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

DIPLOMSKI RAD br. 1672

Sučelje za uporabu Lusca sustava iz Eclipse razvojnog okruženja

Dražen Nežić

Zagreb, rujan 2007.

Sažetak

U ovom radu opisane su osnovne karakteristike i arhitektura Lusca ispitnog okruženja za automatizirano provođenje testova. Postavljeni su zahtjevi za izgradnju korisničkog sučelja za udaljeno upravljanje Lusca ispitnim okruženjem. Postavljeni zahtjevi implementirani su u vidu dodatka za Eclipse, Lusca Controller. Uporaba, implementacija i korištene tehnologije pri njegovoj izradi također su opisane ovim radom.

Abstract

This diploma thesis describes basic architecture and specifications of Lusca Test Framework, framework for remote testing of network applications. Requirements for development of user friendly management console are set. According to these requirements, Lusca Controller was built, in form of Eclipse plugin. Lusca Controller and used technologies during plugin's development are described in this work.

Sadržaj

1. Uvod

Lusca je sustav u ranoj fazi razvoja čija je primarna namjena automatizirano ispitivanje mrežnih aplikacija. Za razliku od postojećih sustava za ispitivanje aplikacija, Lusca nastoji cijeli proces pojednostavniti i približiti korisniku, programeru, koji ne želi previše vremena utrošiti na pripremu već na samu provedbu ispitivanja. Ispitivanje se provodi automatizirano, tijekom kojeg korisnik ima uvid u njegovo provođenje, a po završetku preuzima rezultate i analizira ih. Tijek izvršavanja testa, upute Lusci, nalaze se u datoteci, ispitnoj deklaraciji. Ispitna deklaracija je XML datoteka definirana pravilima posebno kreiranima za ovu namjenu. Kako je bilo potrebno omogućiti značajnu izražajnost ispitne deklaracije, njezina složenost nije zanemariva te je za novog korisnika vrlo teško pripremiti ispitivanje aplikacija. Osim teškoća prilikom deklariranja testova, zbog njihove složenosti, problematično je kontrolirati i nadzirati tijek ispitivanja pogotovo pri većem broju testova kojima se vrši ispitivanje. Također, potrebno je istovremeno omogućiti uporabu Lusce od strane više korisnika s udaljenih računala. Kako bi se olakšala uporaba Lusca ispitnog okruženja izrađeni su dodaci za razvojno okruženje *Eclipse* u obliku uređivača ispitnih paketa i Lusca upravljačke aplikacije. Potonji dodatak tema je ovog diplomskog rada. Osim pojednostavljenja korištenja Lusce ovi dodaci pokušavaju iskoristiti kvalitetu i popularnost razvojnog okruženja Eclipse te na taj način i sami popularizirati Luscu te je približiti mnoštvu Eclipseovih korisnika. S druge strane, poželjno je izgradnju i ispitivanje aplikacija provoditi unutar istog razvojnog okruženja.

U drugom poglavlju diplomskog rada opisuju se arhitektura Lusca ispitnog okruženja, ispitna deklaracija, način na koji Lusca provodi ispitivanje i zahtjevi za izgradnjom upravljačkog programa. U trećem poglavlju opisane su korištene tehnologije pri razvoju upravljačkog programa i osnove uporabe alata za kontrolu inačica datoteka. Četvrto poglavlje opisuje mogućnosti, osnovnu arhitekturu, upute za korištenje i nedostatke dodatka za upravljanje Lusca ispitnim okruženjem.

2. Lusca ispitno okruženje

Lusca je okruženje namjenjeno automatiziranom provođenju testova, s naglaskom na ispitivanju mrežnih aplikacija. Problematika koju Lusca želi riješiti je ispitivanje aplikacija provođenjem testova, pri čemu je često nužno istovremeno pokretanje više programa koji međusobno komuniciraju i pri tome generiraju izlazne podatke. Korisnik, programer, morao bi svaku od tih aplikacija dopremiti na računalo ili više računala, prevesti svaku zasebno iz izvornog koda te svaku od njih pokrenuti s određenim parametrima. Taj proces može uključivati i na desetke aplikacija i računala. Također, morao bi pratiti izlaz svake aplikacije te analizirati ispravnost rada. Pri svakoj promjeni koju izvrši nad aplikacijom cijeli postupak mora ponoviti, što je velika količina vremena utrošena na ispitivanje. *Ispitnom deklaracijom* korisnik može automatizirati sve navedene korake pri ispitivanju.

2.1. Ispitna deklaracija

Ispitna deklaracija je XML datoteka opisana XML shemom, prikazanoj u Dodatku A, koja sadrži sve informacije potrebne Lusci za provedbu ispitivanja. Svakom ispitnom deklaracijom može se izvršiti jedan ili više ispitnih slučajeva, testova (*test cases*). Ispitna deklaracija sastoji se od tri osnovna dijela:

- zaglavlje
  Sadrži osnovne informacije o autoru testa, datumu kreiranja testa i opis.
- definicija komponenti
  Sadrži podatke o potrebnim skriptama, datotekama, varijablama, aplikacijama i računalnoj opremi koja sudjeluje u ispitivanju (poglavlje 2.3). Aplikacija koja sudjeluje u ispitivanju opisuje se imenom, autorom, te lokacijom izvornog ili izvršnog koda. Varijable služe za čuvanje rezultata izvršavanja skripti te kao parametri pri pokretanju skripti. Ukoliko su varijable deklarirane u ovom odjeljku, vrijede globalno za sve ispitne slučajeve. Dodatno se varijable mogu deklarirati unutar ispitnih slučajeva.
- ispitni slučajevi (testovi)
  Ispitna deklaracija može sadržavati jedan ili više ispitnih slučajeva koje se želi provesti. Ispitni slučajevi mogu biti međusobno ovisni. Ukoliko su definirani kao međusobno ovisni, test A, na primjer, neće moći početi s izvršavanjem ukoliko je definiran završetak testa B kao uvjet za njegov početak.

Ispitna deklaracija s pripadajućom XML shemom, potrebne skripte i ostale datoteke koje sudjeluju u provedbi ispitivanja, zapakirane u TAR GZ arhivu čine *ispitni paket*.

2.2. Ispitni slučaj

Ispitni slučaj sastoji se od tri faze izvođenja:

- priprema (*preparation*)
  Faza pripreme opisana je *Preparation* elementom XML datoteke. Tijekom faze pripreme vrši se prilagođavanje okoline za ispitivanje kako bi se ispitni slučajevi mogli provesti. U fazi pripreme, vrši se razmještaj aplikacija i datoteka određenih

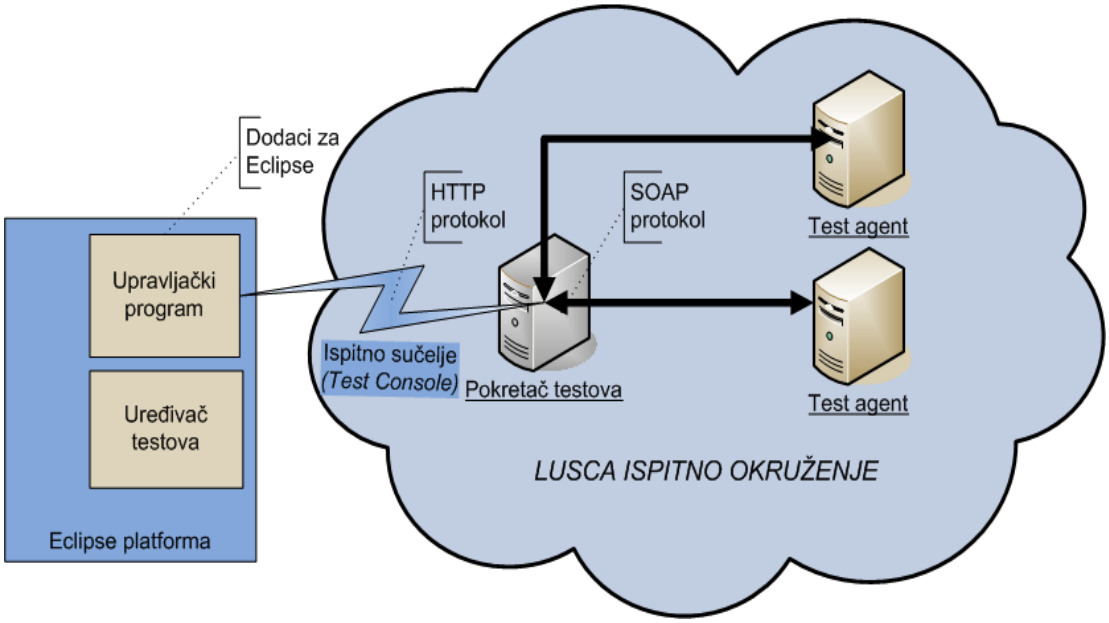
definicijom komponenti. Također, tijekom ove faze, korisnički definiranom skriptom može se dodatno pripremiti ispitno okruženje.

- provedba (*execution*)
  Tijekom faze provedbe, opisane *Execution* elementom XML sheme, vrši se pokretanje skripti koje vrše ispitivanje. Execution element sadrži poslove (*batches*), definirane XML elementom *Batch*. Posao se može izvršavati na jednom ili više agenata, što je definirano njegovim XML podelementom *Executors*. Za svaki posao korisnik može dodatno deklarirati varijable koje se mogu koristiti tijekom izvršavanja ispitnog slučaja. Izvršavanje posla započinje primitkom *signala* koji je definiran imenom unutar elementa *StartsOn*, a završava signalom definiranim elementom *EndsOn*. Izvršavanje posla definirano je XML elementom *Command*, kojim je opisana naredba. Naredba se sastoji od podelemenata *RunScript* i *ResultFile*. Elementom *RunScript* definirana je skripta koja će se pokrenuti, uz pripadajuće ulazne i izlazne varijable. Cijelo vrijeme tijekom izvršavanja skripte prati se izlaz koji ona generira. Na taj način skripta komunicira s ispitnim okruženjem. Skripta može na svojem izlazu, u obliku posebno formatirane tekstualne linije, generirati *događaj*. Za svaki događaj, definiran imenom, može se definirati akcija koja će se izvršiti po nastanku tog događaja. Akcija je definirana XML elementom *Actions*, podelementom *RunScript* elementa. Akcija je opisana imenom događaja kojeg očekuje, odnosno na koji se mora izvršiti. Akcija može biti generiranje signala ili pokretanje neke druge skripte, čime se može izvršiti ugnježdavanje poziva skripti. Ono što je događaj za skriptu to je signal za računala na kojima se izvodi ispitivanje, odnosno način prenošenja informacija. Događajem skripta predaje informacije ispitnom okruženju, unutar računala na kojemu se izvršava, a signalom se ta informacija može dalje prenijeti na druga računala unutar ispitnog okruženja. Za razliku od događaja, signal ima i tip koji može biti *info*, *error* ili *fatal*, od kojih tip *fatal* označava događaj koji rezultira neuspješnim provođenjem testa. *ResultFiles* elementom omogućeno je definiranje skripti koje će na pristigli signal započeti obradu datoteka koje predstavljaju rezultate provedbe ispitnog slučaja. Proces obrade rezultata može rezultirati generiranjem događaja koji će donijeti informaciju ispitnom okruženju o uspješnosti provedbe ispitivanja.
- zaključak (*conclusion*)
  Tijekom ove posljednje faze vrši se prikupljanje rezultata ispitivanja, prijenos zapisa o provedbi ispitivanja (*log*) do korisnika. Također, u ovoj fazi može se pokrenuti korisnički definirana skripta koja će dodatno analizirati rezultate ispitivanja te donijeti zaključak o uspješnosti provedbe testa.

2.3. Arhitektura Lusca ispitnog okruženja

Lusca ispitno okruženje se sastoji od pokretača ispitnog okruženja i jednog ili više agenata (slika 2.1). Pokretač ispitnog okruženja ima dvojaku funkciju, upravljanje procesom ispitivanja te interakcija s korisnikom uporabom ispitnog sučelja (*Test Console*). Upravljanje procesom ispitivanja uključuje pripremanje ispitne okoline, provjeru dostupnosti komponenti koje sudjeluju u ispitivanju, komunikaciju između agenata, dohvat datoteka, prevođenje aplikacija u izvršni kod, izvršavanje ispitivanja prema ispitnoj deklaraciji te naposljetku prikupljanje rezultata ispitivanja. Lusca agenti su računala ili virtualna računala na kojima se, u stvari, provodi ispitivanje. Agenti međusobno komuniciraju preko pokretača

ispitnog okruženja koji koordinira njihovim radom. Agenti i pokretač ispitnog okruženja komuniciraju međusobno SOAP protokolom.



Slika 2.1: Arhitektura Lusca ispitnog okruženja

2.3.1. Ispitno sučelje

Ispitno sučelje (*Test Console*) omogućuje interakciju korisnika s Luscom uporabom HTTP protokola. Ispitnom sučelju može se pristupiti internet preglednikom ili Lusca Controller dodatkom za Eclipse. Pristup internet preglednikom ne zahtijeva instalaciju dodatne programske opreme, a njime se mogu vršiti akcije pokretanja ispitnih paketa, pregledavanja rezultata ispitivanja, dodavanja novih ispitnih paketa. Uporaba dodataka za Eclipse omogućuje, pored navedenog, uređivač ispitnih paketa te nešto više mogućnosti i lakoće pri upravljanju Luscom. Detaljan pregled svih mogućnosti naveden je u poglavlju 4.1.

2.3.2. Pokretač ispitnog okruženja

Uloga pokretača ispitnog okruženja (*Test Engine*) je koordinacija svih Lusca agenata i omogućavanje pristupa korisnicima ispitnom okruženju uporabom ispitnog sučelja (slika 2.1). Korisnik, izvana, ne može izravno pristupiti Lusca agentima, već informacije o njihovom statusu prima slanjem upita pokretaču ispitnog okruženja. Pokretač istovremeno može provoditi više ispitnih paketa, ovisno o broju raspoloživih agenata. Tijekom izvođenja ispitnog paketa, pokretač ispitnog okruženja šalje signale zaprimljene od nekog agenta svim ostalim agentima koji imaju definiranu neku akciju na taj signal. Završetkom ispitivanja pokretač ispitnog okruženja prikuplja sve informacije o tijeku ispitivanja s agenata te ih omogućuje korisniku za dohvat i podrobniju analizu.

2.3.3. Agent

Agent je zadužen za izvršavanje ispitivanja. Izvršava skripte koje služe za pripremu ispitne okoline, izvršava *Batch* elemente definirane ispitnom deklaracijom, šalje i prima signale od

pokretača ispitnog okruženja te, ovisno o ispitnoj deklaraciji, izvršava akcije ili pokreće skripte (poglavlje 2.2). Agenti ne mogu međusobno komunicirati, već se sva komunikacija između njih vrši preko posrednika, pokretača ispitnog okruženja.

2.4. Zahtjevi daljinskog upravljanja Lusca ispitnim okruženjem

Upravljanje Lusca ispitnim okruženjem može se podijeliti na dvije cjeline:

- nadzor pokretača ispitnog okruženja i agenata
- kontrola provođenja testova

Komunikacija između korisnika i Lusca ispitnog okruženja provodi se između upravljačkog programa i pokretača ispitnim sučeljem. Sve podatke o agentima i testovima korisnik dohvaća slanjem zahtjeva pokretaču ispitnog okruženja koji ih mora pribaviti i proslijediti. Na taj način, samo računalo na kojemu se nalazi pokretač ispitnog okruženja mora biti dostupno upravljačkom programu.

2.4.1. Nadzor pokretača ispitnog okruženja i agenata

S obzirom da se najbitnije operacije i cijelo upravljanje Lusca ispitnim okruženjem (agentima, ispitnim paketima) odvijaju na pokretaču ispitnog okruženja, poželjno je pratiti podatke koji opisuju stanje računala na kojemu se on nalazi. Uporabom upravljačkog programa korisnik može doći do informacija o zauzeću memorije, opterećenju procesora, pokrenutim procesima i ostalim pokazateljima rada računala te donijeti određene zaključke. Testovi se izvršavaju na agentima, koji mogu biti na zasebnim računalima, pa je potrebno omogućiti praćenje i njihovog rada. Osim gore navedenih parametara lako se može implementirati i dohvat svih ostalih informacija koje korisnika interesiraju a opisuju stanje računala, raspoložive resurse. Stanja resursa računala na kojima se testovi izvršavaju, agenata, mogu biti pokazatelj ispravnosti rada aplikacija koje se ispituju. Na taj način mogu se identificirati prekomjerno zauzeće resursa, bezgranično zauzimanje memorije (*memory leak*), pa čak i beskonačno čekanje na uvjet, odnosno zastoj.

