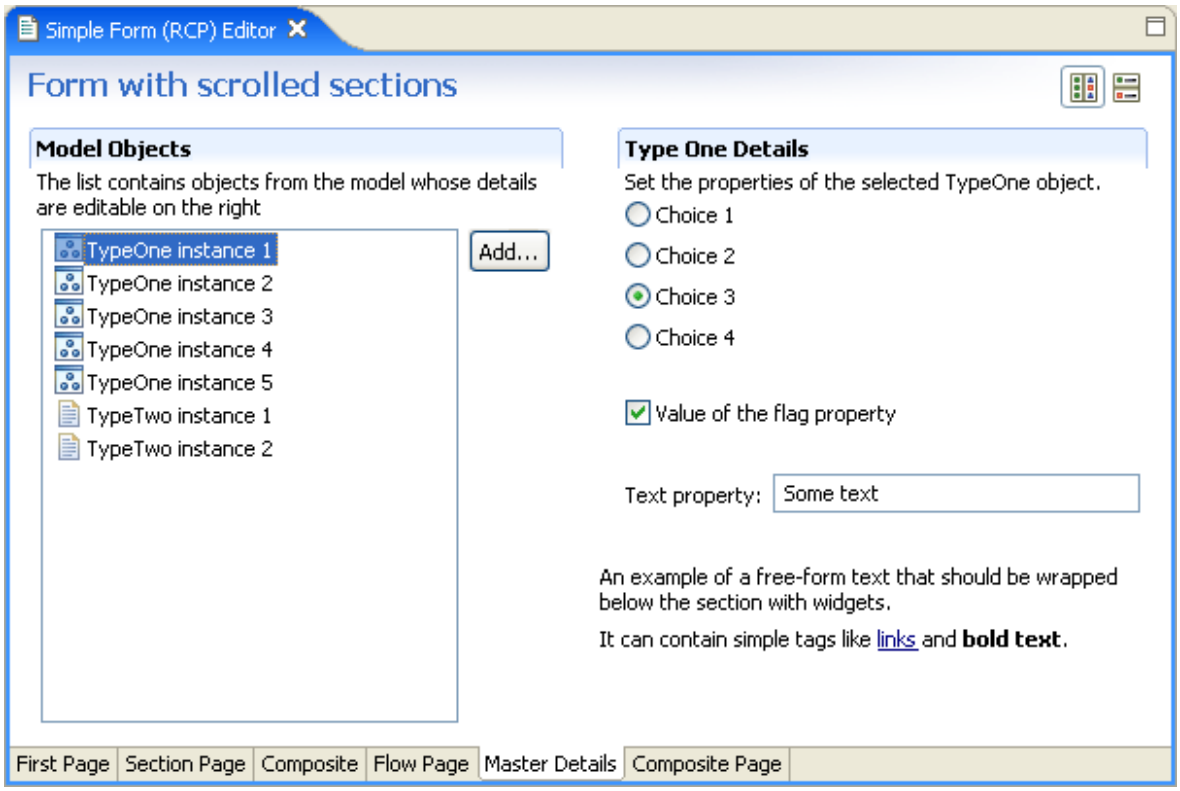
Osnovna ideja ove programske šablone je napraviti stablo ili tablicu koje nakon što se označi jedan od elemenata stabla ili tablice šalje se obavijest upravljanoj formi (engl. *managed form*), koja detektira promjenu, dok se označeni objekt prosljeđuje detalju koji je registriran na razred objekta koji aktivira prikaz detalja. Na aktivaciju detalja, stranica detalja se mijenja i u prostoru za detalje otvara se stranica s pripadajućim detaljima.

Korištenje ove programske šablone zahtjeva slijedeće:

- Kreirati razred gospodara tako da naslijedi `MasterDetailsBlock`, implementirati metodu `createMasterPart()` u kojoj se kreiraju svi grafički elementi koje će se nalaziti na gospodar dijelu, ovdje se kreira i stablo ili tablica koja u kombinaciji s JFace Viewer razredom služi kao izvor objekata na koje reagiraju detalji. Kada je došlo do promjene označenoga objekta, potrebno je dojaviti pozivom metode `fireSelectionChanged()` kojoj se predaje označeni objekt
- potrebno je za svaki razred koji će biti instanciran i smješten u stablo ili tablicu napraviti detalje, detalji su forma koja implementira sučelje `IDetailsPage`, te metodu `createContents()` u kojoj se izgrađuje grafičko sučelje za prikaz objekta koje prikazuje detalj, objekt se prima preko metode `selectionChanged(IFormPart part, ISelection selection)` gdje se iz parametra `selection` izdvoji objekt koji se prikazuje
- nakon što su napravljeni svi detalji, u gospodar razredu u metodi `registerPages(DetailsPart part)` prijavljuje na koji razred se aktivira

koji detalj, npr. `part.registerPage(Type.class, new TypeDetails());` ovdje se razred `Type` prijavljuje na novu instancu detalja `TypeDetails`, važno je napomenuti da se stvara samo jedna instanca detalja, neovisno koliko se promjena selekcije dogodilo.

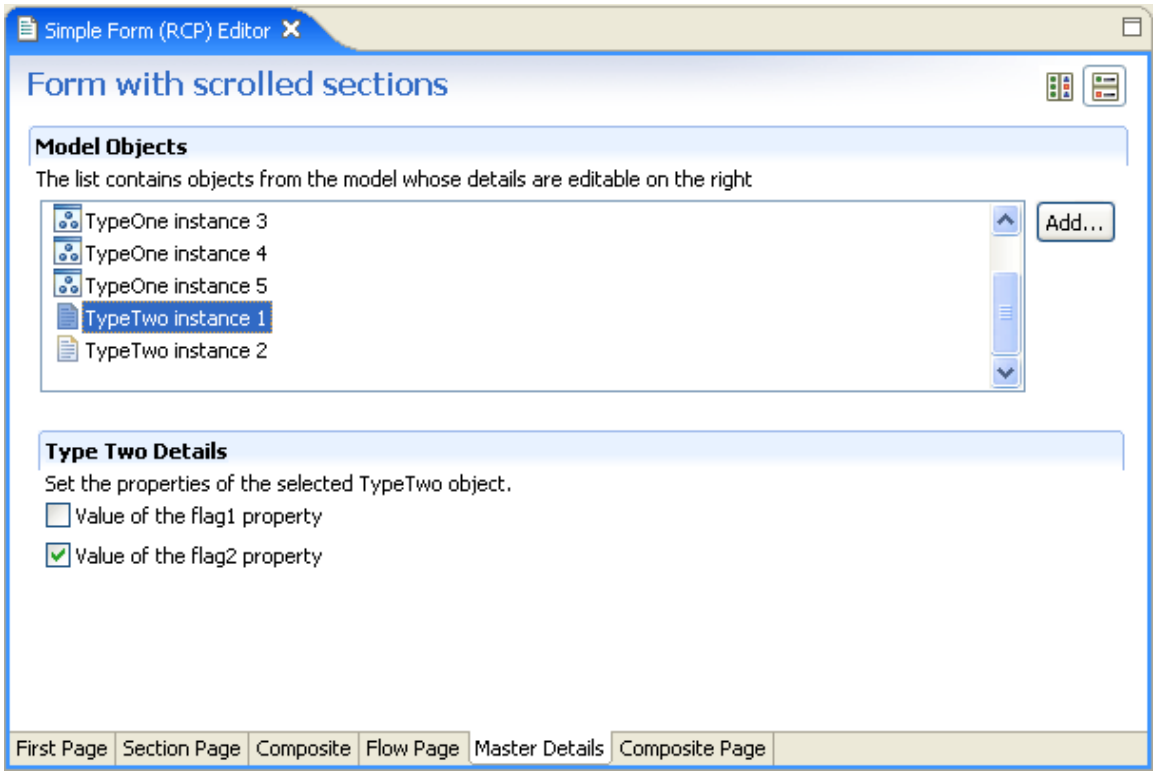
Neka imamo listu elemenata koju smještamo tablicu s jednim retkom, u toj listi mogu se naći dva tipa objekata (TypeOne, TypeTwo). Detalji i gospodar mogu se smjestit na dva načina horizontalno (slika 4.6.) ili vertikalno (slika 4.5.). Podjela se može i dinamički mijenjati prema potrebi gumbima u gornjem desnom kutu. U slučaju da ne želimo imati gumbe za promjenu podjele gospodara/detalja gumbi ne moraju biti prikazani, tada se ne može promijeniti raspored gospodar/detalja.




Slika 4.5. Označena instanaca objekta TypeOne u gospodar dijelu, s vertikalnom podjelom

Označivanjem instance objekta TypeOne u dijelu gospodar, otvara se odgovarajuća strana detalja s TypeOneDetails (slika 4.5.). Strana s detaljima sadrži odgovarajuće grafičke elemente pomoću kojih se mijenjaju svojstva označenoga objekta.

Promjenom orijentacije gospodar/detalja i promjenom odabira na objekt instance tipa TypeTwo dobivamo stanje kako je prikazano na slici 4.6. Iako se potpuno promijenio izgled prikaza, zapravo su se dogodile dvije stvari: promijenio se odnos gospodara/detalja iz vertikalne podjele u horizontalnu i promijenila se stranica detalja. Na ovaj način korisnik ili programer može za različite podatke podesiti izgled kako mu najbolje odgovara.