2.4.2. Kontrola provođenja testova

Kako je osnovna zadaća Lusce provedba testova, najveći naglasak treba staviti na upravljanje i nadzor njihovog izvođenja. Provedba testova definirana je ispitnim paketom, odnosno ispitnom deklaracijom, koja se mora nalaziti na pokretaču ispitnog okruženja kako bi ispitivanje moglo započeti. Ukoliko ispitni paket nije unaprijed dostupan pokretaču, potrebno je omogućiti prijenos paketa s korisnikovog računala pomoću upravljačkog programa. Također, ispitni paket može se i preuzeti s pokretača te dati korisniku na uređivanje uporabom Eclipse dodatka za uređivanje Lusca ispitnih paketa.

Ispitni paket se može uspješno pokrenuti ukoliko su zadovoljeni svi preduvjeti koje njegova ispitna deklaracija zahtijeva. Aplikacije koje se ispituju, sadržane u ispitnoj deklaraciji, moraju biti dostupne Lusci, pa se njihov opis i lokacija mogu uporabom upravljačkog programa definirati ili naknadno uređivati. Lusca zahtijeva adresu Subversion repozitorija (poglavlje 3.4) koji sadrži određenu aplikaciju koja sudjeluje u ispitivanju. Osim lokacije koda potrebno je unijeti simboličko ime aplikacije, kratak opis te adresu internetske stranice aplikacije, ako ona postoji.

Korisnik može uporabom upravljačkog programa pokrenuti ili zaustaviti izvršavanje pojedinog ispitnog paketa. Kako svaki ispitni paket može sadržavati više ispitnih slučajeva (*test cases*), korisnik može odabrati pokretanje točno određenih, unutar jednog ispitnog paketa, te na taj način jednostavno modificirati opseg i uvjete ispitivanja. Kako bi se ispitivanje još više olakšalo, potrebno je omogućiti provedbu definiranih testova s različitim verzijama aplikacija. Tako se za svaku aplikaciju koja sudjeluje u ispitivanju može točno odabrati verzija, s kojom se želi provesti ispitivanje, uporabom alata za nadzor izvornog koda. Ispitivanje se, na taj način, može vršiti s bilo kojom revizijom aplikacije ili s radnom kopijom aplikacije, odnosno revizijom aplikacije na kojoj programer upravo radi te koja se nalazi na njegovom osobnom računalu.

Tijekom same provedbe ispitivanja aplikacije, svi zapisi (*logs*) koji se generiraju moraju se moći prosljeđivati upravljačkoj aplikaciji u realnom vremenu kao i dohvat detaljnijih zapisa i informacija po završetku ispitivanja. Osim zapisa, potrebna je informacija o rezultatu ispitivanja, odnosno jesu li testovi uspješno ili neuspješno provedeni.

2.5. Problematika daljinskog upravljanja Luscom

Aplikacija za daljinsko upravljanje Luscom povezuje se s pokretačem ispitnog okruženja ispitnim sučeljem. Pri izradi aplikacije za daljinsko upravljanje treba implementirati protokol kojim Lusca podržava komunikaciju. S obzirom da je Lusca osmišljena kao lako (eng. *lightweight*) okruženje, izvedeno u cjelosti na tehnologijama otvorenog koda i sama aplikacija za daljinsko upravljanje mora biti dizajnirana i implementirana u istom duhu. Također, s obzirom da je Luscino ispitno sučelje izvedeno na pokretaču ispitnog okruženja, odnosno jednom računalu, sučelje ne smije previše opterećivati računalo na kojem se odvijaju najvažniji procesi pri ispitivanju. Stoga, treba uzeti u obzir moguću količinu podataka koji će se prenositi i koliko bi takva komunikacija mogla opterećivati računalne resurse pokretača ispitnog okruženja. Potrebno je osigurati višekorisnički rad, odnosno zadavanje naredbi Lusci od strane više korisnika uporabom više upravljačkih programa, pri tome svu sinkronizaciju treba obavljati na Lusci, također, više korisnika može istovremeno pratiti zapise generirane provedbom ispitivanja.

3. Tehnologije korištene pri izradi dodatka za Eclipse

Sve tehnologije korištene pri izradi upravljačkog programa za Lusca ispitno okruženje, u vidu dodatka za Eclipse, dostupne su i licencirane po načelima slobodnog koda. Kako je temelj izgradnje upravljačkog sučelja Java platforma, samo sučelje je prenosivo i neovisno o operativnom sustavu na kojemu se izvršava, ipak dodatak je predviđen radu na Linux operativnom sustavu.

3.1. Java

Java je objektno-orijentirano, neovisno o platformi, višedretveno programersko okruženje. Java programski jezik razvio je James Gosling 1991. godine za uporabu u malom uređaju kućne zabavne industrije, koji je razvijao *Sun Microsystems*. Jezik je originalno nazvan *Oak*, po hrastu koji je rastao ispred njegovog ureda. Goslingove ideje su bile razviti virtualni stroj koji će izvršavati Javu i jezik blizak C/C++ notaciji. Uređaj kojemu je Java bila namjenjena nije postao popularan, ali su popularizacijom interneta najveći proizvođači internet preglednika uključili u svoje proizvode mogućnost uporabe Java *appleta* u sklopu web stranica. *Applet* je programska komponenta koja se izvršava u kontekstu drugog programa, npr. web preglednika. Prvu javnu implementaciju Jave izdao je Sun Microsystems 1995. godine. Pokušaji formalizacije Jave pri tijelima, *ISO/IEC JTC1* i *Ecma International*, propali su pa je Java ostala *de facto* standard kontroliran kroz *Java Community Process*. *Java Community Process*, osnovan 1998. godine, je formalan proces koji omogućuje zainteresiranim stranama utjecaj na razvoj i definiranje budućih verzija i mogućnosti Java platforme.

Osnovni ciljevi pri kreiranju jezika Java su:

- objektno-orijentirana programska metodologija
- isti program mora se moći pokrenuti na različitim operativnim sustavima
- mora imati ugrađenu podršku za uporabu računalnih mreža
- pokretanje izvršnog koda s udaljenih izvora na siguran način
- jednostavnost korištenja

3.1.1. Prenosivost Jave

Osnovna ideja, značajka Jave, je prenosivost izvršnog koda između različitih operativnih sustava. Ova značajka postignuta je prevođenjem Java izvornog koda u Java *bytecode*. Java *bytecode* je međujezik, izvorni kod preveden u pojednostavljene instrukcije specifične Java platformi. Tako preveden izvorni kod može se pokretati unutar virtualnog stroja, *Java Virtual Machine*. Virtualni stroj omogućuje interpretiranje i izvršavanje Java međujezika na računalu na kojemu se nalazi. Virtualni stroj omogućuje pristup i uporabu operacija specifičnih za pojedini operativni sustav kojemu je namijenjen. Virtualni stroj izvršava Java međujezik prevodeći ga prije njegovog stvarnog izvršavanja na računalu u strojni kod uporabom *JIT* (*Just in Time*) prevoditelja. Prenosivost ostvarena na ovaj način negativno utječe na performanse izvršavanja koda, stoga Java nosi reputaciju jezika sporih

performansi, pogotovo u usporedbi s jezicima C i C++. Kako bi se povećale performanse izvršavanja Java koda, upotrebljavaju se određene tehnike. Prva tehnika je prevođenje izvornog Java koda izravno u izvršni kod, uporabom npr. *GCJ*-a (*GNU Compiler for Java*). Time se međutim gubi prenosivost. Druga tehnika uključuje prevođenje, uporabom JIT prevoditelja, cijelog izvornog koda u izvršni pri pokretanju Java aplikacije. Ovom tehnikom se zadržava prenosivost i povećavaju performanse, ali se mora dulje čekati na pokretanje programa. Posljednja, tehnika dinamičkog prevođenja, omogućava virtualnom stroju da unaprijed prevede dijelove Java programa za koje postoji pretpostavka da će se češće izvršavati tijekom rada aplikacije. Dinamičkim prevođenjem mogu se postići veće performanse nego pri statičnom prevođenju, jer se tijekom dinamičkog prevođenja u obzir mogu uzeti podatci o okruženju unutar kojeg se program izvršava te o učestalosti poziva metoda i funkcija unutar programa, na osnovu statistike. Na taj način se određene metode ili dijelovi koda mogu dodatno optimirati, npr. dodjeljivanjem većih procesorskih ili memorijskih resursa za te dijelove koda.

3.1.2. Implementacije Jave

Sun Microsystems službeno omogućuje i licencira *Java Standard Edition* platformu za *Microsoft Windows*, *Linux* i *Solaris.* Alternativni proizvođači omogućuju Java okruženja za te iste, ali i druge platforme. Svaka implementacija Jave, da bi dobila Java licencu, mora proći rigorozne testove kompatibilnosti i ispravnosti. Java platforma uključuje *Java Runtime Environment* (JRE), program koji omogućuje pokretanje Java aplikacija na računalu. Ovaj program je namjenjen krajnjim korisnicima koji žele koristiti Java aplikacije na svojem računalu, kao zasebne programe ili u sklopu internet preglednika te je njegovo korištenje besplatno. *Java Development Kit* (JDK) je programski paket namjenjen programerima, sadrži Java prevoditelj (javac), Java *debugger* (jdb), alate za komprimiranje biblioteka funkcija i razreda (programskih paketa) u arhive (jar), te program za automatizirano generiranje dokumentacije izvornog koda (javadoc).

3.1.3. Automatizirano upravljanje memorijskim resursima

Jedno od najznačajnijih svojstava Jave je automatizirano upravljanje memorijskim resursima, uporabom skupljača dereferenciranih objekata, “smeća” (*Garbage collector*). U nekim jezicima brigu o zauzimanju i oslobađanju memorije mora voditi programer, te stoga, u slučaju programerske greške, zaboravljanja oslobađanja rezervirane memorije, nastaje bezgranično zauzimanje, “curenje” memorije (*memory leak*) što u konačnici može rezultirati greškom u radu aplikacije. Također, oslobađanje memorije koja nije prethodno rezervirana za uporabu ili pristup memorijskim lokacijama koje ne pripadaju memorijskom prostoru programa rezultira rušenjem programa. Takve pogreške uglavnom je teško brzo pronaći te mogu prouzrokovati mnoštvo problema, pogotovo u vidu sigurnosnih propusta u aplikacijama (napad prepunjenjem spremnika, *buffer overflow*). U Javi, ovi potencijalni problemi, izbjegavaju se uporabom skupljača dereferenciranih memorijskih objekata. Programer određuje kada će se neki objekt kreirati, a Java će zauzeti potrebnu količinu memorije i nadzirati njegov životni ciklus. Dok program ili neki drugi objekt sadrže referencu, pokazivač na taj prethodno stvoreni objekt, objekt se smatra živim. Kad nema više pokazivača na taj objekt, skupljač smeća automatski obriše memoriju koju je taj nedohvatljiv objekt zauzimao. Na taj način spriječena je greška bezgraničnog zauzimanja memorije. Ipak, ukoliko programer zaboravi ukloniti reference na taj objekt, on se neće moći obrisati iz memorije pa će nastati “curenje” memorije. Kako korisnik, programer ne

može utjecati kako i što će obrisati skupljač smeća, za svaki novostvoreni objekt može se alocirati nova memorija, što je vrlo skup proces. Programer ne može jednom zauzetu memoriju višestruko iskorištavati, kreirajući npr. memorijski bazen (*memory pool*) kako bi poboljšao performanse svojeg programa. Ipak, pri pokretanju Java aplikacije, parametrima može odabrati između više načina rada skupljača smeća te ga prilagoditi aplikaciji koju programira. Prilagodbe uključuju, na primjer, ekstenzivnije korištenje manjih memorijskih segmenata ili rjeđe korištenje većih memorijskih segmenata, skupljač smeća prilagođen radu na višeprocesorskim sustavima, životni vijek dereferenciranih objekata, količinu inicijalno rezervne memorije za aplikaciju, maksimalnu količinu memorije koja se može pridodjeliti aplikaciji, veličinu memorijskog bloka koji se može odjednom zauzeti, agresivnost zauzimanja memorije, agresivnost oslobađanja zauzete memorije i sl. Iako neka aplikacija može u početku svojeg rada imati vrlo brz odziv i performanse, može se dogoditi da uslijed zauzeća veće količine memorije, skupljač smeća svojim učestalim operacijama čišćenja sve većih i većih segmenata utječe na rad cijelog sustava, te u graničnim slučajevima, cijelu aplikaciju učini neuporabljivom. Također, ponašanje i preformanse skupljača smeća uvelike ovise o implementaciji Java platforme, odnosno o operativnom sustavu na kojem se Java virtualni stroj izvršava. Analizom aplikacije i pravilnim odabirom i postavkama skupljača smeća može se postići znatno poboljšanje performansi Java aplikacija. Također, uporabom alata za analizu izvršavanja koda (*profiler*), moguće je detektirati u kojim se segmentima programskog koda gubi najviše vremena ili memorije pa posebnu pozornost obratiti na te dijelove.

3.1.4. Java i otvoreni kod

17. studenog 2006., Sun Microsystems je najavio izdavanje Java razvojnog okruženja (JDK) po pravilima *GNU General Public License* (GPL). Većim dijelom to se i dogodilo 8. svibnja 2007. Veliki dio izvornog koda predan je *OpenJDK* zajednici. OpenJDK zajednica razvija otvorenu inačicu Java razvojnog okruženja, prema specifikaciji *Standard Edition*, definiranu *Java Community Process* zajednicom.

3.2. Apache Log4j

Apache Log4j je programerska alatka, nastala u sklopu Apache zajednice (*Apache Software Foundation*), koja omogućuje programeru generiranje zapisa i slanje istih na razna odredišta. Log4j namjenjen je integriranju u Java aplikacije kojima se može pridodati kao biblioteka, *Java archive*, s nastavkom *jar*. Zbog praćenja ispravnosti rada aplikacija i detektiranja pogrešaka korisno je generirati što više korisnih informacija te zapisivati ih i pratiti pri izvođenju. Neki od problema koji nastaju pri zapisivanju su velika količina informacija koju treba negdje spremiti i proučiti, nedostupnost stroju na kojem se vrši zapisivanje i činjenica da ekstenzivno zapisivanje usporava aplikaciju. Log4j omogućuje hijerarhiju zapisa, granularitet, podjelu prema važnosti informacija uz potpunu kontrolu i konfigurabilnost što vodi k reduciranju volumena informacija koje se zapisuju te povećanju performansi aplikacija koje generiraju zapise. Log4j je posebno dizajniran kako ne bi utjecao na performanse aplikacija koje ga koriste, pa stoga prilikom izdavanja aplikacija krajnjim korisnicima može ostati integriran unutar njih bez bojazni o znatnijem gubitku performansi.

Informacije, zapisi, dijele se na slijedeće razine:

- *DEBUG* (detaljne informacije koje opisuju rad programa, razina se koristi pri razvoju aplikacija)
- *INFO* (informacija, označava normalan rad programa, obično opisuje tijek izvođenja)
- *WARN* (upozorenje, program može normalno nastaviti rad)
- *ERROR* (pogreška u radu aplikacije iz koje se program može oporaviti)
- *FATAL* (vrsta informacije koja označuje pogrešku iz koje se aplikacija ne može oporaviti i normalno nastaviti rad).

Osim ovih nivoa moguće je kreirati i korisnički definirane nivoe. Osnova Log4j paketa za zapisivanje je klasa *Logger* koja se instancira na način:

```
public static Logger log =
Logger.getLogger(LuscaController.class);
```

ili:

```
public static Logger log = Logger.getLogger(“LuscaController”);
```

kao statički atribut nekog razreda, npr. `LuscaController`, nakon čega se može koristiti na način:

```
log.debug("Some debug message.");
```

Ime instanciranog zapisivača označava njegovo mjesto u hijerarhiji svih objekata za zapisivanje. Hijerarhija zapisivača se određuje prema paketu u kojemu se razred nalazi (`LuscaController.class`) ili prema imenu danom od korisnika. Tako, na primjer, zapisivač s nazivom `com.foo` je roditelj zapisivača `com.foo.LuscaController`. Hijerarhijskom podjelom zapisivača prema razredima, paketima, može se vršiti filtiranje i preraspodjela zapisa. Tako da se u zapisnik `com.foo` prosljeđuju svi zapisi generirani u razredima i paketima koje sadrži paket `com.foo`. Osnovni zapisivač, najviši u hijerarhiji, u Log4j-u, je korijenski zapisivač (*root logger*) u koji se prosljeđuju svi zapisi od svih zapisivača.

Log4j može se konfigurirati programski, ali je bolje definirati konfiguracijsku datoteku s informacijom o nivou zapisivanja, raspodjelom zapisa, načinu na koji će se zapisi upisivati ili slati na udaljeno računalo. Na taj način samom promjenom konfiguracijske datoteke moguće je utjecati na nivo i način zapisivanja informacija, bez intervencija u programu.

Konfiguracijsku datoteku naziva `log4j.properties` potrebno je definirati na slijedeći način:

```
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
```

Na ovaj način definirali smo upisivač, `appender`, odnosno objekt koji definira što će se događati sa zapisima. Log4j podržava `ConsoleAppender` (omogućuje ispis zapisa na korisničku konzolu Java aplikacije), upisivače u datoteke, najjednostavniji `FileAppender`, te `RollingFileAppender` koji omogućuje kreiranje sažete kopije zapisa kada dostigne određenu veličinu diskovnog prostora. Log4j podržava i upisivače za GUI komponente, udaljene servere, JMS, NT Event zapisivače i udaljene UNIX Syslog servise.

```
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%-5p %c %x - %m%n
```

Uporabom opisa izlaza zapisa (eng. *layout*), npr. `PatternLayout` paketa moguće je odrediti kako će izgledati zapisi prosljeđeni u neki upisivač. Predloškom za konverziju može se postići vrlo velika ekspresivnost. Osim samog teksta s informacijom o događaju, moguće je dobiti vrijeme i naziv datoteke s kodom u kojem je zapis nastao. Također, vrijeme od početka izvršavanja programa, ime procesa ili dretve u kojem je zapis nastao su dostupni.

Svakom upisivaču definiranom u konfiguracijskoj datoteci može se pridodjeliti ime zapisivača, instanciranog u programu, za koji će upisivač primati zapise:

```
log4j.rootLogger=DEBUG, root, stdout
log4j.logger.luscacontroller=DEBUG, LuscaController
log4j.logger.luscacontroller.TickThread=DEBUG, TickThread
```

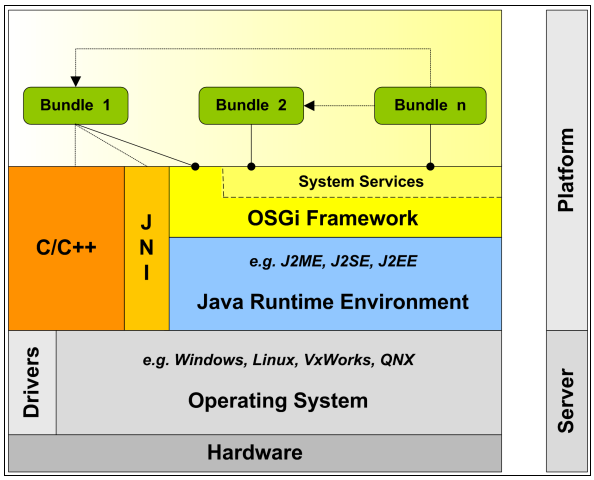
Na ovaj način zapisi, s nivoima DEBUG ili višim, generirani unutar razreda `LuscaController` i `TickThread` prosljeđivati će se istoimenim upisivačima, a zapisi razreda `TickThread`, i upisivaču `LuscaController` zbog definirane hijerarhije među njima. Također, svi upisivači prosljeđivati će zapise glavnom korijenskom zapisivaču, `rootLogger`. Ovo hijerarhijsko zapisivanje se može isključiti ukoliko nije poželjno. Prilikom izdavanja aplikacije za uporabu krajnjem korisniku ili npr. vršenja testova brzine izvođenja aplikacije, može se promijeniti nivo zapisivanja s nivoa *DEBUG* u neki viši ili u potpunosti isključiti zapisivanje jednostavnom promjenom konfiguracijske datoteke. Kako je jedan od glavnih ciljeva i značajki Log4j paketa brzina, mnoge komponente su posebno dodatno optimirane. Osim Jave, Log4j se može koristiti i u drugim jezicima, prenesen je i u jezike: C, C++, C#, Perl, Python, Ruby i Eiffel.

3.3. Eclipse

Eclipse je dio zajednice otvorenog koda čiji su projekti usmjereni k izgradnji otvorenog razvojnog okruženja sastavljenog od nadogradivih komponenti, pomoćnih alata, izvršnih okruženja za izgradnju, distribuciju i upravljanje aplikacijom tijekom njenog životnog ciklusa. Eclipse je izgrađen u programskom jeziku Java što mu omogućuje fleksibilnost i prenosivost. Primarno je Eclipse integrirano razvojno okruženje namjenjeno razvoju Java aplikacija i izgradnju produkata baziranih na Eclipse platformi. Ova platforma, nezavisna od proizvođača, sastoji se od jezgre (*core*) i raznih dodataka (*plugin*). Jezgra pruža usluge za upravljanje dodacima, a dodaci pružaju stvarnu funkcionalnost. Eclipse je svoju uporabu našao kao pomoć pri razvoju velikih poslovnih aplikacija (*Enterprise Development*), pri razvoju ugrađenih, integriranih aplikacija u raznim uređajima (*Embedded and Device Development*), složenih korisničkih aplikacija, platformi (*Rich Client Platform*), pri izradi dodataka kao pomoć drugim aplikacijama, a služi i kao integrirano razvojno okruženje za brojne programske jezike.

Osnova Eclipsa je RCP, *Rich Client Platform*. RCP se sastoji od:

- *Runtime, Core* – jezgra, pokreće Eclipse i dodatke (*plugins*)
- *OSGi* okruženje – okruženje kojim je definiran životni ciklus aplikacije i upravljanje servisima. OSGi tehnologija je univezalni srednji sloj (*middleware*). Omogućava programerima standardizirano okruženje bazirano na servisima, komponentama te standardizirane načine upravljanja životnim ciklusom aplikacija.

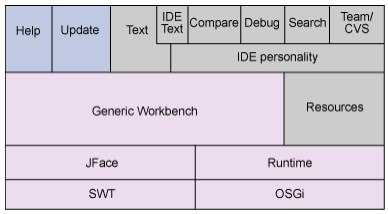


Slika 3.1: OSGi sloj

- *Standard Widget Toolkit* (SWT) – prenosivi skup alata za izradu GUI komponenti, napravljenih na način da koriste izgled grafičkih elemenata ovisno o operativnom sustavu na kojem se koriste.
- *JFace* – upravljanje tekstom, uređivači teksta. Složeniji grafički elementi izgrađeni pomoću SWT-a, omogućuju jednostavnije korištenje korisniku.
- *Eclipse Workbench* – radno mjesto. Omogućuje poglede, uređivače teksta, perspektive, čarobnjake. Eclipse radno mjesto osigurava ujednačeni izgled sučelja i dodacima omogućuje univerzalni razvojni proces tako što im pruža alate za traženje,

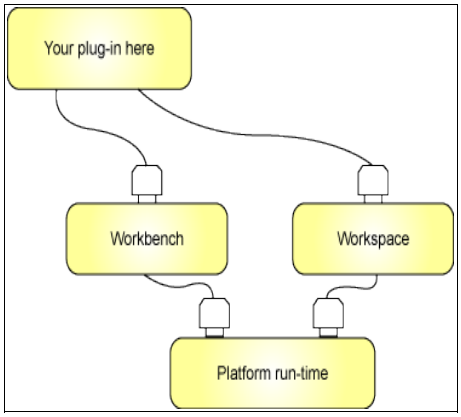
timski rad, jedinstven sustav pomoći, radni prostor u koji se smještaju projekti, datoteke, sustav za ažuriranje dodataka. Eclipse radno mjesto proširuje JDT, pružajući okolinu za Java razvoj, i PDE, za razvoj dodataka.

- *Help* (*User Assistance*) – Eclipse komponenta koja omogućava pružanje savjeta korisnicima.
- Team – Komponenta koja omogućava priključivanje sustava za upravljanje inačicama koda.



Slika 3.2: Struktura Eclipse platforme

3.3.1. Eclipse dodatak



Slika 3.3: Eclipse radno mjesto i okružje, osnovna podrška Eclipse dodacima

Cijela funkcionalnost Eclipse platforme ostvarena je dodacima. Osim male pokretačke (*run-time*) jezgre (*core*), sve ostalo u Eclipseu je dodatak. Svaki dodatak koji korisnik razvije integrira se s Eclipseom na isti način kao i ostali i tretiraju se jednako. Radno mjesto (*Workbench*) i radno okružje (*Workspace*) su također dva Eclipse dodatka koja omogućuju *točke proširenja* ostalim dodacima. Eclipse dodatak potrebuje točku proširenja na koju se može prikvačiti kako bi mogao funkcionirati.

Workbench komponenta sadrži točke proširenja koje omogućuju korisničkim Eclipse dodacima nadogradnju i uporabu Eclipse korisničkog sučelja, uporabu izbornika, alatnih traka, primanje događaja i generiranje pogleda. *Workspace* komponenta sadrži točke proširenja koje omogućuju interakciju drugih dodataka s projektima i datotekama unutar Eclipse IDE-a. Osim ove dvije komponente, koje se mogu nadograđivati drugim dodacima postoji i *Debug* komponenta koja omogućuje nekom dodatku pokretanje programa,

interakciju prilikom izvršavanja, obradu pogrešaka i sve potrebno za izgradnju *debuggera*. Osim *Debug* komponente tu su još *Help* i *Team* komponente. *Team* komponenta omogućuje Eclipse resursima interakciju sa sustavima za upravljanje inačicama (*Version Control Systems, VCS*). Ovisno o zahtjevima koje mora ispoštovati dodatak kojeg korisnik izrađuje, isti će se prikvačiti na odgovarajuće točke proširenja unutar Eclipse platforme.

Najlakši način kreiranja dodataka za Eclipse je uporaba okruženja za razvoj dodataka (*Plugin Development Environment, PDE*), koji sadrži čarobnjake, primjere i uređivače konfiguracijskih datoteka Eclipse dodataka te automatizira cijeli proces izgradnje, ispitivanja i postavljanja izrađenih dodataka.

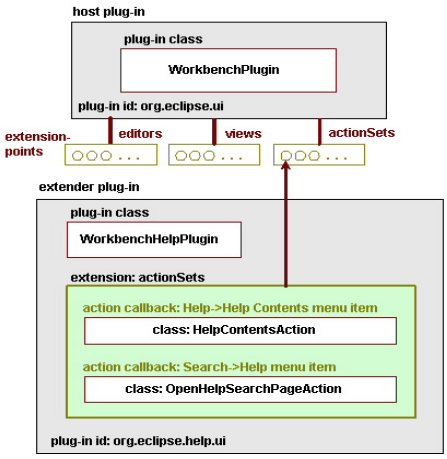
Manifest Eclipse dodatka sadrži detaljne informacije koje će Eclipse iskoristiti pri integraciji plugina u okruženje. Njega čine dvije datoteke `plugin.xml` i `manifest.mf`. Ukoliko koristimo PDE, ove datoteke moći ćemo uređivati uporabom posebnih uređivača, ali se one mogu uređivati bez problema i ručno. Manifest datoteka, njen sadržaj, nadahnut je RFC822 standardom. Osnovna namjena ove datoteke je opis *Java archive*, jar datoteka. Sadržaj je prikazan parovima `ime:vrijednost`. Grupe takvih parova nazivaju se sekcije. Više sekcija međusobno je odvojeno praznim linijama. Uporaba manifest datoteke usklađuje Eclipse još više OSGi modelu. Primjer datoteke `manifest.mf`:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: LuscaController
Bundle-SymbolicName: LuscaController; singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: luscacontroller.Activator
Bundle-Localization: plugin
Require-Bundle: org.eclipse.ui,org.eclipse.core.runtime
Eclipse-LazyStart: true
Bundle-ClassPath: src/log4j-1.2.14.jar, src/jcommon-1.0.10.jar,.
Export-Package: luscacontroller
```

`Bundle-SymbolicName` jednoznačno predstavlja dodatak unutar Eclipse okruženja, a uporabom Java konvencije imenovanja paketa (npr. `com.<ime tvrke>.<ime proizvoda>`) i općenito. Osim imena dodatka, manifestom se definiraju i njegova inačica, ovisnosti dodatka o drugim dodacima i radno okruženje dodatka. Na primjer, `Export-Package` definira koji se dijelovi, paketi dodatka mogu koristiti iz drugih dodataka, odnosno, dostupni su *izvana*. Ukoliko se dodatak izdaje kao jedna cjelovita *jar* arhiva, `Bundle-ClassPath` deklaracija treba biti prazna kako Eclipse ne bi tražio razrede navedene u dodatku izvan njegove arhive. `Require-Bundle` definira koji dodaci unutar Eclipse okruženja moraju biti vidljivi dodatku tijekom njegovog izvršavanja. `Bundle-Vendor` označava ime autora dodatka. Među ostalim, manifest dodatka definira startnu točku pokretanja dodatka (`Bundle-Activator`, razred koji će se prvi pokrenuti pri učitavanju dodatka) *Activator*.

Datotekom `plugin.xml` definiraju se proširenja (*extensions*). Osnovni dodatak, domaćin (*host*), proširuje se svojim nastavkom, *extender*. Bilo koji dodatak može dozvoliti da bude proširen dodavanjem novih procesnih elemenata, nastavaka. Proširenje se definira od strane nastavnog dodatka i uzrokuje da domaćin promijeni svoje ponašanje. Na ovaj način nekom dodatku mogu se dodavati nove mogućnosti ili promijeniti ponašanje već postojećih. Dodatak može dozvoliti da bude unaprijeđen različitim vrstama nastavaka, na primjer, UI

komponenta radnog mjesta (*Workbench*) dozvoljava proširenja svojih izbornika i uređivača. U tom slučaju, nastavni dodatak se mora prilagoditi, biti konforman, skupu konfiguracijskih i ponašajnih zahtjeva. Dodatak koji omogućava svoja daljnja proširenja omogućuje razne vrste priključaka na koje se drugi dodaci, nastavci, mogu prikačiti. Ovi priključci se nazivaju točke proširenja (*extension points*). Svaka točka proširenja dozvoljava neograničen broj nastavaka. Primjer točaka proširenja Eclipse *Workbench UI* dodatka prikazan je na slici 3.4.



Slika 3.4: Primjer točaka proširenja Eclipse Workbench UI dodatka

Primjer datoteke `plugin.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
        <extension
                point="org.eclipse.ui.views">
                <category
                        name="LuscaController"
                        id="LuscaController"
                </category>
                <view
                        name="Lusca View"
                        icon="icons/lusca-small.png"
                        allowMultiple="true"
                        category="LuscaController"

class="luscacontroller.views.LuscaView"
                        id="luscacontroller.views.LuscaView"
                </view>
        </extension>
</plugin>
```

Datoteka opisuje proširenje, nastavak na Eclipse Workbench dodatak, `org.eclipse.ui.views`, koji definira UI komponentu *pogled*. Proširenje na komponentu *pogled* izvršit će dodatak `luscacontroller.views.Luscaview`. Opcija `allowMultiple="true"` označava mogućnost instanciranja više od jednog objekta

`LuscaView` istovremeno, odnosno postojanje više objekata tog tipa istovremeno unutar Eclipse okruženja. Nastavak `LuscaView` mora, da bi mogao djelovati, zadovoljiti implementacijom specifičnosti točke proširenja *pogled*. Ukoliko ne implementira sva potrebna svojstva, sučelje se neće moći pokrenuti, koristiti unutar Eclipse radnog okruženja koje će javiti grešku.

3.4. Subversion

Subversion je sustav za kontrolu, upravljanje inačicama datoteka. Subversion upravlja datotekama i direktorijima i promjenama učinjenim nad njima. Upravljanje omogućava vraćanje starijih inačica podataka, pregled povijesti izmjena i načina na koji su podatci mijenjani. Sustav za upravljanje inačicama (*Version Control System, VCS*) mnogi doživljavaju kao jednu vrstu vremenskog stroja. Subversion se može koristiti u mrežnim okruženjima, koji omogućuju uporabu istog od strane više korisnika na različitim, udaljenim računalima. S obzirom da se sve promjene na podatcima vrše na jednom mjestu, lako se mogu integrirati u podatkovni skup, a u slučaju potrebe promjene i odbaciti. S obzirom da Subversion može upravljati kolekcijom datoteka, koristi ga se za upravljanje inačicama izvornog koda neke aplikacije i samim time za razvoj aplikacija od strane programerskog tima. Razvoj Subversiona je počeo 2000. godine od tvrtke CollabNet kao pokušaj zamjene dotad postojećeg VCS-a CVS (*Concurrent Version System*) zbog njegovih nedostataka u dizajnu. Također, Subversion je trebao zamijeniti CVS na način da se krajnji korisnici, s iskustvom korištenja CVS-a, mogu lako prilagoditi Subversionu. Stoga je Subversion zadržao veći dio CVS terminologije i funkcija.