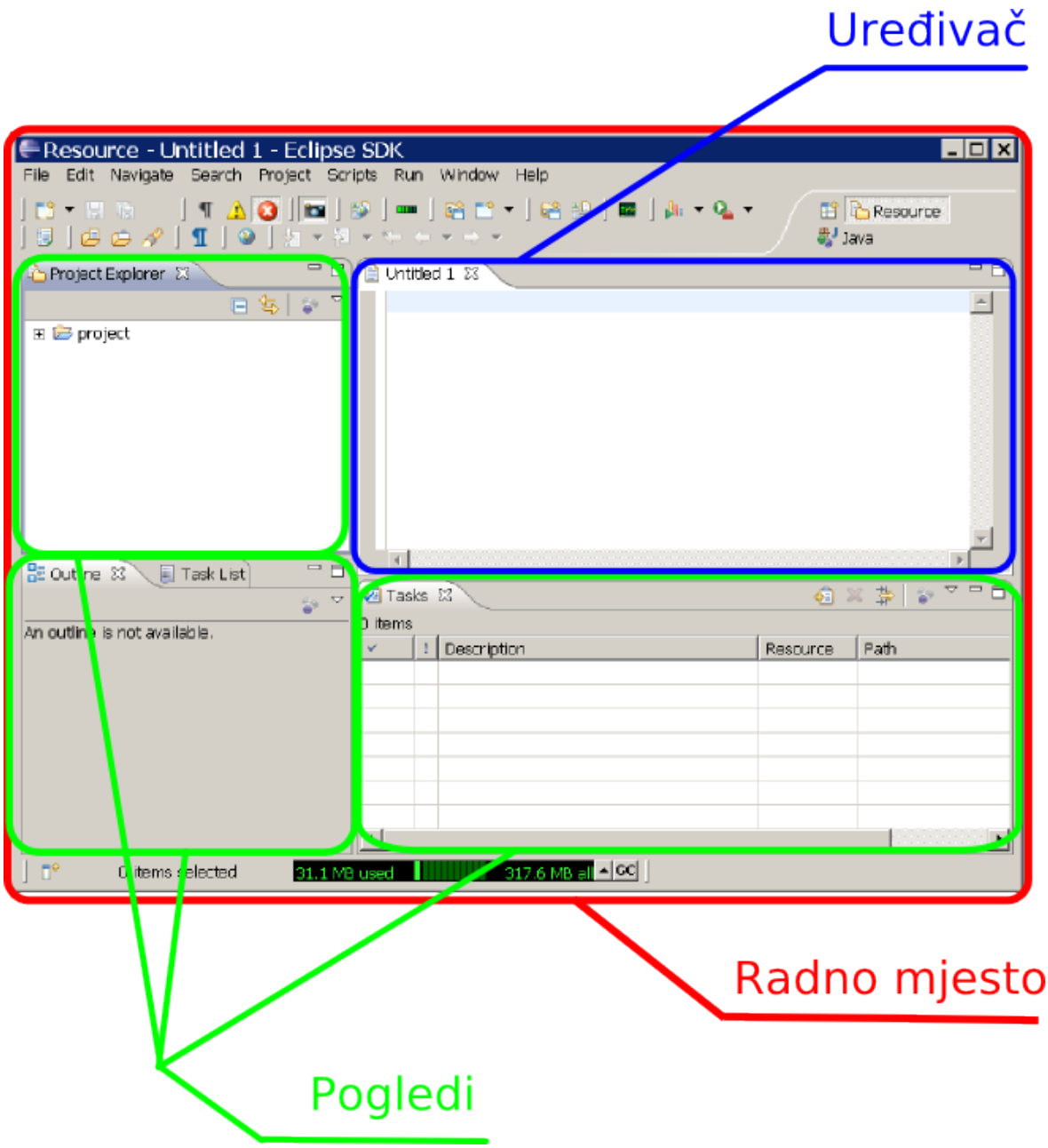

Slika 4.6: Označena instanca objekta TypeTwo u gospodar dijelu, s horizontalnom podjelom

Implementacija ove izrazito moćne i korisne paradigme jednostavna je i omogućuje modularan razvoj. Svaki detalj može se razvijati zasebno i ovisno o objektu koji se želi predstaviti, a konačni izgled djeluje kao jedna cjelina.

Kako bi implementirali gospodar/detalji element prvo je potrebno definirati podatkovni model i sve njegove potrebne razrede. Zatim se za svaki razred potrebno je napisati detalj koji implementira `IDetailsPage` sučelje. Svaki detalj implementira potrebne grafičke elemente za prikaz i obradu razreda na koji će biti registriran. Svaki detalj kao ulaz dobiva objekt razreda za koji je registriran, a objekt koji dobiva ista je instanca koju mu je predao gospodar, na taj se način cijelo vrijeme predaje isti objekt, a ne njegove kopije.

Nakon što je definiran podatkovni model i svi detalji za sve razrede koje je potrebno prikazati, oni se moraju registrirati u gospodaru u metodi `registerPages`, tako da se poveže tip razreda i objekt detalja.

## 4.6. Elementi Eclipse radnog mjesta




Slika 4.7. Prikaz radnog mjesta

Eclipse radno mjesto unificirano je grafičko sučelje koje pruža Workbench dodatak. Workbench dodatak omogućuje svoje proširenje i na taj način raspoređuje poglede i uređivače na radni prostor (slika 4.7).

### 4.6.1. Eclipse uređivač i pogledi

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

Pogledi nude informacije o objektima na kojima korisnik radi u radnom prostoru, obično se 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, njegovo je 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:

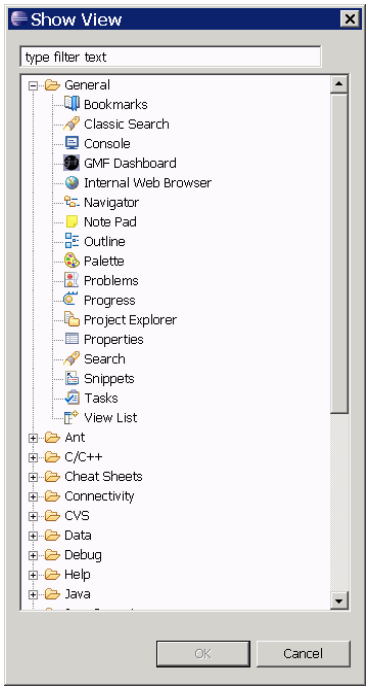
- nasljeđivanje odgovarajućeg nadrazreda (ViewPart ili EditorPart)
- implementacija metode `createPartControl` koja stvara SWT grafičke elemente koji predstavljaju vizualne komponente sučelja, ovdje se vrše sva ostala alokacija potrebnih resursa
- implementacija metode `setFocus` koja definira ponašanje 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)

### 4.6.2. Eclipse 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 pa se njihova lokacija ne može promijeniti, dok su pogledi razmješteni oko uređivača, a njihova se 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 4.8.) možemo odabrati koji pogled želimo aktivirati. Radi lakšeg snalaženja pogledi su grupirani u kategorije.




Slika 4.8. Grupiranje pogleda u kategorije - dialog Show View

Dodavanje novog pogleda sastoji se od tri koraka:

- definiranje kategorije kuda pripada pogled u manifestu dodatka
- definiranje pogleda u manifestu dodataka
- definiranje 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 4.3. za kategoriju je potrebno dodati dvije stvari

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

XML 4.3: Primjer dodavanja pogleda i kategorije pogleda

```
<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>
```

Kako bi definirali pogled u manifestu dodataka potrebno je proširiti točku proširenja `org.eclipse.ui.views` dodavanjem novog XML pod elementa view u `plugin.xml`. Kako XML 4.3. prikazuje, potrebno je dodati XML element *view* s sljedećim obaveznim atributima:

- *id*: jedinstveni identifikator pogleda
- *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 i koja 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. Kako bi implementirali ovo sučelje potrebno je ostvariti slijedeće metode:

- createPartControl(Composite): ova metoda stvara SWT elemente korisničkog sučelja i 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 4.1.

*Java 4.1: Primjer jednostavnog pogleda, koji ne sadrži nikakve grafičke elemente*

```
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
          }
}
```

### 4.6.3. Eclipse 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`. Na taj način je veći dio uobičajenog ponašanja naslijeđen i ne treba ga ponavljati.

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

Kako 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, primjena više stranica prikazana je u primjerom Java 4.2.
- *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 datoteke 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 registria sa `plugin.xml, fragment.xml` i `manifest.mf` datotekama.
- *launcher*: razred koji implementira `org.eclipse.ui.IEditorLauncher` sučelje, koje ostvaruje otvaranje eksternog 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 i 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 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 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 kako 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, u ovoj se metodi 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 za 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 4.2 i Java 4.3 odsječcima. Java 4.2. prikazuje specifične detalje višestrani uređivač kao što su manipulacije stranicama, dok Java 4.3. odsječak prikazuje elemente koji se odnose općenito na uređivače na spremanje dokumenata.

*Java 4.2: Primjer višestranog uređivača manipulacija stranicama*

```
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) {
```

*Java 4.3: Primjer višestranog uređivača definicija spremanja dokumenta*

```
        /**
         * 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;
        }


}
```

# 5. Korisničko sučelje za uređivanja Lusca testova

Korisničko sučelje za izradu Lusca ispitnih paketa je kao Eclipse dodataka, te je na taj način omogućena integracija s ostalim alatima koji već postoje za Eclipse platformu.

## 5.1. Arhitektura Eclipse dodatka

Lusca dizajner ispitnih paketa zamišljen je kao Eclipse dodatak, čija je osnovna namjena uređivanje datoteke *`test_paket.ltd`*. Kao takav osnova dodatka je proširiti uređivač tako da otvara datoteke s ekstenzijom *`ltd`*. Uređivač je izgrađen model-pogled-upravitelj paradigmom MVC, koja se temelji na tome da se podaci prikaz i manipulacija nad podacima razdvoje u tri različita elementa, podaci-model, prikaz-pogled i manipulacija-kontroler. Uređivač čita XML datoteku i na temelju XML objektnog modela izgradi interno stablo objekata nad kojima vrši promjene, nakon obavljenog posla stablo objekata se sprema u XML. Učitavanje XML-a u objektni model vrši se deserijalizacijom XML, a spremanje se vrši serijalizacijom objektnog stabla u XML, za serijalizaciju/deserijalizaciju koristi se Simple XML Serialization Framework. Nakon što je dobiveno objektno stablo koje oponaša definirani Lusca XML imamo vršni objekt kojeg vežemo na grafičke elemente sučelja.

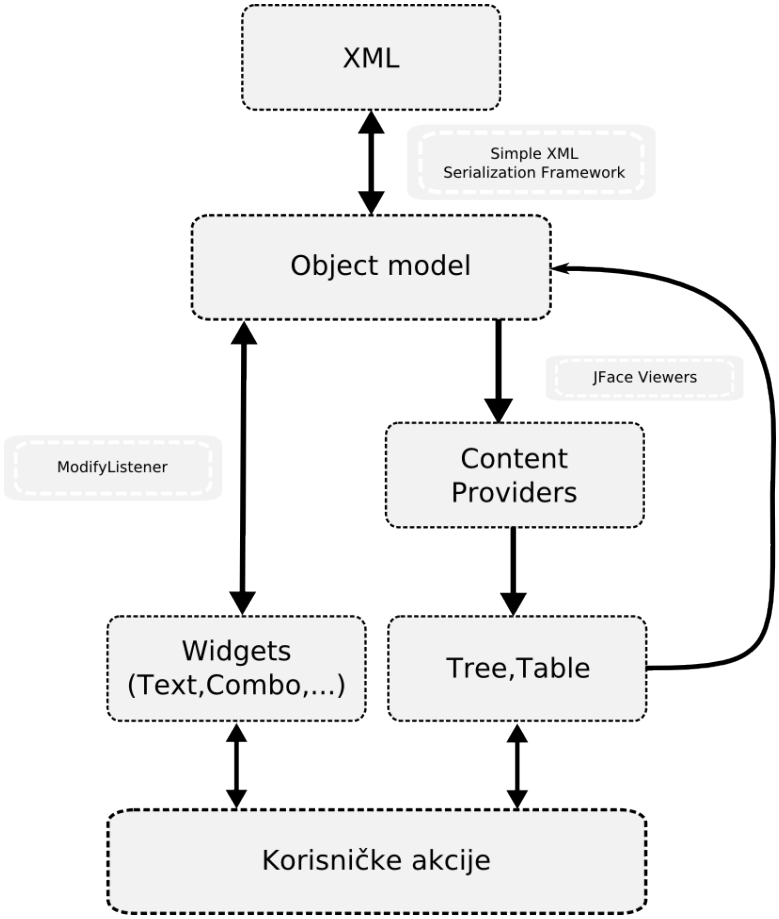
Vezanje na grafičke elemente vršimo na dva načina, direktno i indirektno. Direktno vezanje se vrši preko posebnog objekta čiji razred proširuje razred `ModifyListner`, koji pri promjeni sadržaja grafičkog elementa za unos (`Text`, `Combo`) aktivira svoju metodu `modifyText` u kojoj se sadržaj iz grafičkih elemenata stavlja u odgovarajući element objektnog stabla, a uređivač se označava promijenjenim (engl. *dirty*). Grafički elementi se popunjaju sadržajem iz objektnog stabla. Indirektno vezivanje vrši se preko JFace Viewer-a koji prima ulazni objekt (engl. *Input object*), te preko CellEditora mijenja ulazni objekt.