Subversion omogućuje:

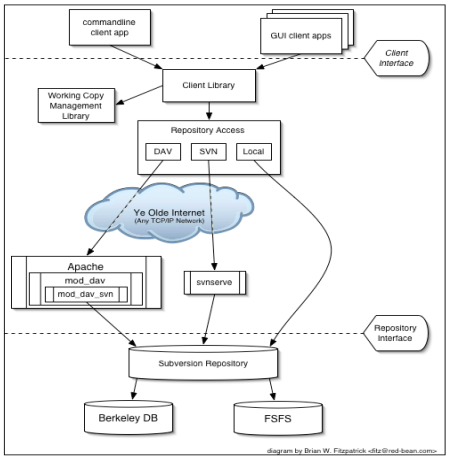
- Dodjeljivanje inačica direktorijima, verzioniranje – CVS prati povijest izmjena individualnih datoteka, dok Subversion implementira virtualni datotečni sustav koji prati izmjene nad cijelim stablom direktorija kroz povijest. Datoteke i direktoriji su verzionirani.
- Pravu povijest inačica – CVS ne podržava preimenovanje i kopije datoteka, također ne može se zamijeniti već verzionirana datoteka nekom novom, istoimenom, bez da preuzme cijelu povijest prošle datoteke. Subversion omogućuje dodavanje, brisanje, kopiranje, preimenovanje datoteka i direktorija. Svakoj novo dodanoj datoteci pridodaje se čista, nova povijest izmjena.
- Nedjeljivo potvrđivanje – kolekcija modifikacija se u cijelosti dodaje repozitoriju ili uopće ne. Ova mogućnost omogućuje korisnicima da više promjena pošalju u obliku logičkih segmenata i sprječava moguće parcijalno izvršavanje promjena što može dovesti do konfuzije.
- Verzionirani *meta* podatci – Svaka datoteka i direktorij mogu posjedovati skup svojstava. Svojstva također mogu biti verzionirana kroz povijest, kao i sam sadržaj istih.
- Odabir mrežnih slojeva – pristup Subversion repozitorijima opisan je na apstraktni način što omogućuje implementaciju novih mrežnih mehanizama i protokola za pristup Subversion poslužitelju. Subversion se može uključiti u sklopu Apache HTTP poslužitelja kao modul (*mod_dav_svn*). Ovo svojstvo daje Subversionu prednost u stabilnosti i međuoperativnosti s postojećim svojstvima HTTP poslužitelja:

autentikaciju, autorizaciju i kompresiju podataka. Osim HTTP poslužitelja, dostupan je i samostalni Subversion poslužitelj. Ovaj poslužitelj koristi se specifičnim protokolom koji se može tunelirati uporabom SSH (*Secured Shell*) protokola.

- Konzistentno upravljanje podatcima – Subversion izražava razlike u datotekama binarnim razlikovnim algoritmom, koji se jednako može primijeniti na tekstualnim i binarnim datotekama. Obje vrste datoteka su smještene u repozitorij sažete, a razlike se šalju u oba smjera mrežom.

3.4.1. Arhitektura Subversiona

Slika 3.5 prikazuje arhitekturu Subversion sustava za upravljanjem inačicama podataka. Na jednom kraju nalazi se Subversion repozitorij koji sadrži sve verzionirane podatke, a na drugom kraju nalazi se Subversion korisnički program koji upravlja lokalnim preslikama verzioniranih podataka, radnim kopijama (*Working copies*). Podatci u Subversion repozitoriju mogu biti smješteni u Berkeley DB ili u FSFS implementaciju datotečnog sustava. FSFS je Subversion alternativa Berkeley DB implementaciji datotečnog sustava.



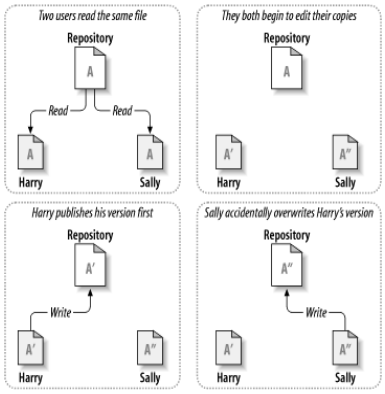
Slika 3.5: Arhitektura Subversiona

3.4.2. Repozitorij

Jezgra Subversiona je repozitorij, koji predstavlja centralno spremište podataka. Repozitorij čuva podatke u obliku stabla datoteka, uobičajenog pristupa prikaza odnosa datoteka i direktorija. Bilo koji broj klijenata može se spojiti na repozitorij, čitati i pisati u datoteke koje se tamo nalaze. Repozitorij je vrsta datotečnog poslužitelja koja pamti svaku promjenu učinjenu nad datotekama. Korisnik može pročitati podatke iz repozitorija, ali također može zatražiti i vidjeti sve izmjene izvršene nad datotečnim sustavom.

3.4.3. Problem dijeljenja datoteka

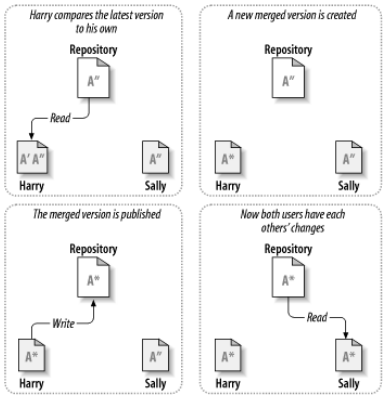
Svi sustavi za upravljanje inačicama podataka moraju riješiti osnovni problem: Kako će sustav dozvoliti korisnicima razmjenu informacija, sprječavajući ih pri tom da istovremeno izvrše izmjenu istih podataka ili dijelova datoteka? Na slici 3.6 primjetno je da su korisnici Harry i Sally istovremeno izvršili promjene nad datotečnim sustavom u repozitoriju, te možebitno uništili izmjene koje je onaj drugi korisnik učinio. Najjednostavnije rješenje ovog



Slika 3.6: Primjer istovremene izmjene podataka

problema je da Harry zaključa repozitorij ili datoteku koju želi mijenjati sve dok ne izvrši i upiše sve izmjene u repozitorij te ga potom otključa. Ovakav pristup je vrlo nepoželjan te onemogućuje ostale korisnike u radu s repozitorijem, time se gubi vrijeme te nije u duhu timskog rada.

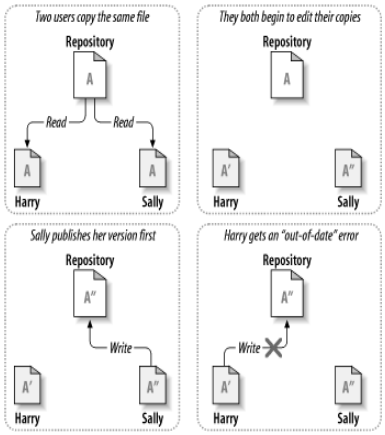
Subversion i CVS koriste *kopiraj-izmjeni-spoji* (*copy-modify-merge*) model kao alternativu zaključavanju (slike 3.7 i 3.8). U ovom modelu svaki korisnik kreira osobnu radnu kopiju (*Working copy*), presliku repozitorija. Lokalna kopija omogućava korisnicima da samostalno i neovisno rade na projektima. Naposljetku, sve lokalne kopije spajaju se u novu, konačnu verziju. VCS pomaže pri spajanju radnih kopija, ali u konačnici, korisnici su odgovorni da se taj proces uspješno okonča.



Slika 3.7: copy-modify-merge

Oba korisnika (slika 3.7) preuzimaju posljednju verziju datoteke iz repozitorija te počinju uređivati svoje radne kopije. Sally upisuje svoje izmjene prva. Harry pri pokušaju upisivanja svojih izmjena dobiva informaciju o pogrešci (*out-of-date*), koja označava da njegova radna kopija sadrži zastarjelu verziju podataka. Rješenje se nudi na slici 3.8.

Usporedbom nove verzije iz repozitorija i svoje radne kopije s izvršenim izmjenama, nova *spojena* verzija je generirana uz korisnikovu intervenciju pri validiranju procesa spajanja (ukoliko su izmjene na neki način u konfliktu). Nova spojena verzija potvrđuje se u repozitorij te oba korisnika mogu pročitati prave, ispravne podatke. Princip *copy-modify-*

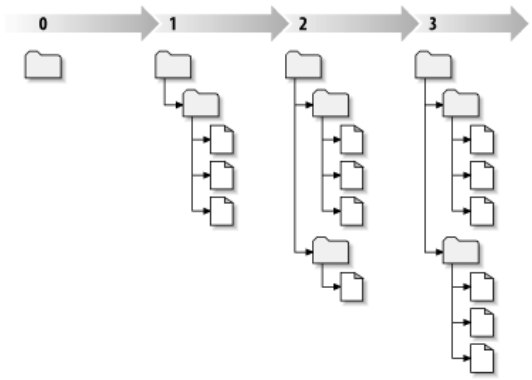


Slika 3.8: copy-modify-merge, rješenje

merge u praksi djeluje vrlo dobro. Korisnici istovremeno upotrebljavaju repozitorij i vrše izmjene nad podatcima. U slučaju da korisnici rade nad istim datotekama, uglavnom se događa da izmjene koje vrše nisu u konfliktu s ostalima, no ipak, najvažnije je pri radu da korisnici međusobno komuniciraju jer u slučaju loše komunikacije vjerojatnost nastanka problema se značajno povećava. U slučaju rada s binarnim datotekama, često je nemoguće spojiti dvije inačice u jednu, pa se u tim slučajevim koristi zaključavanje datoteka.

3.4.4. Revizije

Svaka operacija potvrđivanja (*commit*) radne kopije u repozitorij, s izvršenim izmjenama, izvodi se kao jedna nedjeljiva operacija. Svaki put kada repozitorij prihvati potvrđivanje, izmjenu, kreira se novo datotečno stablo, revizija. Svakoj novoj reviziji zadan je prirodni broj, za jedan veći od prethodne revizije. Inicijalna revizija, bez podataka, numerirana je nulom (slika 3.9).



Slika 3.9: Repozitorij

Subversion dodjeljuje broj revizije cijelom datotečnom stablu, a ne individualnim datotekama.

3.4.5. Diff – razlike između datoteka

Subversion alatkom *diff* može generirati datoteku koja sadrži opis razlika između datoteka, odnosno revizija repozitorija. Diff (*difference between files*) omogućuje korisniku da detaljno ispita razlike između dvije revizije repozitorija, ili radne kopije i neke od revizija koje se nalaze u repozitoriju. Diff datoteke generirane su u unificiranom diff formatu (*unified diff format*) i mogu služiti kao ulaz u program *patch.* Program *patch* može primijeniti opisane razlike diff datotekom nad datotečnim stablom (npr. radnom kopijom) i na taj način modificirati njegov sadržaj. Umjesto da korisnik preuzima cijelu najnoviju reviziju nekog koda, ili ručno zamjenjuje stare datoteke novima kojih može biti mnogo, program *patch* automatizira cijeli taj proces. Osim toga, nije potrebno prebacivati mrežom ili nekim drugim medijima veće količine informacija, već samo razlike. Diff alatka nastala je ranih 70. godina na Unix operativnom sustavu. Konačnu inačicu, izdanu uz 5. verziju Unixa 1974. godine, u cijelosti je napisao Douglas Mcllroy, a 1976. godine, zajedno s James W. Huntom koji je razvio inicijalni prototip, službeno je objavio svoje istraživanje.

Diff algoritam je baziran na rješavanju problema najveće zajedničke podsekvence niza (*Longest Common Subsequence*).Ukoliko imamo dvije sekvence elemenata (Tablica 3.1):

a	*b*	*c*	*d*	*f*	*g*	*h*	*j*	*q*	*z*				
a	*b*	*c*	*d*	*e*	*f*	*g*	*i*	*j*	*k*	*r*	*x*	*y*	*z*

Tablica 3.1: diff primjer (1)

želimo naći najdulju sekvencu elemenata koji su zastupljeni u oba niza u jednakom redoslijedu. Tražimo novu sekvencu koja se može dobiti iz prve sekvence brisanjem elemenata, a iz drugog niza brisanjem drugih elemenata. Dobivena sekvenca prikazana je tablicom 3.2.

a	*b*	*c*	*d*	*f*	*g*	*j*	*z*

Tablica 3.2: diff primjer (2)

Konačan Diff izlaz prikazan je tablicom 3.3.

e	*h*	*i*	*k*	*q*	*r*	*x*	*y*
+	-	+	+	-	+	+	+

Tablica 3.3: diff primjer (3)

Postoji više formata, izgleda diff datoteke, unificirani format nasljeđuje tehnička unaprijeđenja kontekstualnog formata, ali generira manju i lakše čitljivu diff datoteku.

Format datoteke započinje s tri znaka minus “---” koja označavaju originalnu datoteku i tri znaka “+++” koja označavaju novu datoteku u drugoj liniji. Nakon ove dvije linije u datoteci slijede segmenti promjena koji sadrže opise razlika u datotekama. Segmenti promjena sastavljeni su od linija koda s umetnutim opisnim karakterima. Ukoliko su linije koda nepromijenjene prethodi im razmak, ukoliko su linije dodane prethodi im “+” karakter te “-” ukoliko su uklonjene. Segment počinje s informacijom o svojem rasponu unutar datoteke:

```
@@ -R +R @@
```

Raspon segmenta *R*, kojemu prethodi znak “-” označava raspon segmenta u originalnoj datoteci, a raspon segmenta nove datoteke označen je znakom “+”. Svaki raspon segmenta prikazan je u formatu *l,s* gdje je *l* početni broj linije, a *s* broj linija na koje se segment odnosi u datoteci. Ukoliko veličina segmenta ne odgovara broju linija u datoteci znači da je diff datoteka neispravna.

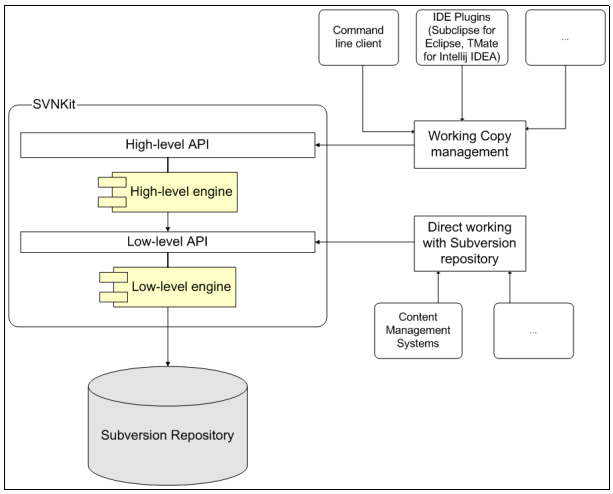
Primjer diff datoteke:

```
--- bar.c       (revision 3)
+++ bar.c       (working copy)
@@ -1,7 +1,12 @@
+#include <sys/types.h>
+#include <sys/stat.h>
+#include <unistd.h>
+
+#include <stdio.h>

 int main(void) {
-  printf("Sixty-four slices of American Cheese...\n");
+  printf("Sixty-five slices of American Cheese...\n");
 return 0;
 }
```

3.5. SVNKit

SVNKit je Java biblioteka koja omogućuje klijent pristup Subversion repozitorijima iz korisničkih Java aplikacija. Struktura biblioteke prikazana je na slici 3.10.



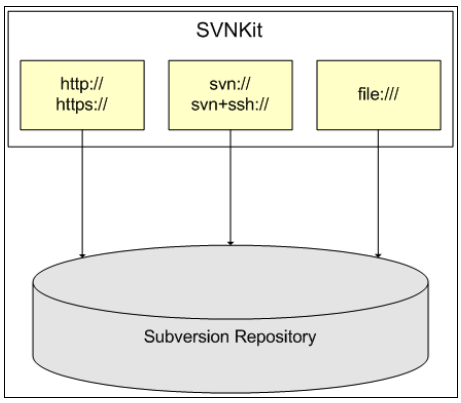
Slika 3.10: Arhitektura SVNKit biblioteke

SVNKit je integralni dio *Subclipse* dodatka za Eclipse IDE koji omogućuje korisnicima timski rad pri razvoju aplikacija iz Eclipse okruženja. SVNKit je izveden u dvije osnovne razine:

- Visoka razina (*high-level layer*) upravlja radnim kopijama. Odgovarajući API omogućuje upravljanje radnim kopijama na isti način kao i uporabom neke Subversion klijent aplikacije.
- Implementacija niske razine (*low-level layer*) slična je protokolu koji koristi Subversion za pristup svojim repozitorijima, predstavlja prilagodnik (*driver*) koji omogućuje direktan rad sa Subversion repozitorijem.