Proslijeđivanje podatak između pojedenih dijelova uređivača prikazano je slikom 5.1. Uređivač je podijeljeni u tri sloja:

- sloj XML
- podatkovni model
- korisničko sučelje

Sloj XML-a je datoteka XML koja sadrži Lusca ispitnu deklaraciju, zadaća XML je omogućiti ručno uređivanje testova. XML se serijalizira u stablo objekata koje odgovara XML stablu.

Uređivač ima objekt koji predstavlja Lusca ispitni paket, pa iz njega vadi informacije koje su potrebne kako bi se ispunio određeni dio korisničkog sučelja. Pri promjeni sadržaja u grafičkom korisničkom sučelju, sadržaj se preslikava u instancu objekta uređivača. Jedna instanca objekta omogućuje jednostavnu serijalizaciju u XML, i svim dijelovima uređivača prosljeđuje se ta instanca objekta pa ako bilo koji dio uređivača promijeni instancu objekta, sve promjene su vidljive u ostalim dijelovima uređivača.




Slika 5.1. Arhitektura prijenosa podataka između pojedinih elemenata dodatka

## 5.2. Organizacija koda

Kod je organiziran u java pakete radi bolje organizacije i čitkosti. Paketi su imenovani prema namjeni, a osnovni Java paket je `hr.fer.zemris.lusca.testdesigner`.

Razredi koji ostvaruju podatkovni model za uređivač smješteni su nekoliko osnovnih paketa :

`hr.fer.zemris.lusca.testdesigner.model`

`hr.fer.zemris.lusca.testdesigner.model.definitions`

`hr.fer.zemris.lusca.testdesigner.model.test`

`hr.fer.zemris.lusca.testdesigner.model.types`

Paket `definitions` sadrži razrede koji pripadaju u Lusca definicije, dok `test` sadrži razrede koji povezuju Lusca test elemente. Paket `types` sadrži pobrojane (Enum) razrede koji sadrže vrijednosti koje se mogu pojavljivati u nekom od razreda modela

Grafičko korisničko sučelje uređivača podijelio je isto u nekoliko paketa:

`hr.fer.zemris.lusca.testdesigner.editors`

`hr.fer.zemris.lusca.testdesigner.editors.definitions`

`hr.fer.zemris.lusca.testdesigner.editors.test`

Osnovni paket `editors` sadrži elemente potrebne za manipulaciju višestranim uređivačem, dok paketi `definitions` i `test` sadrže elemente za izgradnju grafičkog korisničkog sučelja vezani za dijelove definicija i testova.

Osim uređivača postoji i nekoliko razreda koji implementiraju čarobnjake za izradu novog projekta i za izvoz ispitnog paketa u kompresirani ispitni paket, oni su smješteni u pakete:

`hr.fer.zemris.lusca.testdesigner.wizards`

`hr.fer.zemris.lusca.testdesigner.wizards.export`

Razredi čija namjena je bilježenje korisničkih aktivnosti u svrhu određivanje kvalitete grafičkog korisnčkog sučelja smješteni su u paketu:

`hr.fer.zemris.lusca.testdesigner.usability`

## 5.3. Ostvarenje grafičkog korisničkog sučelja ltd uređivača

Uređivač je ostvaren prema uzoru, na Eclipse PDE manfiest editor, koristeći Eclipse Forms UI. Ideja izrade grafičkog korisničkog sučelja bila je da korisniku sintaksa XML ne bude važna, već sadržaj koji se nalazi u XML, te se na taj način gradi sučelje koje korisniku omogućuje uređivanje sadržaja bez da se opterećuje kako će se sadržaj smjestiti u XML elemente.

Uređivač je napravljen je kao višestrani uređivač s četiri strane:

- Overview
- Definitions
- Tests
- XML Source

Osnovni razred uređivača je `MultiPageTestEditor` on sadrži instancu objekta svake pojedine stranice i instancu objekta ispitnog paketa. Ovaj razred implementira svu logiku uređivača i svakoj strani prosljeđuje instancu objekta ispitnog paketa. Sve stranice, osim XML Source, implementiranju se u zasebnim razredima, dok se za pregledm XML Soureca korsiti Eclpse XML editor ako postoji instaliran kao dodatak, a ako ne postoji tada se koristi običan Eclipse tekst uređivač.

Svaka stranica uređivača (osim XML Source) implementirati sučelje `ItestPackageManipulator (Java 5.1)`.

*Java 5.1: Definicija sučelja ITestPackageManipulator*

```
public interface ITestPackageManipulator {

        public void setTestPackage(TestPackage tp, boolean
withRefresh);

        public void setModelToWidgets();

        public void getModelFromWidgets();

}
```

Metoda `setTestPackage()` služi za postavljanje objekta ispitnog paketa kao referentnog u razred, a parametar `withRefresh` označava da li se mora osvježiti sadržaj u grafičkim elementima.

Metoda `setModelToWidgets()` postavlja sadržaj modela, odnosno objekta ispitnog paketa u grafičke elemente.

Metoda `getModelFromWidgetds()` iz sadržaj grafičkih elemenata postavlja sadržaj objekta ispitni paket.

Stranice Definitions i Test koriste paradigmu Master/Details Eclipse Forms-a.

Definitions stranica sadrži blok gospodar u kojem se nalazi jedno stablo koje sadrži elemente definicije. Za prikaz stabla koristi se JFace TreeViewer koji kao ulaz prima definicije iz glavnog objekta Test paketa. Detalji se registriraju za svaki tip objekata koji se nalazi u definicijama. Svi detalji naslijeđuju razred `DetailsBase`, ovaj razred implementira osnovnu funkcionalnost koju moraju imati svi detalji, kao što su označavanje uređivača promijenjenim i osvježavanje sadržaja gospodara.

Tests sadrži isto blok gospodar, s razlikom da on ima četiri stabla, elementi svakog stabla prijavljeni su gospodar formi, pa se pri njihovom označavanju otvaraju njihovi detalji. Prvo stablo sadrži popis ispitnih slučaja, odabirom iz prvog stabla otvaraju se detalji ispitnog slučaja i popunjaju se ostala tri stabla. Preostala tri stabla predstavljaju dijelove izvođenja ispitnog slučaja pripreme, izvođenje i zaključak. Detalji se registriraju isto kao i za definicije i svi nasljeđuju razred `DetailsBase`.

## 5.4. Podsustav za bilježenje akcija korisnika

Kako bi se izmjerila efikasnost implementiranog korisničkog sustava, izrađen je podsustav mjerenja kvalitete korištenja sučelja. Podsustav za mjerenje kvalitete realiziran je kao sustav koji bilježi radnje koje vrši korisnik u sučelju.

Bilježe se slijedeće radnje:

- promjena strana uređivača
- uređivanje tekstualnih elemenata
- promjena označenoga elementa u stablu

- promjena forme
- dodavanje/brisanje elemenata

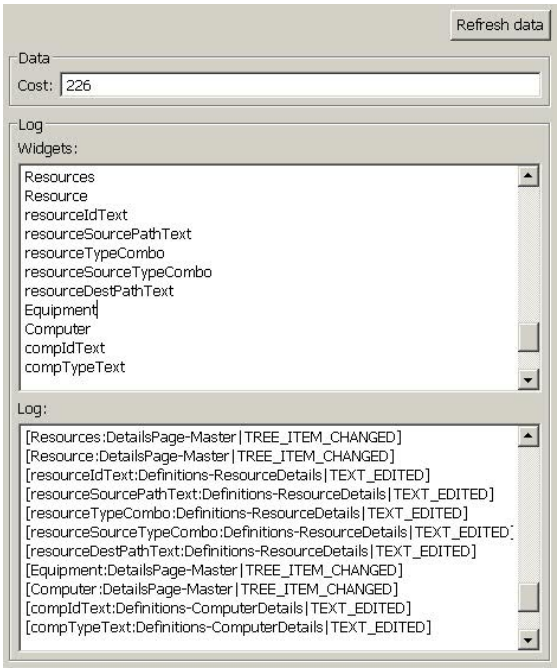
Zabilježene radnje koje korisnik vrši po sučelju koriste se za izračun efikasnosti korištenja sučelja. Na temelju promjena i unaprijed definiranih udaljenosti između pojedinih grafičkih elemenata računa se vrijednost kvalitete korištenja grafičkog sučelja. Svaka radnja ima unaprijed definiranu cijenu.

Algoritam računanja vrijednosti kvalitete sučelja:

```
zadnjiElement=log[0]
Za sve elemente iz loga:
        udaljenost=|zadnjiElement -element|
        vrjednost=vrijednost+cijenaIzvrseneAkcije+udaljenost
vrati vrijednost
```

Vrijednost koji algoritam daje je cijena koju je korisnik potrošio kako bi napravio test. Manja vrijednost za izradu ispitnog paketa znači da je korisnik bolje iskoristio korisničko sučelje, dok veća vrijednost znači lutanje po ekranu i dulje vrijeme da se postigne isti rezultat.

Podaci o korištenju sučelja prikazani mogu se vidjeti u posebnom pogledu *Usability view* (Slika 5.2).




Slika 5.2.. Prikaz podataka korištenja korisnčkog sučelja

Rezultati algoritma ovise o testu koji se uređuje. Izrada svakog testa može se provesti na optimalan način, tada je cijena korištenja sučelja najmanja. Optimalne vrijednosti razlikuju

se između svakog testa. Kako bi se dobiveni rezultati mogli uspoređivati potrebno je definirati referenti test. Kao referenti primjer uzet je test koji se nalazi u Dodatku A.

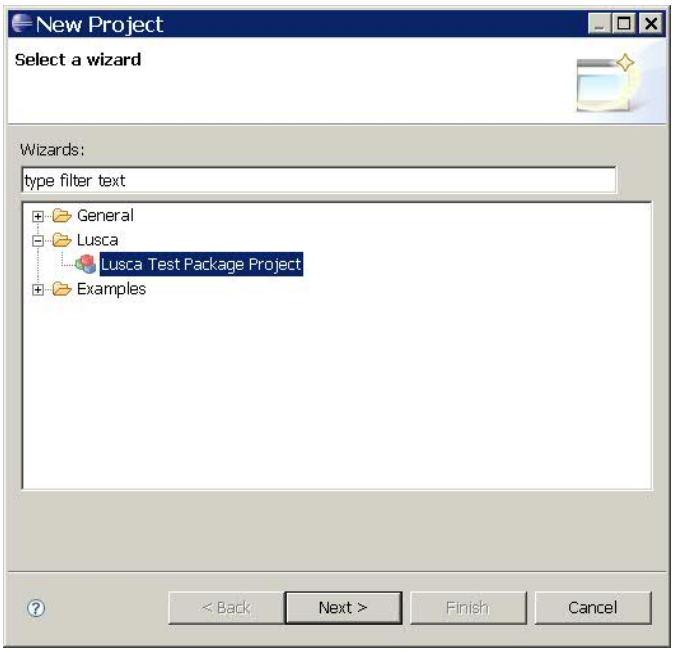
Za optimalnu izradu testa, cijena korištenja sučelja iznosi 639. Korisnici koji su se prvi put susreli sa sučeljem imali su cijenu korištenja 1000 i 1200, dok su iskusni korisnici imali cijenu između 650 i 700.