Visoka razina SVNKit biblioteke, slična je naredbama koje Subversion poslužitelju upućuje bilo koji Subversion komandno-linijski klijent. Sve operacije za upravljanje radnim kopijama logički su podijeljene u različite *SVN*Client* razrede. Uporabom, na primjer, *SVNUpdateClient* razreda možemo izvršiti operacije *check out, update* i slične. Istinsko izvršavanje operacija obavit će se uporabom niske razine.

Niska razina predstavlja, apstraktno, protokol pristupa Subversion repozitoriju te može pristupiti Subversion poslužitelju uporabom različitih protokola (slika 3.11) koje i implementira. Način uporabe protokola i primjer korištenja prikazan je u poglavlju 4.5.7.



Slika 3.11: Protokoli kojima SVNKit može pristupiti Subversion repozitoriju (niska razina)

3.6. JSON

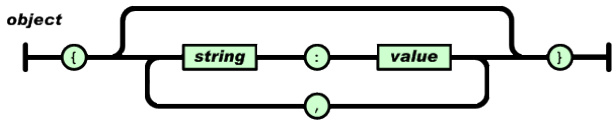
JSON (*JavaSctipt Object Notation*) je laki format razmjene podataka. Lako se čita i piše od strane ljudi i parsira od strane računalnih programa. Baziran je na podskupu *JavaScript* programskog jezika, *Standard ECMA-262 3rd Edition 1999*. JSON je tekstualni format potpuno neovisan o jeziku ali koristi konvencije bliske programerima C-obitelji jezika, uključivši C, C++, C#, Java, JavaScript, Perl, Python i mnoge druge. Ova svojstva čine JSON idealnim jezikom za razmjenu podataka. Implementacije biblioteka funkcija, koje omogućuju olakšano upravljanje ovim podatkovnim formatom, mogu se pronaći za velik broj programskih jezika, pa je prenosivost ovog formata i jednostavnost uporabe vrlo velika.

JSON je izgrađen s dvije strukture:

- Skup parova *ime*:*vrijednost*. U mnogim jezicima ovo se smatra zapisom, objektom, strukturom, rječnikom, *hash* tablicom, asocijativnim poljem.
- Uređenom listom vrijednosti. U mnogim jezicima smatra se poljem, vektorom, listom, sekvencom.

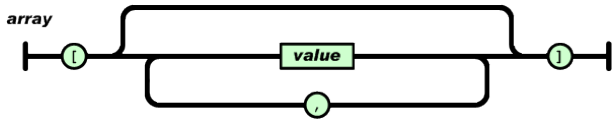
Ovo su univerzalne podatkovne strukture. Svaki moderni programski jezik podržava ove strukture, pa je logično da podatkovni format bude baziran na njima.

JSON objekt (slika 3.12) je neuređen skup parova *ime*:*vrijednost*. Objekt počinje otvorenom vitičastom zagradom '{', a završava zatvorenom vitičastom zagradom '}'. Parovi ime/vrijednost odvojeni su zarezom ','. Vrijednost, identificirana imenom može biti drugi JSON objekt, polje, tekst, broj, logička vrijednost.



Slika 3.12: JSON objekt, sintaksa

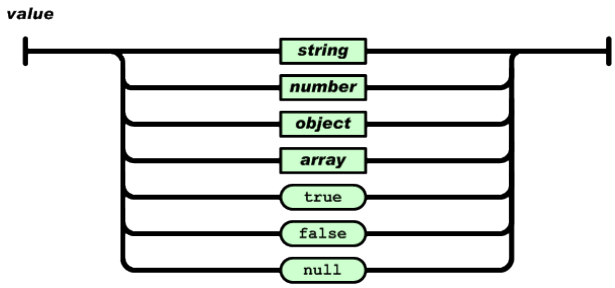
JSON polje (slika 3.13) je skup uređenih vrijednosti. Polje započinje otvorenom uglatom zagradom, '[', a završava zatvorenom uglatom zagradom, ']'. Vrijednosti su odvojene zarezima. Svaka vrijednost polja može biti JSON objekt, polje, broj, tekst, logička vrijednost.



Slika 3.13: JSON polje

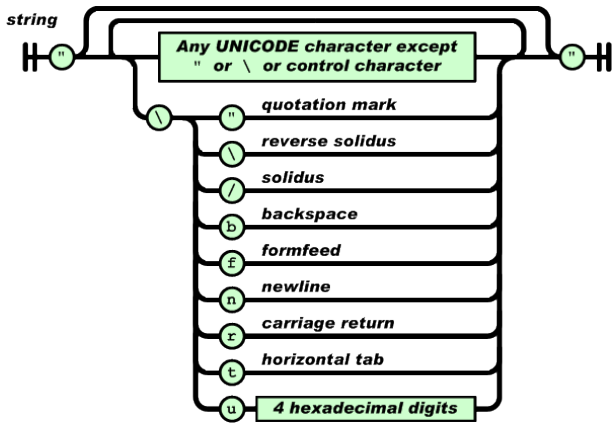
Vrijednost unutar JSON polja može biti *string* omeđen navodnim znakovima, '"', broj, logička vrijednost, ili neki drugi objekt ili polje. Ove strukuture mogu biti ugnježđene.

JSON vrijednost može biti *string*, broj, objekt, polje, logička vrijednost ili *null*, ništa (slika 3.14).



Slika 3.14: JSON vrijednost, tipovi

JSON tekst (*string*) može biti bilo koji *Unicode* znak osim ASCII kontrolnih znakova te znaka '"'. Ukoliko se ove znakove želi koristiti potrebno je ispred njih umetnuti *backslash* znak "\" (slika 3.15).



Slika 3.15: JSON String, sintaksa

4. Lusca Controller

Lusca Controller je dodatak za razvojno okruženje Eclipse. Implementacija Lusca Controllera unutar Eclipse okruženja omogućuje istovremen rad na aplikaciji uporabom Eclipse IDE okruženja i ispitivanje pomoću Lusca ispitnog okruženja. Lusca Controller implementiran je kao proširenje na Eclipse Workbench UI, odnosno na točku proširenja pogled, `org.eclipse.ui.views`. Lusca Controller omogućuje upravljanje Lusca ispitnim okruženjem uporabom Luscinog ispitnog sučelja. Interakcija s Luscom odvija se u osnovi HTTP protokolom. S obzirom da je riječ o korisničkoj aplikaciji, Controller je osmišljen kao inicijator konekcija na Luscu. Lusca ne može samoinicijativno pristupiti Controlleru, već samo odgovarati na upite i predavati zatražene informacije. Ovakav pristup olakšava konfiguraciju sigurnosnih postavki mreže u kojoj se nalazi Lusca ispitno okruženje, pa ukoliko se Lusca Controller nalazi izvan interne mreže, administrator mreže mora samo omogućiti pristup Lusca pokretaču ispitnog okruženja, odnosno računalu i portu na kojem Lusca ispitno sučelje očekuje zahtjeve. Nadalje, u slučaju višekorisničkog rada, potrebno je na Lusci izvršiti sinkronizaciju pristiglih zahtjeva, kako ne bi došlo do nekonzistencija u radu.

4.1. Kratki pregled mogućnosti Lusca Controllera

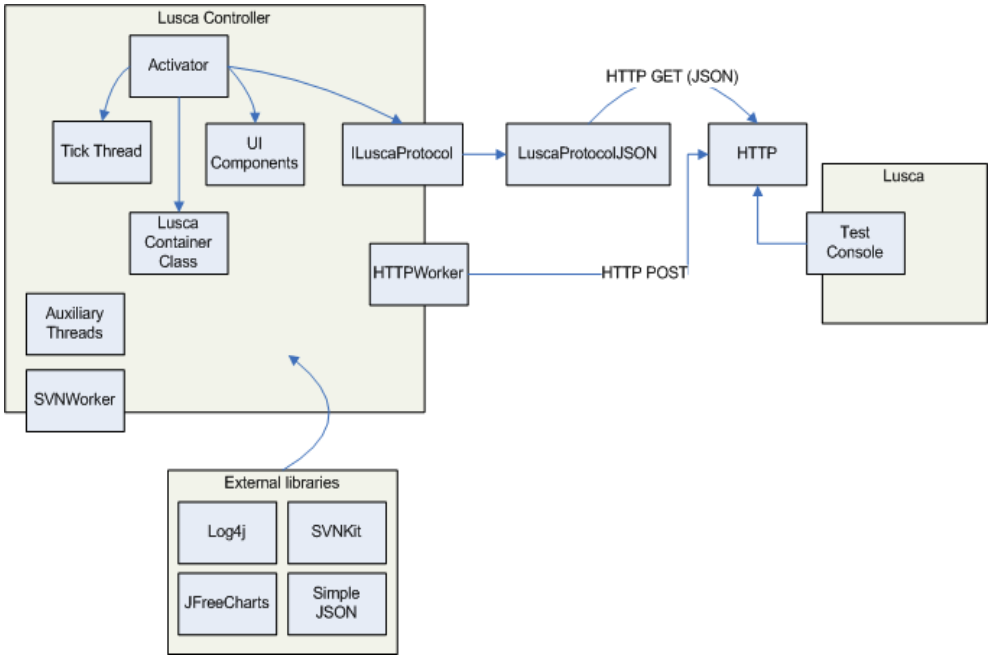
Lusca Controller omogućuje uporabu kompletnog sučelja *Test Console* uz laku nadogradnju dodatnim mogućnostima u budućnosti.

- **registracija Lusca Controllera na Luscin pokretač ispitnog okruženja** – slanjem inicijalnog zahtjeva Lusca odgovara statusom pokretača ispitnog okruženja te može pripremiti sve potrebno za daljnju komunikaciju s Lusca Controllerom. Ova registracija može uključivati i neku vrstu identifikacije upravljačkog programa u budućim implementacijama.
- **dohvat agenata** – na ovaj zahtjev, Lusca odgovara popisom svih dostupnih, definiranih Lusca agenata s pripadajućim statusom, odnosnom informacijom o njihovoj upotrebljivosti. Svaki agent identificiran je svojim jedinstvenim ključem uz pripadajući status. Status agenta može biti spreman, *ready* ili nedostupan, *down*.
- **dohvat ispitnih paketa –** dohvat svih ispitnih paketa definiranih unutar Lusce, s pripadajućim statusom. Ispitni paketi su identificirani svojim jedinstvenim ključem, uz pripadajući status.
- **dohvat aplikacija** – Lusca odgovara informacijama o svih definiranim aplikacijama.
- **dohvat statusa** – dohvat statusa svih komponenti unutar Lusca ispitnog okruženja. To se odnosi na raspoložive agente, ispitne pakete i aplikacije
- **dohvat informacija o zauzeću memorije** – informacije o količini zauzete i slobodne memorije na pokretaču ispitnog okruženja i posebno, za svakog Lusca agenta.
- **informacije o zauzeću procesora** – informacije o opterećenju procesora, izražene postocima, za pokretač ispitnog okruženja i za svakog Lusca agenta.
- **informacije o procesoru** – prikaz osnovnih informacija o procesoru i računalu na kojemu se nalazi pokretač ispitnog okruženja i posebno, za svakog Lusca agenta.

- **dohvat popisa pokrenutih procesa** – prikaz svih pokrenutih procesa za svako računalo unutar Lusca ispitnog okruženja.
- **dodavanje, brisanje, ažuriranje aplikacija** – mogućnost dodavanja novih aplikacija koje će sudjelovati u ispitivanju, te ažuriranje već postojećih. Za Luscu aplikacija je definirana svojim jedinstvenim ključem, imenom, opisom, web stranicom i, najvažnije, Subversion repozitorijem. Svaki korisnik, zasebno, može upisati svoje korisničko ime i lozinku za pristup aplikaciji u Subversion repozitoriju.
- **dodavanje, brisanje, ažuriranje radnih kopija** – Za svaku aplikaciju korisnik može definirati jednu ili više radnih kopija, pa onda prilikom pokretanja ispitnih paketa, odabrati način na koji će se, ukoliko je potrebno, kreirati zakrpe za definirane aplikacije u ispitnom paketu.
- **dodavanje, brisanje, dohvat i (ažuriranje) ispitnih paketa** – uporabom uređivača ispitnih paketa može se dohvaćeni ispitni paket s Lusce ažurirati. Također, može se dodati novi ispitni paket, kreiran uređivačem ispitnih paketa ili učitan, lokalno, s računala. Ažuriranje ispitnih paketa se vrši preuzimanjem paketa s Lusce, te njegovim uređivanjem, lokalno, nakon čega slijedi prijenos uređenog ispitnog paketa natrag Lusci.
- **pokretanje ispitnih paketa s odabirom ispitnih slučajeva** – pri svakom pokretanju ispitnog paketa potrebno je odabrati sve ispitne slučajeve, koji su definirani u ispitnom paketu, a želimo ih provesti našim ispitivanjem. Pored toga, za sve aplikacije definirane ispitnim paketom može se definirati i poslati zakrpa, *patch* u obliku *diff* datoteke (poglavlje 3.4.5). Zakrpa se može učitati direktno iz datoteke ili generirati uporabom Subversion alata. Zakrpa se može generirati kao razlika u revizijama iste aplikacije, definirane Subversion repozitorijem, ili radne kopije (ili više njih) i neke od revizija aplikacije iz repozitorija. Po završetku provođenja ispitivanja, ispisuje se informacija o uspješnosti ili neuspješnosti izvršavanja testova, odnosno, ispitni paket je uspješno izvršen ukoliko ispitivane aplikacije zadovolje sve ispitne slučajeve (koje smo odabrali pri pokretanju). Svaka konfiguracija pokretanja ispitnog paketa može se spremiti kao ispitni profil, pa pri slijedećem pokretanju, ukoliko ne želimo vršiti nove promjene, možemo direktno pokrenuti prethodno spremljeni ispitni profil. Svaki ispitni paket može imati nula ili više ispitnih profila.
- **zaustavljanje ispitnih paketa** – izvršavanje svakog pokrenutog ispitnog paketa može se pokušati zaustaviti, a ukoliko se operacija uspješno izvrši, biti će naznačeno.
- **zahtjev za pregledom ispitnih zapisa –** ukoliko je ispitivanje u tijeku, može se predati zahtjev Lusci za pregledom izlaza koji test generira. U novom pogledu, u Eclipse okruženju, ispisivat će se zapisi ispitnog paketa, koji će se, tijekom izvršavanja ispitnog paketa, dohvaćati s Lusce. Zapisi su označeni svojim rednim brojem, opisom i vrstom. Mogu se filtrirati po vrsti, a dostupna je i pretraga zapisa po opisu i broju. Prikupljeni zapisi mogu se izvesti u CSV (*Comma Separated Values*) datoteku.
- **zahtjev za dohvatom rezultata ispitivanja** – u bilo kojem trenutku korisnik može zatražiti od Lusce rezultate provedbe ispitivanja, za svaki definirani ispitni paket. Lusca odgovara datotekom, arhivom, u kojoj se nalaze sve datoteke generirane pri pripremi i izvršavanju ispitivanja.

4.2. Arhitektura Lusca Controllera

Osnovna arhitektura Lusca Controllera prikazana je na slici 4.1.



Slika 4.1: Arhitektura Lusca Controllera

Prvi razred, instanciran od strane Eclipse platforme, prema specifikaciji danoj manifestom dodatka (poglavlje 3.3.1), *Activator*, zadužen je za inicijalizaciju svih ostalih glavnih komponti Lusca Controllera i pripremu okoline. Najvažnije komponente koje su sadržane u dodatku su:

- *Lusca Container Class* – razred koji sadrži sve objekte koji opisuju Lusca ispitno okruženje. To su korisnički definirani Lusca pokretači ispitnog okruženja, agenti, ispitni paketi, ispitni profili. Uporabom sinkroniziranih metoda razreda *Lusca Container*, svi se oni mogu dohvaćati i administrirati.
- *UI Components* – Najvažnija UI komponenta je *pogled* dodatka (*Lusca View*). Lusca View omogućava korisniku pregled i zadavanje operacija, u stvari, uporabu dodatka. Glavni element *pogleda* je *Tree Viewer* komponenta koja svojom stablastom strukturom prikazuje hijerarhijski sve potrebne elemente Lusce s pripadajućim operacijama u vidu izbornika. Svakom promjenom izvršenom uporabom Lusca Contrainer razreda, mora se osvježiti stablo, posebnom UI dretvom, s obzirom da, zbog Eclipse arhitekture, obične dretve nemaju pravo pristupa UI komponentama.
- *Tick Thread* - dretva koja u određenim vremenskim intervalima osvježava stanje elemenata *Lusca Container* razreda te njihovog prikaza u stablu *pogleda*. Dretva osvježava samo stanje elemenata Lusce, odnosno pokretača ispitnog okruženja koji je trenutno označen aktivan (*enabled*).
- *ILuscaProtocol* – razred kojim je opisano sučelje prema *Test Console* protokolu.