# 6. Upotreba sučelja za dizajniranje Lusca ispitnog paketa

U ovom poglavlju biti će prikazan postupak izrade jednog Lusca ispitnog paketa, ali će naglasak biti uređivanju na Lusca ispitne deklaracijei za koju izgrađen uređivač, izrada skripti neće biti opisivana jer su one drugačije za svaki programski jezik u kojem se mogu pisati.

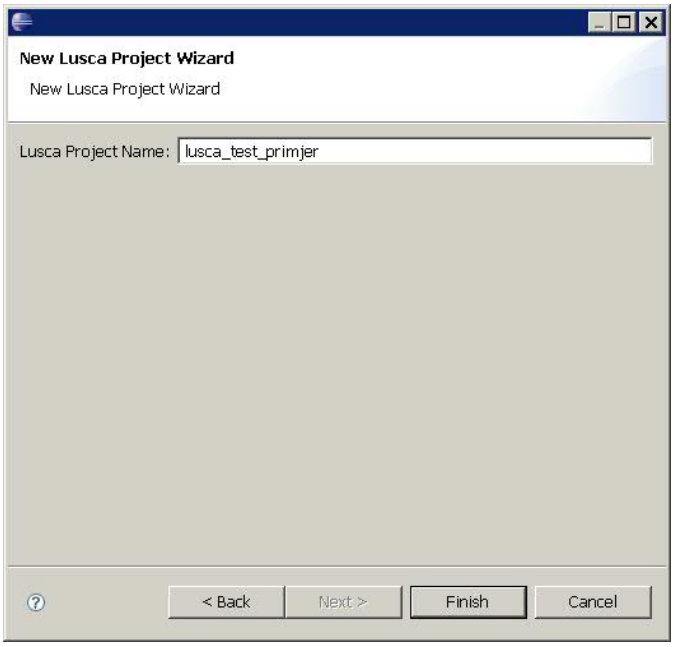
## 6.1. Novi Lusca ispitni paket

Izrada ispitnog paketa kreće generiranjem Eclipse Lusca Test Package projekta. Kako bi se to učinilo u Eclipse IDE u meni File potrebno je odabrati New-> Project. Otvoriti će se čarobnjak za odabir tipa projekta i u njemu je potrebno odabrati Lusca Test Package Project (slika 6.1.).



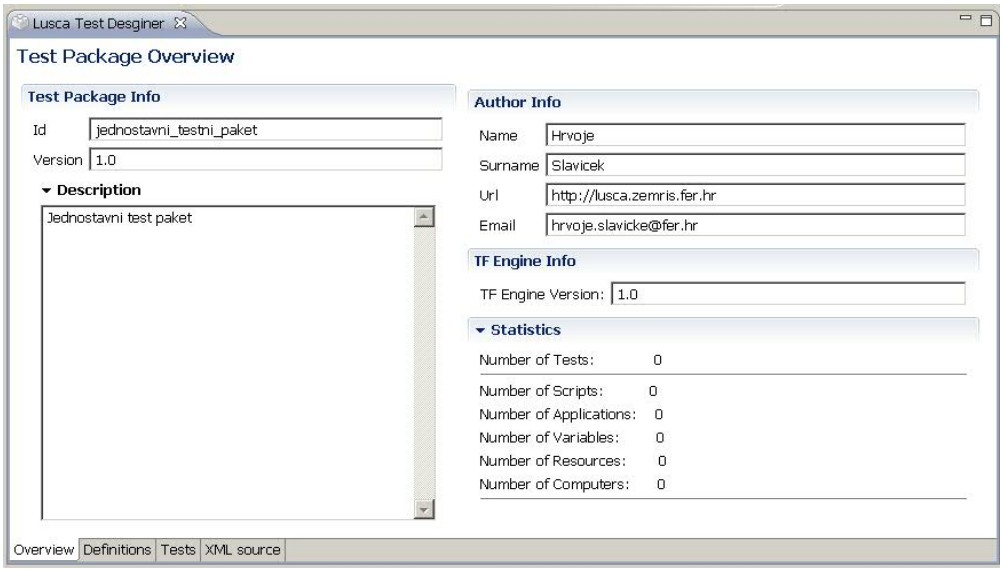

Slika 6.1: Odabir Lusca Test Package projekta

Nakon odabira projekta potrebno je nastaviti pritiskom na gumb Next. Otvoriti će se slijedeća strana čarobnjaka za izradu Lusca ispitnog paketa. Idući korak je definiranje imena projekta za novi Lusca ispitni paket (slika 6.2.). Nakon ovog koraka dovoljno je pritisnuti tipku Finish nakon čega će čarobnjak generirati potrebnu strukturu projekata s direktorijima conf i scripts i minimalnom Lusca ispitnom deklaracijom u datoteci test_package.ltd.




Slika 6.2: Ime novog Lusca ispitnog paketa

Prvi korak uređivanja ispitnog pakete kreće uređivanjem općih podataka ispitnog paketa (slika 6.3.). Osim uređivanja osnovnih podataka o ispitnom paketu koji zapravo reprezentiraju LuscaTestInfo, ovdje je i ispis statistika o cijelom ispitnom paketu, kao što su broj ispitnih slučaja, skripta, varijabli, aplikacija, resursi, računala.

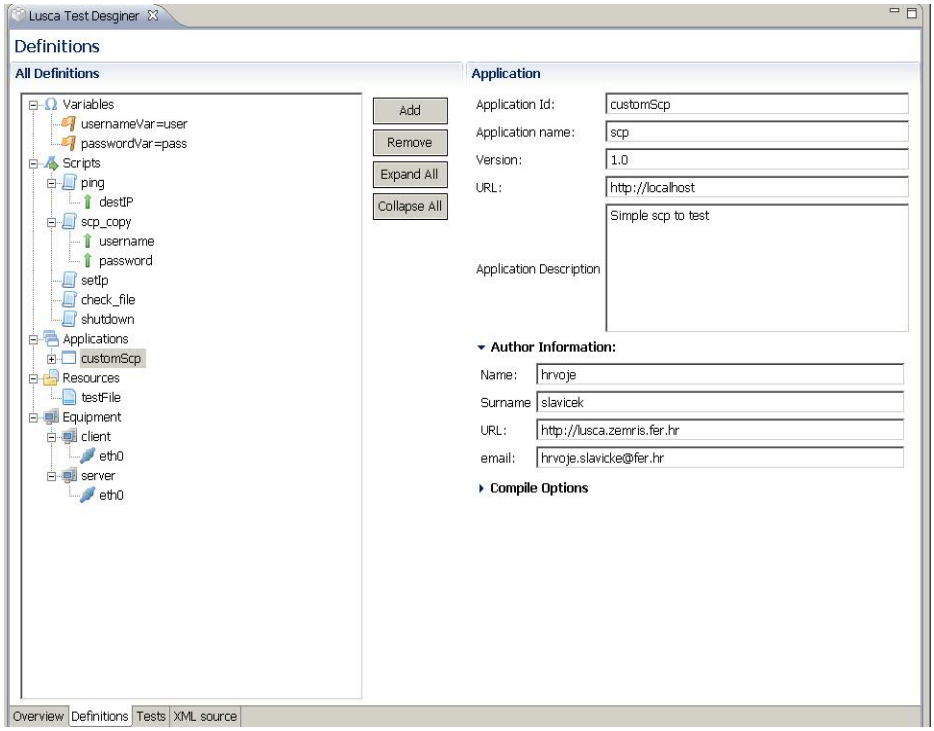



Slika 6.3. Osnovni podaci o ispitnom paketu

## 6.2. Uređivanje definicije Lusca ispitne deklaracije

Nakon unosa osnovnih podataka, slijedeći korak u izradi ispitnog paketa, je definiranje svih potrebnih elemenata za dizajn ispitnih slučaja koji će biti kasnije definirani. Definicije elemenata rade se na stranici Definitions (slika 6.4.).

Dodavanje jednog od elemenata (varijable, skripte resura, aplikacije ili računala) vrši se odabirom odgovarajućeg elementa u stablu gospodara i desnim klikom miša odabirom stavke New->New element. Novi element definicija biti će dodan na odgovarajuće mjesto u stablu s podrazumijevanim podacima. Uređivanjem određenog elementa radi se njegovim označivanjem, dok se s desne strane otvaraju svojstva tog elementa koja se mogu mijenjati. Kako svaki element ima drugačiji sadržaj i izgled svaki ekran biti će zasebno objašnjeni.

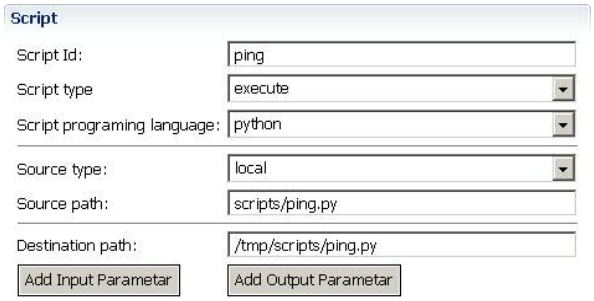



Slika 6.4. Stranica s definicijama varrijabli,skripta, aplikacija, resursa, računala

Nakon što je dodan novi element varijabla, uređivanje varijable svodi se na određivanje njezinoga identifikatora i vrijednosti (Slika 6.5.)


Slika 6.5. Svojstva varijabli

Uređivanje skripte (Slika 6.6.) zahtijeva određivanje nekoliko elemenata, a kako bi se pojednostavnio unos onih polja koja imaju samo određeni broj parametara koje moguće odabrati ta polja su izrađena kao padajući izbornici iz kojih se može označiti željeni element. Takva polja su script type, script programing language, source type, ostale parametre skripte treba ručno unijeti.




Slika 6.6. Uređivanje parametara skripte

Kako bi se dodao ulazni ili izlazni parametri skripte oni se dodaju gumbima Add Input Parametar i Add Output Parametar, pritiskom na jedan od ta dva gumba dodaje se ulazni ili izlazni parametar skripti. Ti parametri postaju vidljivi u stablu definicija, s podrazumijevanim vrijednostima, korisnik mijenja te vrijednosti tako da odabere željeni parametar ulazni (Slika 6.7.) ili izlazni (Slika 6.8.).


Slika 6.7: Uređivanje ulaznog parametra skripte

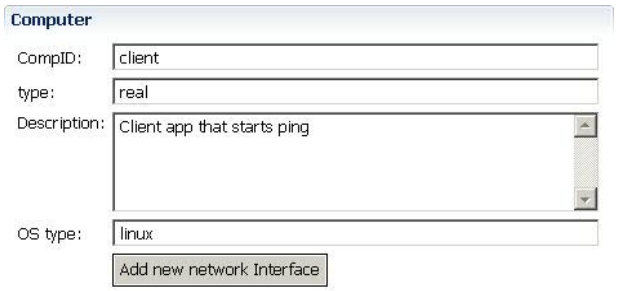

Slika 6.8: Uređivanje izlaznog parametra skripte

Jedan od najvažnijih elemenata definicija je aplikacija, njezina specifikacija obrađuje se dijelom prikazanim slikom 6.9. Kako bi se smanjila količina podataka na ekranu podaci o autoru aplikacije i podaci za prevođenje stavljeni su sekcije koje se mogu smanjiti. Većinu podataka potrebno je ručno unijeti.

Uređivanje statičnih sadržaja resursa prikazano je na slici 6.10. Odabir tipa resursa moguće je odabrati iz ponuđenih opcija (file,directory), osim toga ponuđeni je i tip izvorišne putanje, ostala polja korisnik mora ručno unijeti, s time da mora paziti da se ispravno unese izvršno ime.

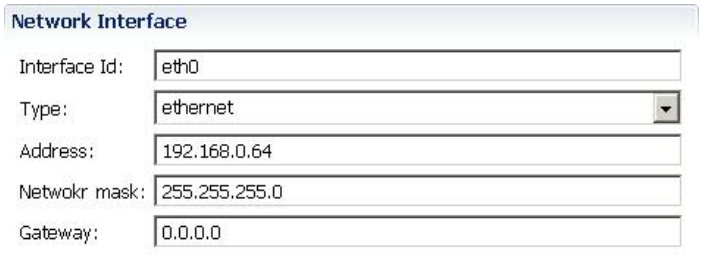

Slika 6.9. Uređivanje parametara aplikacije

Zadnja stvar koju treba definirati je računalo (slika 6.11.), ime, tip, opis i operacijski sustav. Ovdje se nalazi i gumb za dodavanje mrežne kartice čiji se parametri uređuju odabirom mrežne kartice u stablu definicija.




Slika 6.11. Uređivanje opisa računala

Parametri mrežnog sučelja uređuju se u posebnom dijelu (slika 6.12.). Potrebno je ručno unijeti sljedeće parametre se id (bit će predložen kao ethX), IP adresu mrežnog sučelja, mrežnu masku i pristupnik (engl. *gateway*).

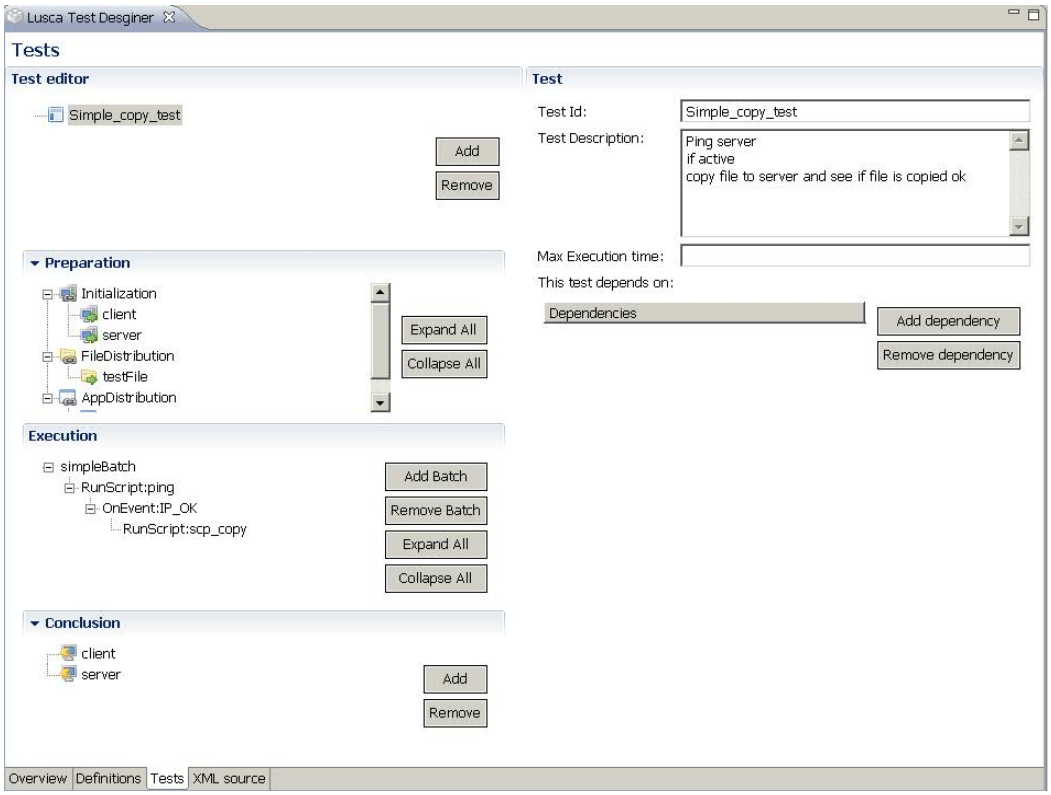



Slika 6.12. Uređivanje parametara mrežnog sučelja računala

## 6.3. Uređivanje ispitnih slučaja Lusca ispitne deklaracije

Nakon što su u definicijama određeni svi potrebni elementi za izradu ispitnih slučaja, oni se definiraju na stranici Tests. Prvi korak je dodavanje novog ispitnog slučaja tipkom Add, nakon čega će u listu ispitnih slučaja bit dodan novi ispitni slučaj. Slijedeći korak je u listi ispitnih slučaja odabrati ispitni slučaj i urediti njegova svojstva (Slika 6.13.). Odabirom ispitnog slučaja omogućeno je uređivanje tri njegova dijela:

- preparation
- executiion
- conclusion

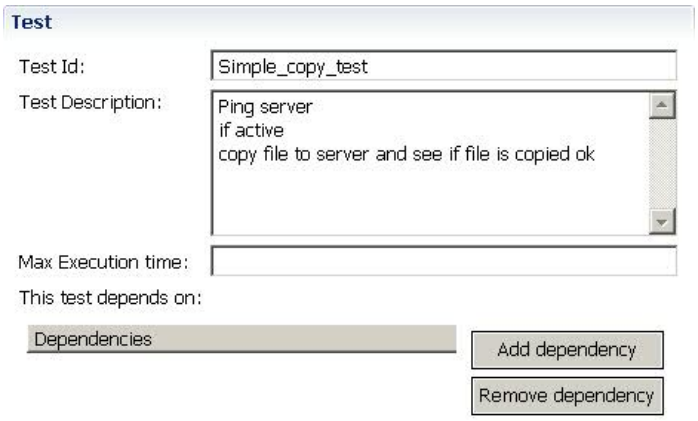



Slika 6.13. Uređivanje testova

Pošto se *preparation* i *conclusion* dijelovi najkraće uređuju, njih je moguće sakriti pritiskom na trokutić u naslovu sekcije. Svaki od navedenih dijelova ima svoje stablo elemenata, a u svako se stablo elementi dodaju na drugačiji način. U preparation dijelu elementi se dodaju desnim klikom miša New-> (Host for prepartion, File for distribution, Application for distribution).

U *execution* dijelu novi batch-evi se dodaju pritiskom na tipku Add Batch, a uklanjaju odabirom elementa kojeg želimo obrisati i pritiskom na tipku Remove Batch.

U *conclusion* dijelu dodavanje novih elemenata vrši se tipkom Add, a uklanjanje odabirom elementa kojeg želimo ukloniti i pritiskom na tipku Remove.




Slika 6.14. Opis i osnovni podaci o ispitnom slučaju

Prvi korak u uređivanju ispitnog slučaja je uređivanje njegovih informacija poput identifikatora, opisa, koje je maksimalno vrijeme izvođenja ispitnog slučaja (ako je ono potrebno specificirati). Za njega je potrebno dodati listu ispitnih slučaja o kojima izravno ovisi (Slika 6.14.).

Nakon što su određeni osnovni podaci o ispitnom slučaju, može se krenuti u uređivanje *preparation* dijela ispitnog slučaja. Računala za inicijalizaciju uređuju se tako da iz popisa definiranih računala odabere identifikator računala definiran u definicijama, a zatim se tom računalu pridruže sve skripte koje je potrebno pokrenuti (Slika 6.15.). Dodavanje skripte vrši se odabirom skripte iz popisa skripti i pritiskom na gumb Add Script. Ako se želi ukloniti skripta iz popisa, ona se odabere u popisu i pritisne na gumb Remove Script.


Slika 6.15. Pokretanje pripremnih skripti na računalu

Resursi za distribuciju uređuju slično, iz popisa resursa odabere se identifikator resursa, te mu se pridruže sva računala na koja resurs mora biti distribuiran (Slika 6.16.).

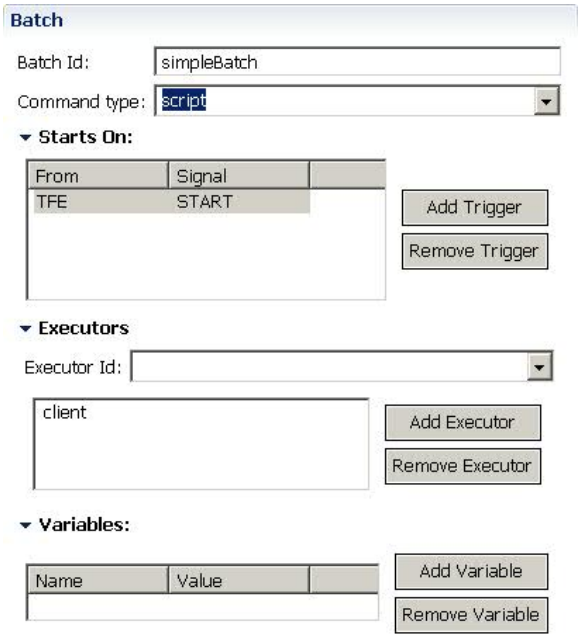

Slika 6.16. Distribucija resursa na računala

Aplikacija koja se distribuira uređuje se na isti način kao i resurs za distribuciju s razlikom da se stavlja identifikator aplikacije. Aplikaciji treba pridružiti sva računala na koja aplikacija treba biti distribuirana. Dodavanje računala vrši se odabirom računala iz popisa i pritiskom na gumb Add Host. Ako se želi ukloniti računalo iz popisa, ono se odabere u popisu i pritisne na gumb Remove Script.


Slika 6.17. Distribucija aplikacije na računala

Nakon što je završeno uređivanje *prepration* dijela, može se krenuti uređivati *excution* dio. Osnovni dio excution dijela je Batch (Slika 6.15.). Batch, kao i većina elemenata ima svoj identifikator, koji treba ručno unijeti, Command type opcija Lusce koja je ostavljena za buduća proširenja, a trenutno je jednino moguće staviti script (jedino što se može i odabrati iz popisa).

Batch je podijeljeni u tri dijela. StartsOn dio određuje na koje okidače se pokreće Batch.