- *HTTPWorker* – s obzirom da Lusca prijenos datoteka vrši ovim protokolom, *HTTPWorker* implementira metode koje obavljaju ove operacije.
- *SVNWorker* – pomoćni razred koji, uporabom *SVNKit* Subversion alata, omogućuje pristup dodatku Subversion repozitorijima i radnim kopijama.
- *External libraries* – neophodne komponente koje se moraju integrirati u dodatak da bi mogao funkcionirati su:
    - *SimpleJSON* – pomoćni alat za parsiranje nekog ulaza s podatcima u JSON formatu te generiranje ispravno formatiranih izlaza.
    - *JFreeChart* – alat za iscrtavanja dijagrama. Potrebno je preuzeti najnoviju verziju biblioteke, s obzirom da je tek odnedavna podržan SWT za iscrtavanje dijagrama. Ukoliko se koristi stara verzija alata, može se iskoristi *SWT_AWT* most (*bridge*), međutim isti nije u cijelosti implementiran za Linux operativni sustav.
    - *SVNKit* – implementacija klijent biblioteke funkcija za korištenje Subversion repozitorijima i radnim kopijama.

4.3. Izvedba komunikacije s Luscom

Na slici 4.1 prikazan je način na koji Lusca Controller može komunicirati s pokretačem ispitnog okruženja, odnosno njegovim *Test Console* sučeljem. Lusca Controller šalje HTTP GET zahtjeve, a prima podatke u JSON formatu. Servis za takav protokol Lusca omogućuje na adresi:

```
http://<lusca_host>:<port>/ConsoleProtocol/http/json
```

Ovaj protokol implementiran je razredom *LuscaProtocolJSON*, prema specifikaciji koja je dana apstraktnim razredom *ILuscaProtocol*. Ukoliko se želi promijeniti način komunikacije između Lusca Controllera i Lusce potrebno je samo napraviti novu implementaciju sučelja. Pri tome, ostali dijelovi Lusca Controllera ostaju netaknuti. Prijenos datoteka je, kao i između pokretača ispitnog okruženja i agenata, izveden uporabom HTTP protokola, metodom POST. Protokol za razmjenu datoteka Lusca omogućuje na adresi:

```
http://<lusca_host>:<port>/ConsoleProtocol/http
```

Svaku naredbu Lusca Contoller šalje Lusci asinkrono, uporabom posebnih dretvi unutar Eclipse okruženja, *Jobs*. Kada se neki *Job* izvrši, poziva se njegova metoda *done* sa događajem *Event,* koji opisuje pokrenuti posao, te objektom *Status,* kojim se opisuje uspješnost njegovog obavljanja. Korisnik kreira novi posao uporabom razreda *JobCommandProtocol*, koji uporabom *Java Reflection* / RTTI (*Run Time Type Identification*) tehnologije poziva odgovarajuću metodu *ILuscaProtocola*, te prosljeđuje povratnu vrijednost metode upakiranu *Hashtable* strukturom. Struktura povratne vrijednosti odgovara stukturi JSON objekta, pa je jednostavno dodavati i mijenjati naredbe, kako na Lusci tako i *LuscaProtocolJSON* razredu. Ukoliko nastane nekakva iznimka ili greška pri pozivu metode, npr. ukoliko je Lusca nedostupna, informacija o iznimki se šalje natrag

JobCommandProtocolom do korisnika u objektu *Status*, pa se može prikazati porukom na UI sučelju.

4.4. Integracija Lusca Controllera u Eclipse IDE

Potrebne komponente za rad s Lusca Controllerom su Eclipse IDE, preporučena verzija 3.2 ili viša te Java Runtime Environment 1.5.

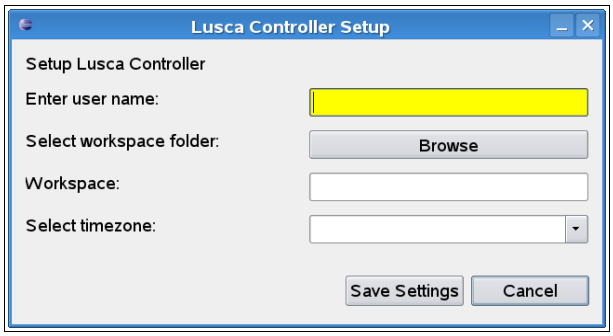
Nakon preuzimanja dodatka za Eclipse, u obliku arhive, `LuscaController_1.0.0.tar.gz` potrebno je istu otpakirati u mapu u kojoj se nalaze svi Eclipseovi dodaci. Unutar arhive nalaze se sve potrebne komponente koje su potrebne za rad dodatka. U Linux operativnom okruženju mapa sa svim dodacima za Eclipse nalazi se u direktoriju `/usr/share/eclipse/plugins`. Nakon toga, na računalu moramo imati mapu s dodatkom:

```
/usr/share/eclipse/plugins/LuscaController_1.0.0
```

Kako bi mogli koristiti dodatak, potrebno je ponovno pokrenuti Eclipse IDE.

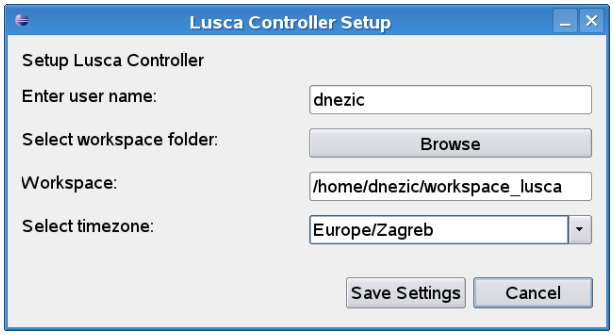
4.5. Korištenje Lusca Controllera

Prilikom prvog pokretanja Eclipsea, nakon integracije dodatka, potrebno je izvršiti podešavanje. Korisniku je prikazan izbornik *Lusca Controller Setup*, slika 4.2.



Slika 4.2: Inicijalno podešavanje

U izborniku je potrebno upisati ime korisnika, *user name*, koji će identificirati osobu koja će Lusci zadavati naredbe. Kasnije, ovim imenom, biti će naznačeno ime korisnika koji je pokrenuo ili zaustavio izvođenje nekog ispitnog paketa. Potrebno je odabrati i radnu mapu, radni prostor *Workspace*. Lusca Controller u radnu mapu spremati će sve informacije o svim definiranim Lusca pokretačima ispitnog okruženja. U istu mapu spremati će se svi preuzeti rezultati ispitivanja, kao i generirane zakrpe te preuzeti ispitni paketi. Informacije o korisničkim ispitnim profilima i radnim kopijama također će se nalaziti u ovoj mapi. Odabir vremenske zone važan je za ispravno prikazivanje vremena generiranja zapisa, s obzirom da se Luscin pokretač ispitnog okruženja može nalaziti u nekoj drugoj vremenskoj zoni u odnosu na korisnika. U padajućem izborniku, korisnik može izabrati jednu od definiranih vremenskih zona. Primjer ispunjenog izbornika prikazan je na slici 4.3.



Slika 4.3: Primjer postavki Controllera

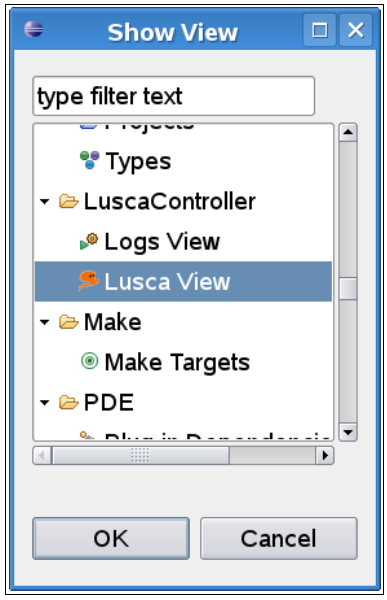
Nakon odabira opcije *Save Settings*, Lusca Controller će ove postavke spremiti u obliku datoteke u mapi koja pripada djelatnom (*run time*) okruženju ovog dodatka, što u Linux operativnom sustavu može izgledati otprilike:

```
/home/user/
runtime-EclipseApplication/.metadata/.plugins/LuscaController
```

U ovoj mapi nalazi se datoteka sa spremljenim postavkama, `lusca_controller.conf` slijedećeg sadržaja:

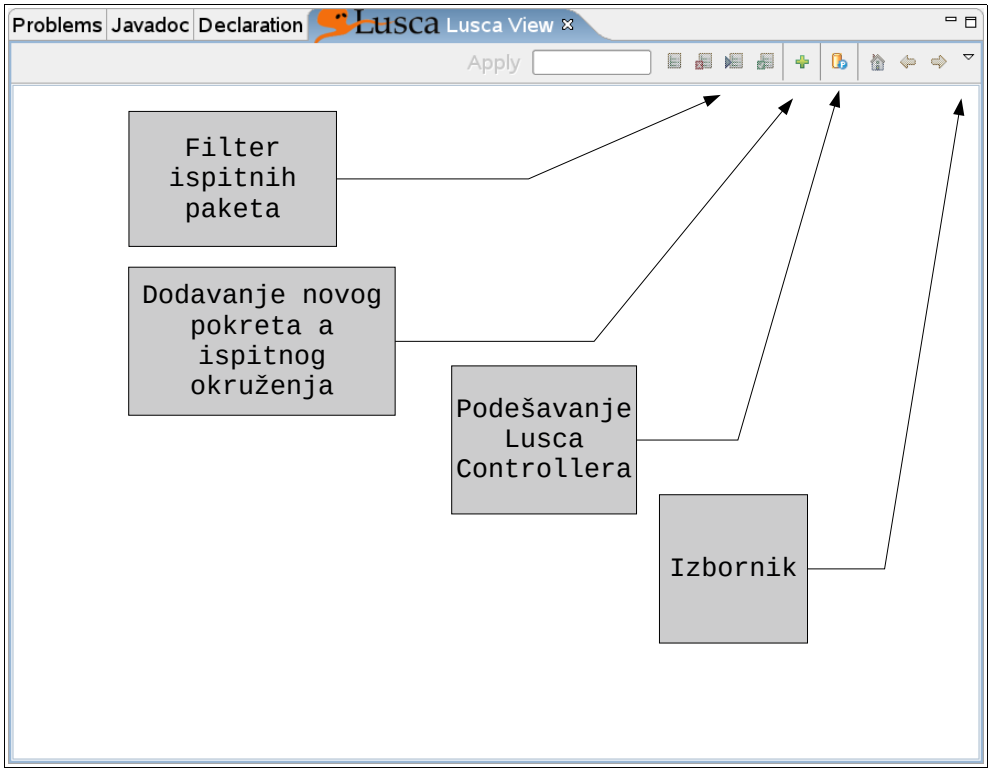
```
#Settings
#Sun Sep 02 15:59:50 GMT+01:00 2007
workspace=/home/user/workspace_lusca
timezone=Europe/Zagreb
initialized=true
username=user
```

Nakon što smo inicijalno postavili Lusca Controller, možemo otvoriti njegov pogled *Lusca View*, koji se nalazi u izborniku Eclipse IDE-a, ***Window->Show View->Other...***, gdje u mapi *LuscaController* odaberemo *Lusca View* (slika 4.4).



Slika 4.4: Odabir prikaza pogleda Lusca View

Nakon odabira, u Eclipse radnom okruženju, ukoliko je operacija izvršena ispravno, prikazat će nam se Lusca View, osnovni pogled Lusca Controllera, slika 4.5.

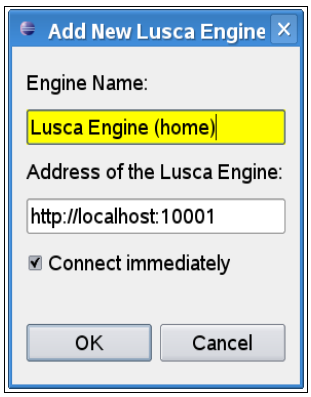


Slika 4.5: Lusca View

Opcija *Test filter* omogućava filtriranje prikaza ispitnih paketa po raznim parametrima, opcijom *Lusca Properties* možemo modificirati postavke Lusca Controllera (slika 4.2). *Izbornik* nam u bilo kojem trenutku nudi sve raspoložive opcije iz alatne trake. Kako bi započeli rad s Lusca Contollerom potrebno je definirati novi pokretač ispitnog okruženja (Lusca Engine), stoga je potrebno odabrati opciju *Add new Engine*.

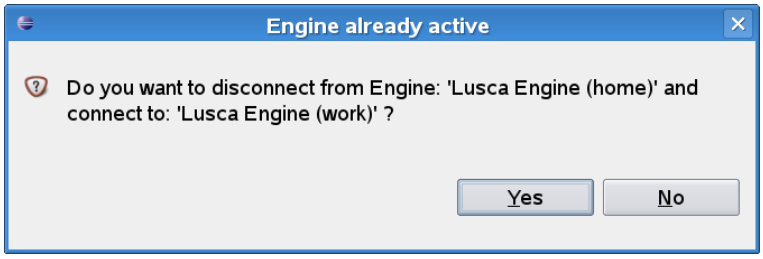
4.5.1. Dodavanje novog pokretača ispitnog okruženja

Odabirom opcije *Add new Engine*, prikazat će nam se izbornik sa slike 4.6:



Slika 4.6: Dodavanje novog pokretača ispitnog okruženja

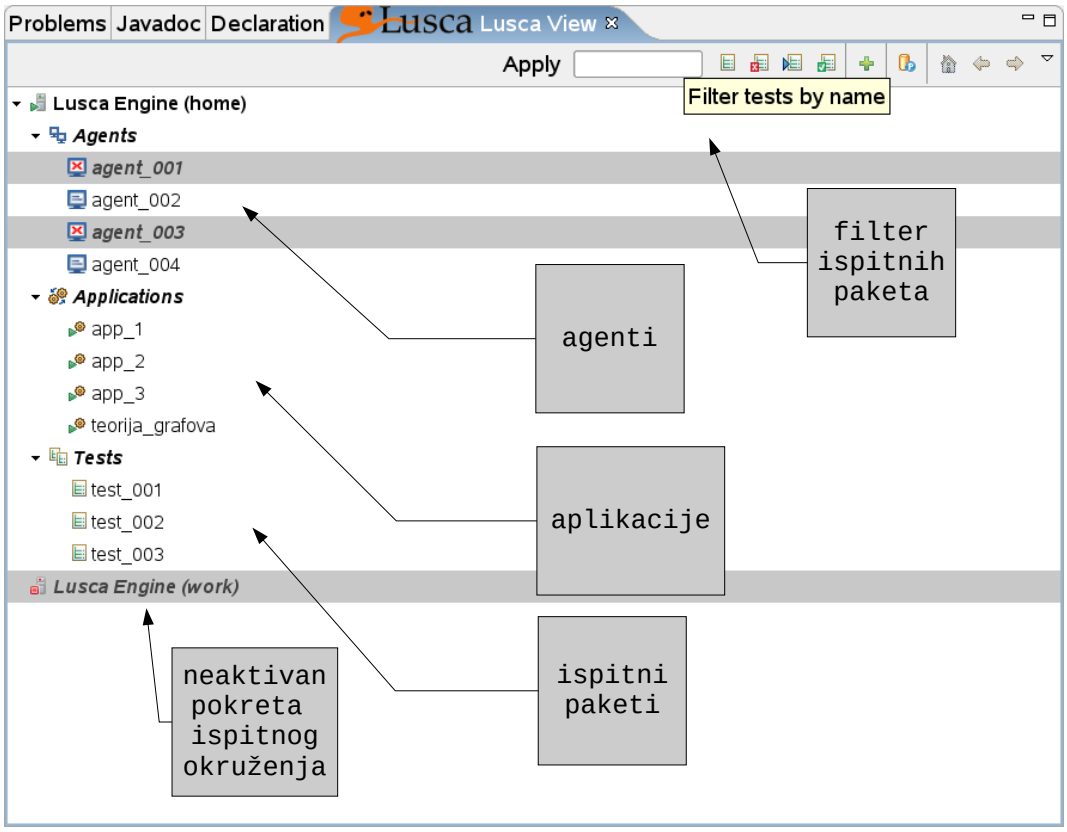
Kako bi dodali novi pokretač, potrebno je definirati njegovo simboličko ime, *Engine Name*, koji će služiti korisniku da ga razlikuje od drugih pokretača, definiranih unutar Lusca Controllera. Adresa pokretača ispitnog okruženja mora biti definirana u obliku `http://ime_ra unala:<port>` s obzirom da je sučelje Lusce, *Test Console* implementirano uporabom HTTP protokola. Kvačica *Connect immediately*, označava da se Lusca Controller odmah registrira na pokretač ispitnog okruženja, a u slučaju već postojećeg, aktivnog pokretača, korisniku nudi mogućnost odspajanja od njega (slika 4.7).