Slika 6.18. Uređivanje Batch-a

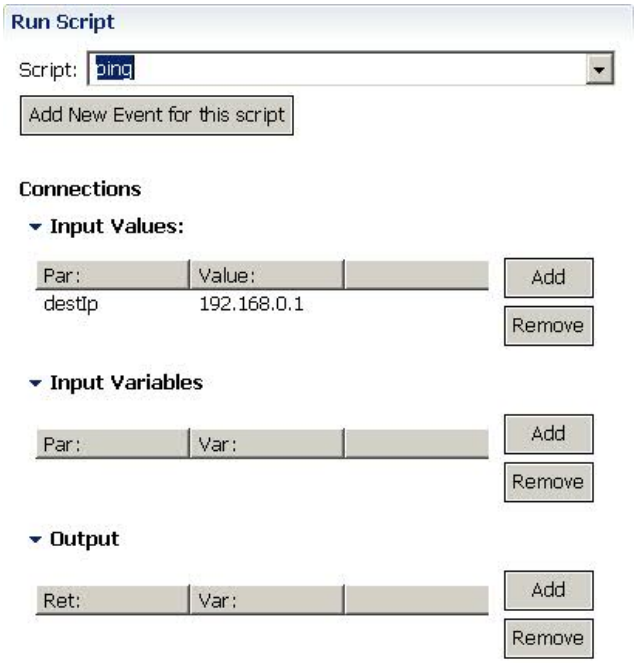
Okidača može biti više a oni se dodaju,uklanjaju gumbima s strane. Nakon što se okidač doda u listu potrebno je prilagoditi parametre okidača. Oni se uređuju direktno u listi, klikom na kolonu elementa, koju želimo izmijenit.

Slijedeći dio su Executors, odnosno računala na kojima se pokreće batch, računala se dodaju u popis odabirom identifikatora i pritiskom na gumb Add Executor, micanje se vrši odabirom u listi i pritiskom na gumb Remove Executor.

Treći dio Batch dijela su varijable koje se odnose na batch, one se dodaju gumbom Add Varijable, a uređuju se direktno u popisu, isto kao i okidači.

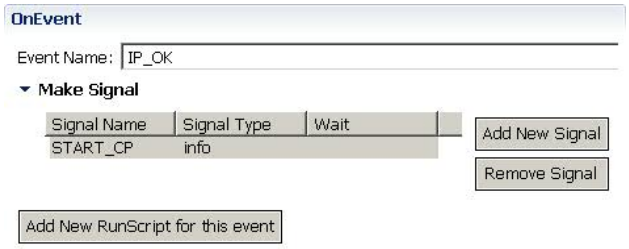
Batch ima pridružen element RunScript (Slika 6.19) koji služi za pokretanje skripte. Iz popisa skripata odabere se identifikator skripte koju želimo pokrenuti. Skripte koje se pokreću mogu imati ulazne i izlazne parametre koji se u ovom dijelu povezuju s vrijednostima ili varijabla.

Svakoj skripti može se dodati događaj, koji se poziva kad agent primi signal, događaji se dodaju u stablo izvođenja, a svojstva događaja uređuju se posebno (Slika 6.20.). Događaju se mora upisati njegov identifikator, odnosno ime signala na koji on reagira, a kako svaki signal može pokrenuti jedan ili više novih signala u događaju je moguće dodati nove signale, čiji se parametri uređuju direktno u tablici signal.




Slika 6.19: Pokretanje skirpti s povezivanjem na vrijednosti i varijable

Kako se na svaki događaj može pokrenut novu skriptu gumbom Add New RunScript for this event dodajem novo pokretanje skripte u stablo izvršavanja.




Slika 6.20. Parametri događaja

Uređivanje elemenata zaključka identično je uređivanju računala za inicijalizaciju (Slika 6.15.), razlika je samo u značenju tog elementa. Uloga skripata koje se pokreću u zaključku to jest pri završetku izvođenja ispitnog slučaja, je skupljanje rezultata i čišćenje ispitne okoline, dok je pri inicijalizaciji značenje pripremanje ispitne okoline.

Ukoliko je potrebno, moguće je uređivati direktno XML Lusca ispitne deklaracije (Slika 6.21.). Direktno uređivanje XML se ne preporuča. Kako bi uređivač mogao čitati i mijenjati Lusca ispitnu deklarciju ona mora biti uređen i valjan XML dokument. Ako se ručnim mijenjanjem XML-a Lusca ispitna deklaracija učini nevaljanom Lusca Test Designer neće moći pročitati XML model. Omogućavanje čitanja XML zahtjeva njegovo ručno uređivanje kako bi se učinio valjanim.


Slika 6.21. Direktno uređivanje XML-a Lusca test deklaracije

# 7. Zaključak

Dosadašnji razvoj Lusca ispitnog okvira pokazao se kao perspektivan, no isto tako uočeno je nekoliko kritičnih problema koje bi mogli smetati potencijalnim korisnicima ovog sustava. Jedna od kritičnih točaka jest složeni XML jezik kojim se definiraju testovi. Pojednostavljenjem XML-a izgubila bi se njegova izražajnost čime bi se smanjile mogućnosti Lusca ispitnog okruženja.

Rješenje ovog problema bila je izgradnja grafičkog korisničkog sučelja za uređivanje XML-a, čime je korisniku olakšano definiranje testova. Korištenjem grafičkog korisničkog sučelja povećava se preglednost i smanjuje se mogućnost pogreške kod unosa. Pamćenje identifikatora elemenata iz definicija dijela XML-a više nije potrebno jer je u izradi ispitnih slučajeva moguće na odgovarajućem mjestu odabrati željeni identifikator iz popisa. Sučelje je podijeljeno na segmente. Prikazan je samo segment s podacima koje korisnik uređuje, čime je spriječen prikaz nepotrebnih informacija korisniku.

Nadogradnja Eclipse platforme omogućila je integraciju s postojećim alata poput alata XML editor-a i PyDev-a za pisanje skripti u Pythonu. Korištenje Eclipse platforme omogućuje nezavisan razvoj ostalih alata za Luscu, a kasnije i njihovu međusobno integraciju u jedinstven alat za razvoj Lusca testova i manipulaciju Lusca ispitnim okruženjem.

Grafičko korisničko sučelje za uređivanje Lusca testova omogućilo je jednostavniju izradu XML testova, što je bilo cilj. Kao takvo trebalo bi privući potencijalne korisnike kojima je ručno uređivanje XML-a odbojno i nepraktično. Moderno i atraktivno grafičko sučelje, dostupno na različitim operacijskim sustavima, pruža korisniku ugodan osjećaj uređivanja i pregleda Lusca testova.

# 8. Literatura


1. J. Rastić, S. Groš, V. Glavinić, LUSCA - Test Framework for Network Applications, Proceedings of Telecommunications & Information, MIPRO Internation Convention, pp. 176-181. Opatija, Croatia, May 2007
2. J. Rastić – Ispitno Okruženje definicija XML-a i parser – Seminar, 2006.
3. Unified Process, Wikipedia, URL:http://en.wikipedia.org/wiki/Unified_Process (07/07/07)
4. Java (programming language) ,Wikipedia URL:http://en.wikipedia.org/wiki/Java_(programming_language) (30/08/07.)
5. XML, Wikipedia, URL:http://en.wikipedia.org/wiki/Xml (15/07/07)
6. XML Schema (W3C), Wikipedia, URL:http://en.wikipedia.org/wiki/Xml_schema (15/07/07)
7. Simple XML serialization Documentation, URL:http://simple.sourceforge.net/ (15/07/07)
8. E. Clayberg, D. Rubel: Eclipse: Building Commercial-Quality Plug-ins, Second Edition, Addison Wesley Professional, 2006.
9. Eclipse Platform – Technical Overview, URL:http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf (15/05/07)
10. The Java Developer's Guide to Eclipse, URL:http://www.jdg2e.com/ch08.architecture/doc/index.html(15/05/07)
11. Eclipse 3.2 Documentation (HTML Help Center) : Platform Plug-in Developer Guide
12. Eclipse 3.3 Documentation (HTML Help Center): Building a Rich Client Platform application
13. Azad Bolour : Notes on the Eclipse Plug-in Architecture URL:http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html (15/05/07.)
14. Standard Widget Toolkit, URL:http://en.wikipedia.org/wiki/Standard_Widget_Toolkit (15/08/07)
15. JFace , Eclipse Wiki, URL:http://wiki.eclipse.org/index.php/JFace (15/08/07)
16. Dejan Glozic, IBM Canada Ltd., Eclipse Forms: Rich UI for the Rich Client URL:http://www.eclipse.org/articles/Article-Forms/article.html (15/08/07)

# Dodatak A: Primjer Lusca test deklaracije

```
<tfs:LuscaTestPackage
 xmlns:tfs="TFSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="TFSchema tfsch.xsd"
LuscaTestID="jednostavni_testni_paket" date="2007-7-30" version="1.0">
   <LuscaTestInfo>
      <Description>Jednostavni test paket</Description>
      <Author email="hrvoje.slavicke@fer.hr"
               name="Hrvoje" surname="Slavicek"
               url="http://lusca.zemris.fer.hr"/>
      <LuscaEngine version="1.0"/>
   </LuscaTestInfo>
   <Definitions>
      <Variables>
         <Variable VarID="usernameVar" value="user"/>
         <Variable VarID="passwordVar" value="pass"/>
      </Variables>
      <Scripts>
         <Script type="execute" lang="python" ScriptID="ping">
            <SrcPath filename="scripts/ping.py" type="local"/>
            <DestPath filename="/tmp/scripts/ping.py"/>
            <IO>
               <Input>
                  <Par index="0" ParID="destIP"/>
               </Input>
               <Output/>
            </IO>
         </Script>
         <Script type="execute" lang="python" ScriptID="scp_copy">
            <SrcPath filename="scripts/scp_copy.py" type="local"/>
            <DestPath filename="/tmp/scripts/scp_copy.py"/>
            <IO>
               <Input>
                  <Par index="0" ParID="username"/>
                  <Par index="1" ParID="password"/>
               </Input>
               <Output/>
            </IO>
         </Script>
         <Script type="execute" lang="python" ScriptID="setIp">
            <SrcPath filename="scripts/set_ip.py" type="local"/>
            <DestPath filename="/tmp/scripts/set_ip.py"/>
         </Script>
         <Script type="execute" lang="python" ScriptID="check_file">
            <SrcPath filename="scripts/check_file.py" type="local"/>
            <DestPath filename="/tmp/scripts/check_file.py"/>
         </Script>
         <Script type="execute" lang="python" ScriptID="shutdown">
            <SrcPath filename="scripts/shutdown.py" type="local"/>
            <DestPath filename="/tmp/scripts/shutdown.py"/>
         </Script>
      </Scripts>
      <Resources>
         <Resource FileID="testFile" type="file">
            <SrcPath filename="conf/test.txt" type="local"/>
            <DestPath filename="/tmp/test.txt"/>
         </Resource>
      </Resources>
```

```
      <Applications>
         <Application AppID="customScp">
            <Description>Simple scp to test</Description>
            <Author email="hrvoje.slavicke@fer.hr"
                    name="hrvoje"
                    surname="slavicek"
                    url="http://lusca.zemris.fer.hr"/>
            <Info name="scp" url="http://localhost" version="1.0"/>
            <Repository type=""></Repository>
            <DependsOn>
               <CompileTime/>
               <RunTime/>
            </DependsOn>
            <Compile type=""/>
         </Application>
      </Applications>
      <Equipment>
         <Computer CompID="client" type="real">
            <Description>Client app that starts ping</Description>
            <Network>
               <Interface gateway="0.0.0.0"
                          address="192.168.0.64"
                          IfaceID="eth0"
                          type="ethernet"
                          netmask="255.255.255.0"/>
            </Network>
            <OS type="linux"/>
         </Computer>
         <Computer CompID="server" type="real">
            <Description></Description>
            <Network>
               <Interface gateway="0.0.0.0"
                          address="192.168.0.1"
                          IfaceID="eth0"
                          type="ethernet"
                          netmask="255.255.255.0"/>
            </Network>
            <OS type="linux"/>
         </Computer>
      </Equipment>
   </Definitions>
   <Tests>
      <Test TestID="Simple_copy_test">
         <Description>Ping server
if active
copy file to server and see if file is copied ok</Description>
         <DependsOn/>
         <Preparation>
            <AppDistribution>
               <Application id="customScp">
                  <Host id="client"/>
               </Application>
            </AppDistribution>
            <FileDistribution>
               <File id="testFile">
                  <Host id="client"/>
               </File>
            </FileDistribution>
            <Initialization>
               <Host id="client">
                  <RunScript id="setIp"/>
               </Host>
               <Host id="server">
                  <RunScript id="setIp"/>
               </Host>
```

```
            </Initialization>
         </Preparation>
         <Execution>
            <Batch BatchID="simpleBatch">
               <Executors>
                  <Host id="client"/>
               </Executors>
               <Variables/>
               <StartsOn>
                  <Trigger signal="START" from="TFE"/>
               </StartsOn>
               <Command type="script">
                  <RunScript id="ping">
                     <Connect>
                        <InVal Val="192.168.0.1" Par="destIp"/></Connect>
                     <Actions>
                        <OnEvent name="IP_OK">
                           <RunScript id="scp_copy">
                              <Connect>
                                 <InVar Par="username" Var="usernameVar"/>
                                 <InVar Par="password" Var="passwordVar"/>
                              </Connect>
                           </RunScript>
                        </OnEvent>
                     </Actions>
                  </RunScript>
               </Command>
            </Batch>
         </Execution>
         <Conclusion>
            <Host id="client">
               <RunScript id="shutdown"/>
            </Host>
            <Host id="server">
               <RunScript id="shutdown"/>
            </Host>
         </Conclusion>
      </Test>
   </Tests>
</tfs:LuscaTestPackage>
```

# Dodatak B: XML Schema Lusca ispitne deklaracije

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="TFSchema"
  xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tfs="TFSchema">

  <complexType name="ETComputerDef">
    <sequence>
      <element name="Description" type="string" maxOccurs="1"
        minOccurs="1">
      </element>
      <element name="Network">
        <complexType>
          <sequence>
            <element name="Interface" maxOccurs="unbounded"
              minOccurs="1">
              <complexType>
                <attribute name="IfaceID"
                  type="tfs:ATIdType" use="required" />

                <attribute name="type"
                  type="tfs:ATIfType" use="required" />


                <attribute name="address"
                  type="tfs:ATIpv4Type" use="required" />

                <attribute name="netmask"
                  type="tfs:ATIpv4Type" use="required" />

                <attribute name="gateway"
                  type="tfs:ATIpv4Type" use="required" />
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
      <element name="OS">
        <complexType>
          <attribute name="type" type="string" use="required" />

        </complexType>
      </element>
    </sequence>
    <attribute name="CompID" type="tfs:ATIdType" use="required" />
    <attribute name="type" type="string" use="required" />
  </complexType>

  <complexType name="ETRunScript">
    <sequence>
      <element name="Connect" maxOccurs="1" minOccurs="0">
        <complexType>
          <sequence>
            <element name="InVal" type="tfs:ETValToPar"
              maxOccurs="unbounded" minOccurs="0">
            </element>


            <element name="InVar" type="tfs:ETVarToPar"
              maxOccurs="unbounded" minOccurs="0" />
            <element name="Out" type="tfs:ETRetToVar"
              maxOccurs="unbounded" minOccurs="0" />
          </sequence>
        </complexType>
      </element>
      <element name="Actions" type="tfs:ETAction" maxOccurs="1"
        minOccurs="0">
```

```
      </element>
    </sequence>
    <attribute name="id" type="tfs:ATIdType" use="required" />
  </complexType>

  <complexType name="ETScriptDef">
    <sequence>
      <element name="SrcPath" type="tfs:ETSourceDef" maxOccurs="1"
        minOccurs="1" />
      <element name="DestPath" type="tfs:ETDestinationDef"
        maxOccurs="1" minOccurs="1" />
      <element name="IO" maxOccurs="1" minOccurs="0">
        <complexType>
          <sequence>
            <element name="Input" maxOccurs="1"
              minOccurs="0">
              <complexType>
                <sequence>
                  <element name="Par"
                    type="tfs:ETScriptPar" maxOccurs="unbounded"
minOccurs="1" />
                </sequence>
              </complexType>
            </element>

            <element name="Output" maxOccurs="1"
              minOccurs="0">
              <complexType>
                <sequence>
                  <element name="Ret"
                    type="tfs:ETScriptRet" maxOccurs="unbounded"
minOccurs="1" />
                </sequence>
              </complexType>
            </element>
          </sequence>
        </complexType>

      </element>
    </sequence>
    <attribute name="ScriptID" type="tfs:ATIdType" use="required" />
    <attribute name="type" type="tfs:ATScriptType" use="required" />
    <attribute name="lang" type="tfs:ATScriptLang" use="required" />
  </complexType>


  <complexType name="ETResourceDef">
    <sequence>
      <element name="SrcPath" type="tfs:ETSourceDef" />
      <element name="DestPath" type="tfs:ETDestinationDef" />
    </sequence>
    <attribute name="FileID" type="tfs:ATIdType" use="required" />
    <attribute name="type" type="tfs:ATResourceType"></attribute>
  </complexType>

  <complexType name="ETApplicationDef">
    <sequence>
      <element name="Description" type="string" maxOccurs="1"
        minOccurs="1" />
      <element name="Info" maxOccurs="1" minOccurs="1">
        <complexType>
          <attribute name="name" type="string" />

          <attribute name="version" type="tfs:ATVersion" />

          <attribute name="url" type="string" />

        </complexType>
```

```
      </element>
      <element name="Author" type="tfs:ETAuthorDef" maxOccurs="1"
        minOccurs="1" />
      <element name="Repository">
        <complexType>
          <sequence>
            <element name="Source" maxOccurs="unbounded"
              minOccurs="0" type="tfs:ETSourceAppDef">
            </element>
          </sequence>
          <attribute name="type" type="tfs:ATAppDistr"
            use="required" />
        </complexType>

      </element>
      <element name="DependsOn">
        <complexType>
          <sequence>
            <element name="CompileTime">
              <complexType>
                <sequence>
                  <element name="Pkg" type="tfs:ETPkg"
                    maxOccurs="unbounded" minOccurs="0" />
                </sequence>
              </complexType>
            </element>
            <element name="RunTime">
              <complexType>
                <sequence>
                  <element name="Pkg" type="tfs:ETPkg"
                    maxOccurs="unbounded" minOccurs="0" />
                </sequence>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
      <element name="Compile">
        <complexType>
          <sequence>
            <element name="Patch" maxOccurs="1"
              minOccurs="0">
              <complexType>
                <attribute name="filename" type="string"
                  use="required">
                </attribute>
              </complexType>
            </element>
            <element name="ConfigOpts" maxOccurs="1"
              minOccurs="0">
              <complexType>
                <attribute name="config" type="string"
                  use="required">
                </attribute>
              </complexType>
            </element>
          </sequence>
          <attribute name="type" type="tfs:ATCompileType"
            use="required" />

        </complexType>
      </element>
    </sequence>
    <attribute name="AppID" type="tfs:ATIdType" use="required" />
  </complexType>

  <complexType name="ETVariableDef">
    <attribute name="VarID" type="tfs:ATIdType" use="required" />
```

```
  <attribute name="value" type="string" use="required" />
</complexType>

<complexType name="ETPkg">
  <attribute name="name" type="string" use="required" />
  <attribute name="version" type="string" />
</complexType>

<complexType name="ETAuthorDef">
  <attribute name="name" type="string" use="required" />
  <attribute name="surname" type="string" use="required" />
  <attribute name="url" type="string" />
  <attribute name="email" type="tfs:ATEmail" />
</complexType>

<complexType name="ETScriptPar">
  <attribute name="ParID" type="tfs:ATIdType" use="required" />
  <attribute name="index" type="integer" use="required" />
</complexType>

<complexType name="ETScriptRet">
  <attribute name="RetID" type="tfs:ATIdType" use="required" />
  <attribute name="index" type="integer" use="required" />
</complexType>

<complexType name="ETSourceDef">
  <attribute name="type" type="tfs:ATSrcPathType" use="required" />
  <attribute name="filename" type="string" use="required" />
</complexType>

<complexType name="ETSourceAppDef">
  <attribute name="type" type="tfs:ATAppSrc" use="required" />
  <attribute name="revision" type="integer" use="optional" />
  <attribute name="path" type="string" use="required" />
</complexType>

<complexType name="ETDestinationDef">
  <attribute name="filename" type="tfs:ATDestPathType"
    use="required" />
</complexType>

<complexType name="ETMakeSignalDef">
  <attribute name="type" type="tfs:ATEventTp" use="required" />
  <attribute name="name" type="string" use="required" />
  <attribute name="wait" type="string" />

</complexType>

<complexType name="ETOnEvent">
  <sequence>
    <element name="MakeSignal" type="tfs:ETMakeSignalDef"
      maxOccurs="unbounded" minOccurs="0" />
    <element name="RunScript" type="tfs:ETRunScript"
      maxOccurs="unbounded" minOccurs="0" />
  </sequence>
  <attribute name="name" type="tfs:ATIdType" use="required" />
</complexType>

<complexType name="ETVarToPar">
  <attribute name="Par" type="tfs:ATIdType" use="required" />
  <attribute name="Var" type="string" />
</complexType>

<complexType name="ETValToPar">
  <attribute name="Par" type="string" />
  <attribute name="Val" type="string" />
</complexType>
```

```
<complexType name="ETRetToVar">
  <attribute name="Ret" type="tfs:ATIdType" use="required" />
  <attribute name="Var" type="tfs:ATIdType" use="required" />
</complexType>

<simpleType name="ATAppSrc">
  <restriction base="string">
    <enumeration value="http" />
    <enumeration value="ftp" />
    <enumeration value="svn" />
    <enumeration value="local" />
  </restriction>
</simpleType>

<simpleType name="ATAppDistr">
  <restriction base="string">
    <enumeration value="source" />
    <enumeration value="binary" />
  </restriction>
</simpleType>

<simpleType name="ATEmail">
  <restriction base="string">
    <pattern value="[a-z].*(@)[a-z].*" />
  </restriction>
</simpleType>

<simpleType name="ATDestPathType">
  <restriction base="string"></restriction>
</simpleType>

<simpleType name="ATScriptLang">
  <restriction base="string">
    <enumeration value="sh" />
    <enumeration value="python" />
    <enumeration value="perl" />
  </restriction>
</simpleType>