Slika 4.7: poruka "Engine already active"

4.5.2. Prikaz elemenata Lusca ispitnog okruženja

Nakon uspješno uspostavljene veze s pokretačem ispitnog okruženja, dohvaćaju se svi njegovi elementi, uključujući njihov status, prikazani stablastom strukturom. Također, filter ispitnih paketa postaje omogućen za uporabu (slika 4.8).

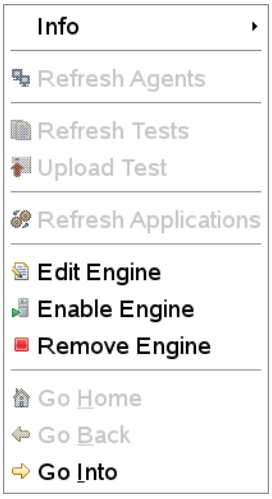


Slika 4.8: Elementi pokretača ispitnog okruženja

Na slici 4.8 prikazani su elementi Lusce s pripadajućim statusom. Elementi su podijeljeni u agente (*Agents*), aplikacije (*Applications*) i ispitne pakete (*Tests*). Pritiskom desne tipke miša na elemente u stablu dobit ćemo mogućnost izbora operacija specifičnih za element kojeg smo odabrali.

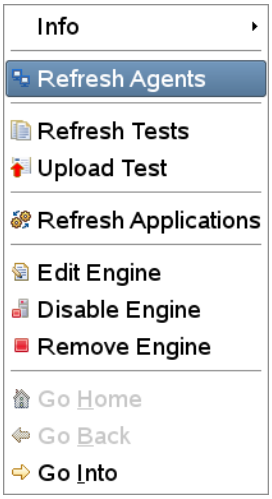
4.5.3. Lusca pokretač ispitnog okruženja

Pritiskom zadnje tipke miša na svaki od elemenata stabla prikazat će nam se izbornik s operacijama koje je moguće izvesti nad elementom. Izbornik pokretača ispitnog okruženja, ukoliko pokretač nije aktivan, nudi opcije aktivacije, pokušaja konekcije na Luscin pokretač (*Enable Engine*), uređivanje i brisanje pokretača (slika 4.9).



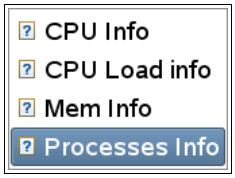
Slika 4.9: Opcije onemogućenog pokretača ispitnog okruženja

Ukoliko je pokretač ispitnog okruženja već omogućen (slika 4.10), dostupne su opcije osvježavanja agenata, ispitnih paketa, aplikacija, onemogućavanje pokretača okruženja (*Disable Engine*) i informacije (*Info*).



Slika 4.10: Opcije omogučenog pokretača okruženja

Operacije osvježavanja stanja agenata, aplikacija i ispitnih paketa izvršavaju se automatizirano od onog trenutka kad se uspostavi uspješna konekcija na Luscu. Stanje komponenti osvježava se pozadinskom dretvom *TickThread*, tako da operacije osvježivanja stanja nije nužno ručno pozivati. Operacija *Upload Test*, omogućuje odabir ispitnog paketa (*Lusca Test Package*), datoteke s nastavkom `.ltp` koja se potom prenosi s lokalnog računala Lusci HTTP (POST) protokolom. Podizbornik *Info* omogućava dohvaćanje informacija o pokretaču ispitnog okruženja, slika 4.11.

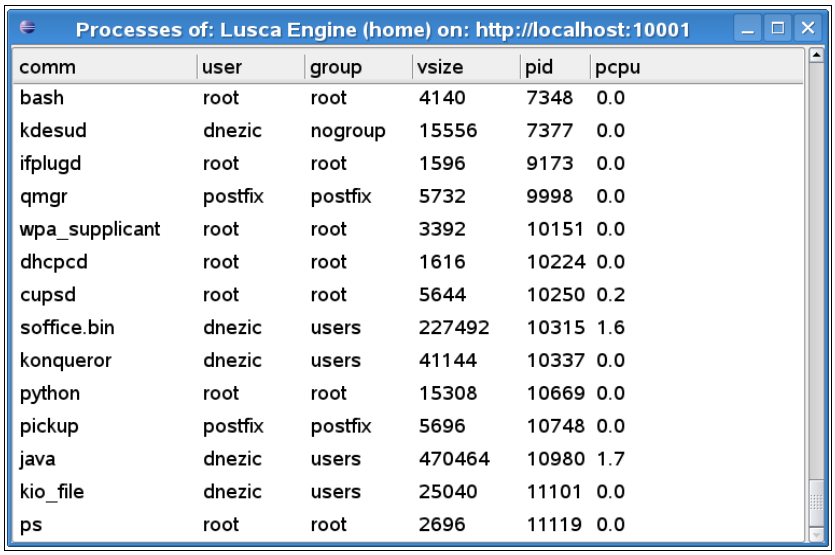


Slika 4.11: Dohvat informacija o računalu

Iste operacije dostupne su i za svaki Lusca agent pojedinačno pa će detaljnije biti opisane u nastavku. Osim navedenih operacija tu su i opcije *Home, Go Back, Go Into*, koje omogućuju šetnju po stablu komponenti pokretača okruženja, ulazak u neku granu stabla ili vraćanje natrag prema korijenu stabla.

4.5.4. Informacije o Lusca agentima i pokretaču ispitnog okruženja

Ukoliko je pokretač ispitnog okruženja aktivan, odnosno Lusca agenti dostupni, moguće je dohvatiti informacije o računalu na kojem se oni nalaze, opcijom *Info*. Istovremeno se mogu pratiti sve informacije o svim agentima i pokretaču ispitnog okruženja, ali u zasebnim prozorima. Dostupne su informacije o procesima (slika 4.12).



comm	user	group	vsize	pid	pcpu
bash	root	root	4140	7348	0.0
kdesud	dnezic	nogroup	15556	7377	0.0
ifplugd	root	root	1596	9173	0.0
qmgr	postfix	postfix	5732	9998	0.0
wpa_supplicant	root	root	3392	10151	0.0
dhcpcd	root	root	1616	10224	0.0
cupsd	root	root	5644	10250	0.2
soffice.bin	dnezic	users	227492	10315	1.6
konqueror	dnezic	users	41144	10337	0.0
python	root	root	15308	10669	0.0
pickup	postfix	postfix	5696	10748	0.0
java	dnezic	users	470464	10980	1.7
kio_file	dnezic	users	25040	11101	0.0
ps	root	root	2696	11119	0.0

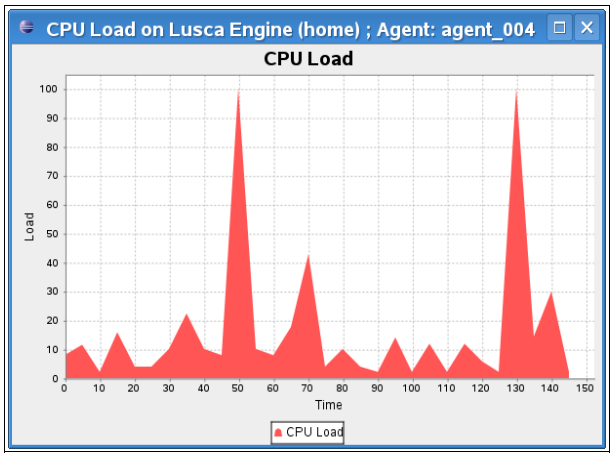
Slika 4.12: Informacije o pokrenutim procesima na pokretaču ispitnog okruženja ili agentima

Informacije o pokrenutim procesima na udaljenom računalu pribavljaju se *Test Console* protokolom. Informacije se osvježavaju svakih nekoliko sekundi te su vrlo korisne za detekciju ispravnosti rada sustava u cjelini. Na slici 4.12 vidimo tablicu s listom procesa. Kolumne tablice identificiraju procese na slijedeći način:

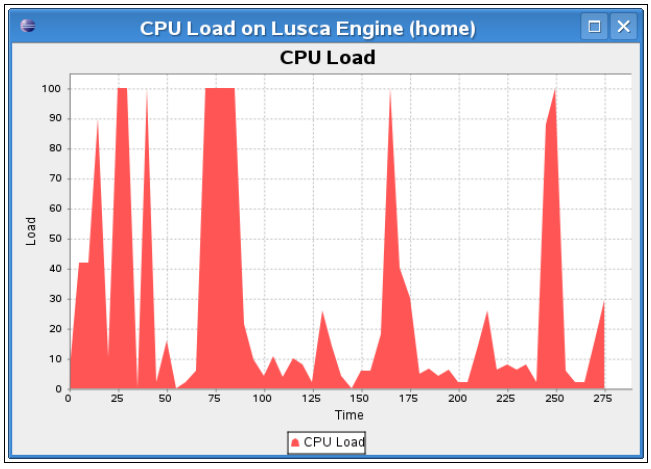
- **comm** – ime procesa
- **user** – korisnik s čijim ovlastima je proces pokrenut
- **group** – grupa s čijim ovlastima je proces pokrenut

- **vsize** – ukupna količina zauzete virtualne memorije u kilobajtima
- **pid** – process id
- **pcpu** – iskorištenje procesora

Osim navedenih informacija lako je implementirati dohvat i prikaz dodatnih informacija o pokrenutim procesima. Za svaki Lusca agent i pokretač ispitnog okruženja dostupne su informacije o opterećenju procesora koje Lusca Controller prikazuje grafom. Grafove Lusca Controller prikazuje uporabom *JFreeChart* biblioteke, koja omogućava prikaz velikog broja grafova na razne načine. Svaki graf se iscrtava na osnovu podataka o točkama, koje se prikupljaju pozadinskom dretvom, za svaki otvoreni prozor s grafom kreira se pozadinska dretva koja prikuplja podatke s Lusce. Na slikama 4.13 i 4.14 prikazan je istovremeni dohvat informacija o zauzeću procesora s pokretača `Lusca Engine (home)` i s agenta `agent_004`.

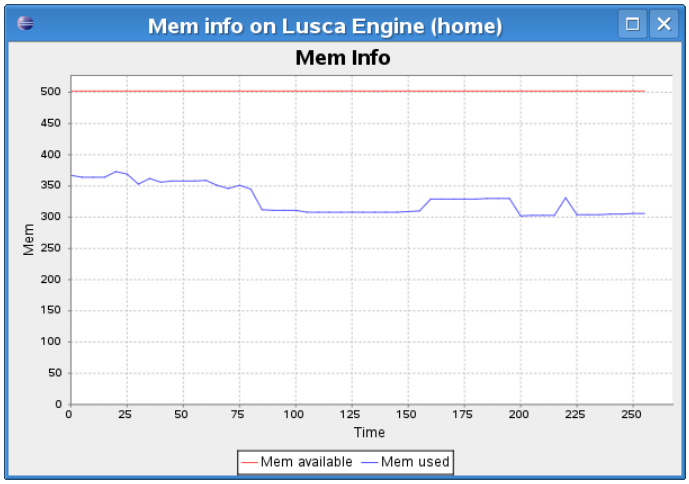


Slika 4.13: CPU Load graf, agent



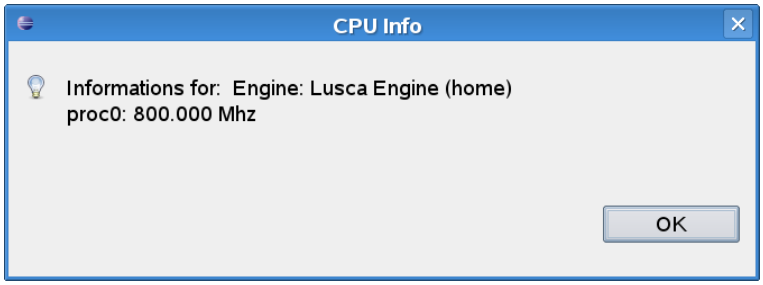
Slika 4.14: CPU Load graf, pokretač

Grafovi prikazuju opterećenje procesora u postocima u odnosu na proteklo vrijeme u sekundama. Dostupne su, na isti način kao i informacije o zauzeću procesora, i informacije o zauzeću memorije na računalima. Slika 4.15 prikazuje zauzeće memorije na pokretaču ispitnog okruženja *Lusca Engine (home)*.



Slika 4.15: Informacije o zauzeću memorije, graf

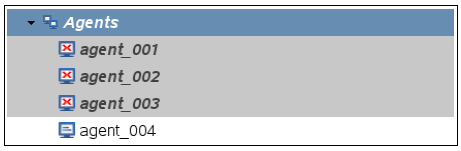
Crvena linija na dijagramu označava količinu raspoložive memorije, a plava količinu zauzete memorije. Na ordinati grafa označeni su megabajti, a apscisi proteklo vrijeme u sekundama. Podaci o zauzeću pripremljeni su i generirani na pokretaču ispitnog okruženja, pa se izračun zauzete memorije može na njemu modificirati. Opcijom *CPU Info* dohvaćaju se informacije o procesoru, brzini svih procesora koji se nalaze na računalu. Informacije o brzini procesora dohvaćaju se s pokretača ispitnog okruženja. Osim brzine procesora u MHz, može se jednostavno dodati još drugih informacija (slika 4.16).



Slika 4.16: CPU Info

4.5.5. Agenti

U stablu elemenata pokretača ispitnog okruženja, u podstablu *Agents*, nalaze se agenti (slika 4.17).

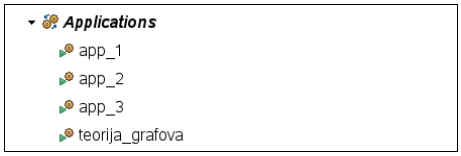


Slika 4.17: Agenti

Agenti (*Agents*) su računala, koordinirana od pokretača ispitnog okruženja, koja sudjeluju u ispitivanju. Svaki agent može imati status *READY* ili *DOWN*, ukoliko nije dostupan, a definiran je za korištenje. Lusca Controller prikazuje i obnavlja status agenata te su agenti označeni crvenim križićem nedostupni, *DOWN*.

4.5.6. Aplikacije

Aplikacije (*Applications*) definirane na pokretaču ispitnog okruženja prikazane su u istom stablu, u grani *Applications*, slika 4.18.

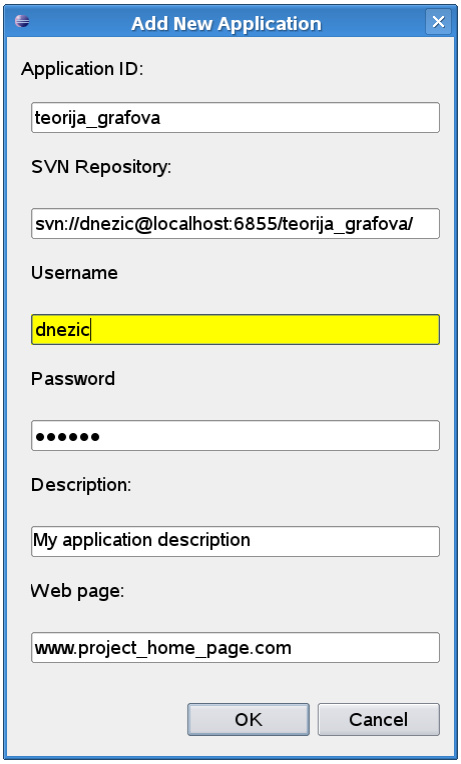


Slika 4.18: Aplikacije

Kako bi se neki ispitni paket mogao pokrenuti, sve aplikacije koje sudjeluju u njegovom izvođenju moraju biti definirane na Lusci. Ukoliko neka aplikacija nije definirana može se definirati izbornikom elementa *Applications*, opcijom *Add new Application,* slike 4.18, 4.19. Osim definiranja nove aplikacije, svaka se aplikacija može naknadno urediti ili ukloniti s pokretača ispitnog okruženja.

Slika 4.19: Izbornik elementa Applications

Odabirom opcije *Add Application*, u novom izborniku potrebno je upisati postavke koje definiraju aplikaciju, slika 4.20.