<element name="LuscaTestPackage">
  <annotation>
    <documentation>Root element</documentation>
  </annotation>
  <complexType>
    <sequence>
      <element name="LuscaTestInfo">
        <annotation>
          <documentation>
            Information about complete test.
          </documentation>
        </annotation>
        <complexType>
          <sequence>
            <element name="Description" type="string">
              <annotation>
                <documentation>
                  Description
                </documentation>
              </annotation>
            </element>
            <element name="Author"
              type="tfs:ETAuthorDef">
              <annotation>
                <documentation>
                  Test author.
                </documentation>
              </annotation>
            </element>
```

```
              <element name="LuscaEngine">
                <annotation>
                  <documentation>
                    Version of TFE required for test
                    execution.
                  </documentation>
                </annotation>
                <complexType>
                  <attribute name="version"
                    type="tfs:ATVersion" use="required" />

                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>
        <element name="Definitions">
          <annotation>
            <documentation>Defiitions.</documentation>
          </annotation>
          <complexType>
            <sequence>
              <element name="Variables"
                type="tfs:ETVariablesDecl" maxOccurs="1" minOccurs="0" />
              <element name="Scripts">
                <complexType>
                  <sequence>
                    <element name="Script"
                      type="tfs:ETScriptDef" maxOccurs="unbounded"
minOccurs="1" />

                  </sequence>
                </complexType>
              </element>
              <element name="Resources" maxOccurs="1"
                minOccurs="0">
                <complexType>
                  <sequence>
                    <element name="Resource"
                      type="tfs:ETResourceDef" maxOccurs="unbounded"
minOccurs="0" />

                  </sequence>
                </complexType>
              </element>
              <element name="Applications">
                <complexType>
                  <sequence>
                    <element name="Application"
                      type="tfs:ETApplicationDef" maxOccurs="unbounded"
                      minOccurs="0" />

                  </sequence>
                </complexType>
              </element>
              <element name="Equipment">
                <complexType>
                  <sequence>
                    <element name="Computer"
                      type="tfs:ETComputerDef" maxOccurs="unbounded"
minOccurs="1" />

                  </sequence>
                </complexType>
              </element>

            </sequence>
          </complexType>
```

```
      </element>
      <element name="Tests">
        <annotation>
          <documentation>Tests.</documentation>
        </annotation>
        <complexType>
          <sequence>
            <element name="Test" type="tfs:ETTestDef"
              maxOccurs="unbounded" minOccurs="1" />
          </sequence>
        </complexType>
      </element>
    </sequence>
    <attribute name="LuscaTestID" type="tfs:ATIdType"
      use="required" />
    <attribute name="version" type="tfs:ATVersion"
      use="required" />
    <attribute name="date" type="date" />
  </complexType>
</element>

<simpleType name="ATVersion">
  <restriction base="string">
    <pattern value="(\d+(\.))*\d+" />
  </restriction>
</simpleType>

<complexType name="ETTestDef">
  <sequence>
    <element name="Description" type="string" />
    <element name="DependsOn">
      <complexType>
        <sequence>
          <element name="Test" maxOccurs="unbounded"
            minOccurs="0">
            <complexType>
              <attribute name="id" type="string"
                use="required" />

            </complexType>
          </element>
        </sequence>
      </complexType>
    </element>
    <element name="Preparation">
      <complexType>
        <sequence>
          <element name="AppDistribution">
            <complexType>
              <sequence>
                <element name="Application"
                  maxOccurs="unbounded" minOccurs="1">
                  <complexType>
                    <sequence>
                      <element name="Host"
                        maxOccurs="unbounded" minOccurs="1"
                        type="tfs:ETHostType">
                      </element>
                    </sequence>
                    <attribute name="id"
                      type="tfs:ATIdType" use="required" />

                  </complexType>
                </element>
              </sequence>
            </complexType>
          </element>
          <element name="FileDistribution">
```

```
              <complexType>
                <sequence>
                  <element name="File"
                    maxOccurs="unbounded" minOccurs="0">
                    <complexType>
                      <sequence>
                        <element name="Host"
                          maxOccurs="unbounded" minOccurs="1"
                          type="tfs:ETHostType">
                        </element>
                      </sequence>
                      <attribute name="id"
                        type="tfs:ATIdType" use="required" />

                    </complexType>
                  </element>
                </sequence>
              </complexType>
            </element>
            <element name="Initialization">
              <complexType>
                <sequence>
                  <element name="Host"
                    maxOccurs="unbounded" minOccurs="0"
                    type="tfs:ETActionNoSig">
                  </element>
                </sequence>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
      <element name="Execution">
        <complexType>
          <sequence>
            <element name="Batch" maxOccurs="unbounded"
              minOccurs="1">
              <complexType>
                <sequence>
                  <element name="Executors">
                    <complexType>
                      <sequence>
                        <element name="Host"
                          maxOccurs="unbounded" minOccurs="1"
                          type="tfs:ETHostType">
                        </element>
                      </sequence>
                    </complexType>
                  </element>
                  <element name="Variables"
                   type="tfs:ETVariablesDecl" maxOccurs="1" minOccurs="0"/>
                  <element name="StartsOn">
                    <complexType>
                      <sequence>
                        <element name="Trigger"
                           type="tfs:ETTrigger" maxOccurs="unbounded"
minOccurs="1" />

                      </sequence>
                    </complexType>
                  </element>

                  <element name="Command">
                    <complexType>
                      <sequence>
                        <element
                          name="RunScript" type="tfs:ETRunScript" />
```

```
                    </sequence>
                    <attribute name="type"
                      type="string" />

                  </complexType>
                </element>
              </sequence>
              <attribute name="BatchID"
                type="tfs:ATIdType" use="required" />

            </complexType>
          </element>
        </sequence>
        <attribute name="maxtime" type="int"
          use="optional">
        </attribute>
      </complexType>
    </element>
    <element name="Conclusion">
      <complexType>
        <sequence>
          <element name="Host" type="tfs:ETActionNoSig"
            maxOccurs="unbounded" minOccurs="0">
          </element>
        </sequence>
      </complexType>
    </element>
  </sequence>
  <attribute name="TestID" type="tfs:ATIdType" use="required" />
</complexType>

<complexType name="ETVariablesDecl">
  <sequence>
    <element name="Variable" type="tfs:ETVariableDef"
      maxOccurs="unbounded" minOccurs="0" />
  </sequence>
</complexType>

<complexType name="ETTrigger">
  <attribute name="from" type="string" use="required"></attribute>
  <attribute name="signal" type="string" use="required"></attribute>
</complexType>

<complexType name="ETParseDef">
  <sequence>
    <element name="RunScript" type="tfs:ETRunScript" />
    <element name="OnEvent" type="tfs:ETOnEvent"
      maxOccurs="unbounded" minOccurs="0" />

  </sequence>
  <attribute name="signal" type="string"></attribute>
  <attribute name="from" type="string"></attribute>
</complexType>

<complexType name="ETTemplateDef">
  <sequence>
    <element name="Symbol" maxOccurs="unbounded"
      minOccurs="1">
      <complexType>
        <attribute name="SymID" type="tfs:ATIdType"
          use="required" />

        <attribute name="string" type="string"
          use="required" />

        <attribute name="value" type="string"
          use="required" />
```

```
      </complexType>
    </element>
  </sequence>
  <attribute name="TplID" type="tfs:ATIdType" use="required" />
</complexType>

<complexType name="ETApplyTemplate">
  <sequence>
    <element name="Replace" maxOccurs="unbounded"
      minOccurs="0">
      <complexType>
        <attribute name="Sym" type="tfs:ATIdType"
          use="required" />

        <attribute name="Var" type="tfs:ATIdType"
          use="required" />

      </complexType>
    </element>
  </sequence>
  <attribute name="Tpl" type="tfs:ATIdType" use="required" />
  <attribute name="File" type="tfs:ATIdType" use="required" />
</complexType>

<simpleType name="ATScriptType">
  <restriction base="string">
    <enumeration value="calculate"></enumeration>
    <enumeration value="parse"></enumeration>
    <enumeration value="execute"></enumeration>
  </restriction>
</simpleType>

<simpleType name="ATSrcPathType">
  <restriction base="string">
    <enumeration value="local"></enumeration>
    <enumeration value="http"></enumeration>
    <enumeration value="ftp"></enumeration>
  </restriction>
</simpleType>

<simpleType name="ATIdType">
  <restriction base="string"></restriction>
</simpleType>

<simpleType name="ATCompileType">
  <restriction base="string">
    <enumeration value="beforeDistribution"></enumeration>
    <enumeration value="afterDistribution"></enumeration>
  </restriction>
</simpleType>

<simpleType name="ATIpv4Type">
  <restriction base="string"></restriction>
</simpleType>

<simpleType name="ATIfType">
  <restriction base="string">
    <enumeration value="ethernet"></enumeration>
  </restriction>
</simpleType>

<simpleType name="ATVariableTpType">
  <restriction base="string">
    <enumeration value="string"></enumeration>
    <enumeration value="integer"></enumeration>
    <enumeration value="decimal"></enumeration>
    <enumeration value="boolean"></enumeration>
  </restriction>
```

```
</simpleType>

<complexType name="ETHostType">
  <attribute name="id" type="tfs:ATIdType" use="required"></attribute>
</complexType>

<simpleType name="ATYesNoType">
  <restriction base="string"></restriction>
</simpleType>

<simpleType name="ATEventTp">
  <restriction base="string">
    <enumeration value="info"></enumeration>
    <enumeration value="error"></enumeration>
    <enumeration value="fatal"></enumeration>
    <enumeration value="pass"></enumeration>
    <enumeration value="warn"></enumeration>
    <enumeration value="debug"></enumeration>
  </restriction>
</simpleType>

<simpleType name="ATProbe">
  <restriction base="string"></restriction>
</simpleType>

<simpleType name="ATProtocol">
  <restriction base="string">
    <enumeration value="TCP"></enumeration>
    <enumeration value="UDP"></enumeration>
  </restriction>
</simpleType>

<complexType name="ETActionNoSig">
  <sequence>
    <element name="RunScript" type="tfs:ETRunScriptMin"
      maxOccurs="unbounded" minOccurs="0">
    </element>
  </sequence>
  <attribute name="id" type="tfs:ATIdType" use="required"></attribute>
</complexType>

<complexType name="ETPassOn">
  <sequence>
    <element name="Trigger" type="tfs:ETTrigger"></element>
  </sequence>
</complexType>

<complexType name="ETAction">
  <sequence>
    <element name="OnEvent" type="tfs:ETOnEvent"
      maxOccurs="unbounded" minOccurs="1">
    </element>
  </sequence>
</complexType>

<complexType name="ETRunScriptMin">
  <attribute name="id" type="tfs:ATIdType"></attribute>
</complexType>

<complexType name="ETScriptResultFiles">
  <sequence>
    <element name="File" maxOccurs="unbounded" minOccurs="0">
      <complexType>
        <attribute name="path" type="string"></attribute>
        <attribute name="FileID" type="string"></attribute>
      </complexType>
    </element>
  </sequence>
```

```
  </complexType>
  <simpleType name="ATResourceType">
    <restriction base="string">
      <enumeration value="file"></enumeration>
      <enumeration value="directory"></enumeration>
    </restriction>
  </simpleType>

</schema>
```