Slika 4.20: Add new Application, izbornik

Aplikacija je jednoznačno definirana parametrom *Application ID*, koji predstavlja jedinstveni ključ i ukoliko već postoji aplikacija s istim ključem neće se moći dodati. *SVN Repository* polje mora sadržavati adresu, URL, do Subversion repozitorija s aplikacijom. Na taj način, Lusca ispitno okruženje i Lusca Controller imati će informacije o načinu dohvata izvornog koda aplikacije koja se ispituje. Korisničko ime i lozinka (*username*, *password*) polja jednistvena su svakom korisniku te se te informacije spremaju lokalno na računalo. Ukoliko nema potrebe za autentifikacijom, mogu se ostaviti početne postavke s vrijednostima parametara `annonymous`. *Description* polje očekuje unos kraćeg opisa aplikacije, a *Web page* polje, web adresu na kojoj se mogu naći dodatne informacije o aplikaciji.

4.5.7. Subversion pristup aplikacijama

Unosom adrese Subversion repozitorija definiran je i način, protokol kojim se pristupa Subversion repozitoriju. S obzirom da je upravljanje Subversion repozitorijima izvedeno alatkom SVNKit, vrijede slijedeća pravila zadavanja adresa repozitorija:

```
file:///<path_to_repository>
```

Validna adresa za ovu vrstu pristupa, pri kojem je repozitorij smješten na lokalnom računalu nema smisla, osim u slučaju kada su pokretač ispitnog okruženja i Lusca Controller te Subversion repozitorij na istom računalu. Definiranjem adrese repozitorija prefiksom "svn" očekuje se pristup na Subversion *laki* poslužitelj. Kako bi autentifikacija na poslužitelj bila uspješna, potrebno ga je konfigurirati. Pristup definiranjem prefiksa adrese "svn+ssh://" označava uspostavljanje veze tunelom (SSH) do računala na kojemu se nalazi Subversion poslužitelj, nakon čega korisnik pristupa poslužitelju na isti način kao da je na njegovom vlastitom računalu. Kako bi ova vrsta konekcije mogla biti uspostavljena, potrebno je imati korisnički račun na Subversion poslužitelju, odnosno mogućnost spajanja na poslužitelj uporabom SSH konzole.

```
svn://<username>:<password>@<hostname>:<port>/<repository_name>
```

Ovim formatom adrese, potrebno je unijeti broj *porta* na kojem Subversion poslužitelj očekuje konekcije, ukoliko nije na uobičajenom broju: *3690*. Korisničko ime i lozinka se odnose na korisnički račun definiran na Subversion poslužitelju.

```
svn+ssh://<username>:<password>@<hostname>:<port>/
/<repository_name>
```

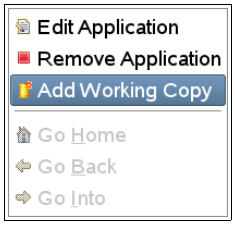
Ovim formatom adrese, potrebno je unijeti broj *porta* na kojem SSH poslužitelj očekuje konekcije, ukoliko nije na uobičajenom broju: *22*. Autentifikacijske postavke odnose se na SSH poslužitelj.

Kako bi se koristio način pristupa uporabom HTTP protokola, potrebno je spojiti se na konfigurirani Apache server s WebDAV proširenjem. Broj porta uobičajeno je *80*, te ga u tom slučaju nije potrebno navoditi.

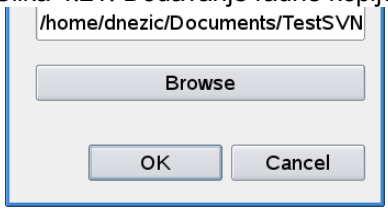
4.5.8. Radne kopije

Svaka aplikacija definirana na pokretaču ispitnog okruženja može imati definiranu jednu ili više radnih kopija. Radne kopije (poglavlje 3.4) su preslike Subversion repozitorija i koriste se za lokalni rad na projektima. Kada se izvrši neka izmjena na izvornom kodu projekta, izmjene se mogu potvrditi operacijom *commit* te spremiti na Subversion repozitorij i time postati vidljive svim korisnicima repozitorija. Kako korisnik ne bi potvrdio promjene na aplikaciji koje nisu prethodno provjerene, Lusca Controller omogućuje, definiranjem radnih kopija, mogućnost slanja *diff* datoteke pokretaču ispitnog okruženja koji će izvršiti ispitivanja uz primjenjene promjene. Ukoliko izvršavanje ispitnog paketa bude uspješno, s primjenjenom zakrpom, korisnik može potvrditi promjene koje je izvršio na radnoj kopiji. Dodavanje radne kopije u izborniku elementa *Application*, prikazano je na slici 4.21.

Svaka radna kopija definira se jedinstvenim ključem, kojim se jednoznačno razlikuje od ostalih radnih kopija neke aplikacije i radnim direktorijem na lokalnom računalu koji sadrži radnu kopiju, slika 4.22.

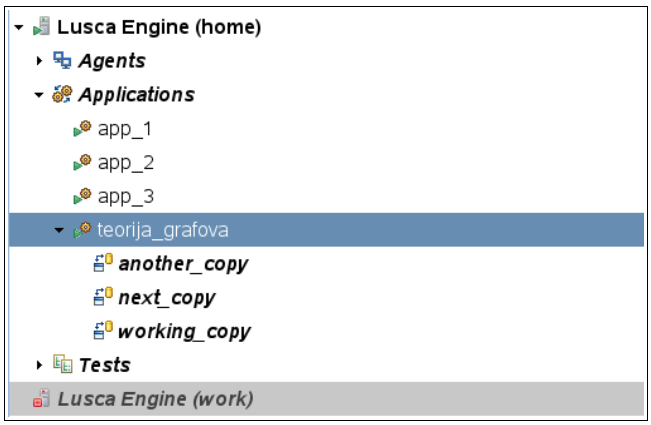


Slika 4.21: Dodavanje radne kopije



Slika 4.22: Dodavanje radne kopije (nastavak)

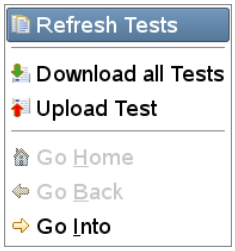
Nakon uspješnog dodavanja radne kopije, može se dodavati još radnih kopija ili uređivati postavke već postojećih. Nakon dodavanja tri radne kopije aplikaciji `teorija_grafova`, možemo vidjeti izgled stabla pokretača ispitnog okruženja, `Lusca Engine (home)`, na slici 4.23.



Slika 4.23: Izgled stabla nakon dodavanja radnih kopija

4.5.9. Ispitni paketi

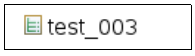
Ispitni paketi (*Tests*) se nalaze na pokretaču ispitnog okruženja te se uporabom Lusca Controllera i uređivača ispitnih paketa, udaljenim putem, mogu pokretati, zaustavljati i konfigurirati. Osnovno upravljanje ispitnim paketima vrši se izbornikom elementa stabla *Tests*, slika 4.24.



Slika 4.24: Izbornik upravljanja ispitnim paketima

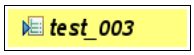
Operacija *Refresh Tests*, ozvježuje informacije prikupljajući ih na zahtjev s Lusca Controllera. Ovu operaciju nije nužno koristiti s obzirom da se informacije o ispitnim paketima učestalo osvježuju pozadinskom dretvom *Tick Thread*. Naredba *Download all Tests*, izvršit će dohvat svih definiranih ispitnih paketa s pokretača ispitnog okruženja, te će korisnik moći izabrati mapu na lokalnom računalu u koju želi smjestiti preuzete ispitne pakete. Akcija *Upload Test* prenijet će s lokalnog računala datoteku s ispitnim paketom na pokretač ispitnog okruženja. U stablu, grani *Tests*, prikazani su ispitni paketi s pokretača, definirani svojim imenom i statusom. Ispitni paket može biti u stanjima:

IDLE



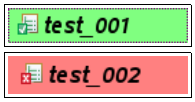
- Ispitni paket se nalazi u stanju *IDLE*, jedino u slučaju kada nikada prije nije bio pokrenut na pokretaču ispitnog okruženja. Dok je paket u stanju *IDLE* može biti pokrenut, preuzet te mogu biti dohvaćeni rezultati prethodnog pokretanja, ukoliko postoje.

RUNNING



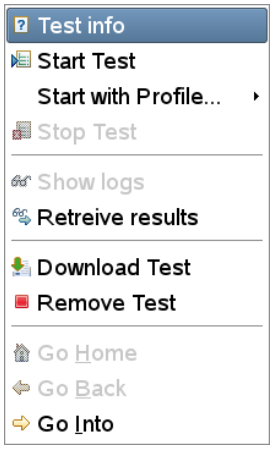
- Ispitni paket se nalazi u stanju *RUNNING* ukoliko ga je neki korisnik Lusca ispitnog sučelja pokrenuo. Tijekom ovog stanja, ispitni paket može biti zaustavljen, a može se pratiti i tijek ispitivanja, opcijom *Show Logs*.

SUCCESS / FAILURE



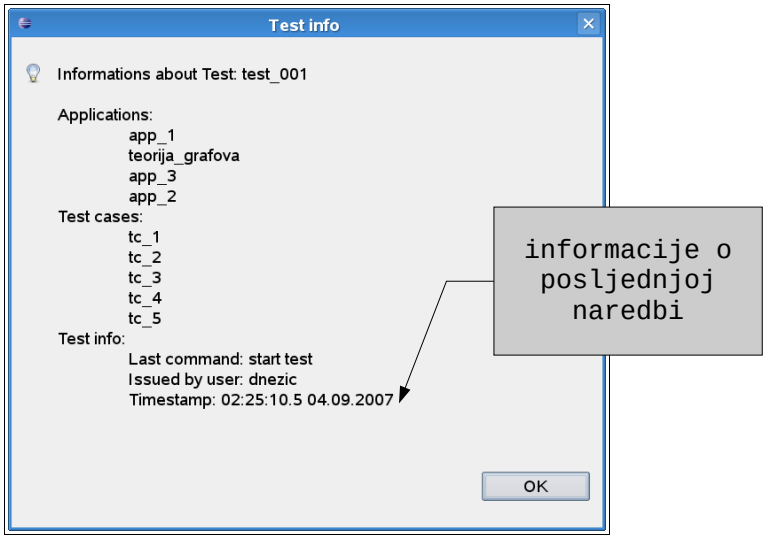
- Ispitni paket može završiti statusom *SUCCESS* ili *FAILURE,* u oba slučaja mogu se preuzeti rezultati ispitivanja u obliku komprimirane datoteke sa svim zapisima koje je ispitni paket generirao tijekom svojeg izvršavanja.

Za svaki definirani ispitni paket, element stabla u Lusca Controlleru, može se otvoriti izbornik s opcijama koje se odnose na njega samog (slika 4.25).



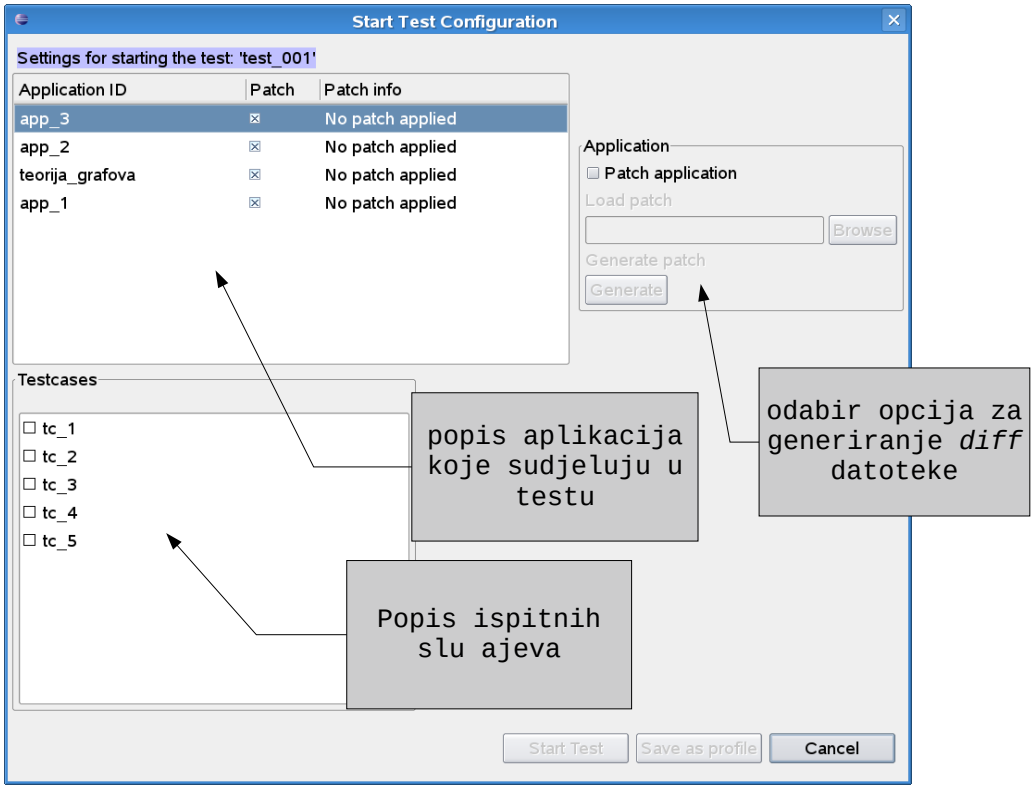
Slika 4.25: Izbornik upravljanja ispitnim paketom

Opcija *Test info* može se pokrenuti neovisno o trenutnom stanju ispitnog paketa, *Test info* prikazuje osnovne informacije o ispitnom paketu te o posljednjoj izvršenoj naredbi nad ispitnim paketom koja je promijenila njegovo stanje. Informacije o posljednjoj izvršenoj naredbi uključuju vrstu naredbe, ime korisnika koji je naredbu primijenio i vrijeme kad je to učinjeno (prikazano u lokalnom vremenu računala na kojem je pokrenut Lusca Controller). Izlaz opcije *Test info* prikazan je na slici 4.26.



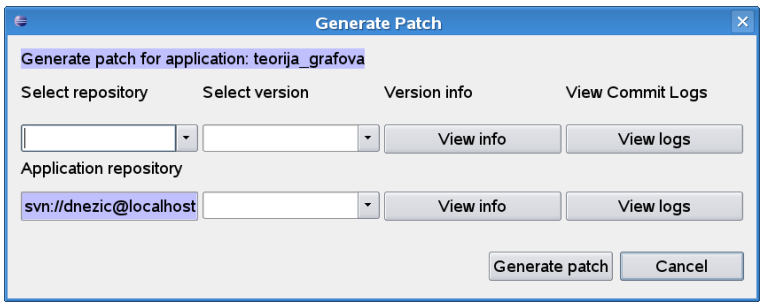
Slika 4.26: Prikaz informacija o ispitnom paketu

Ukoliko ispitni paket već nije u stanju *RUNNING*, može biti pokrenut opcijom *Start Test*. Odabirom ove opcije prikazuje nam se novi izbornik (slika 4.27) u kojem se mogu dodatno podešavati uvjeti pod kojima će se ispitni paket pokrenuti.



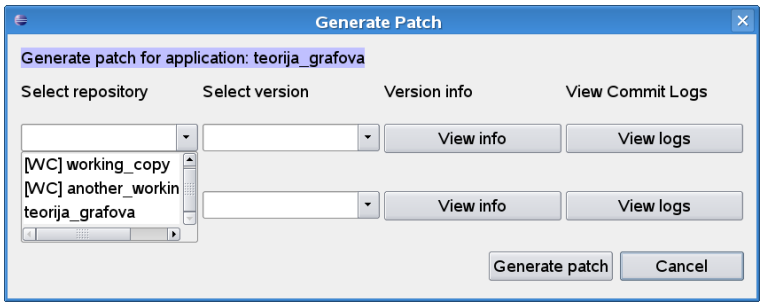
Slika 4.27: Start Test izbornik

Za svaku aplikaciju koja sudjeluje u ispitivanju, koje su definirane ispitnom deklaracijom, navedena je informacija o pridodjeljenoj zakrpi, *diff* datoteci (*Patch info*). Ukoliko želimo generirati zakrpe za aplikacije uz koje će se ispitivanje provesti, moramo odabrati stavku *Patch Application* te definirati zakrpu. Zakrpa se, u obliku *diff* datoteke može učitati, već unaprijed pripremljena, na lokalnom računalu (*Browse*) ili generirati opcijom *Generate*. Odabirom opcije *Generate* odabiremo način na koji će se zakrpa generirati, slika 4.28.



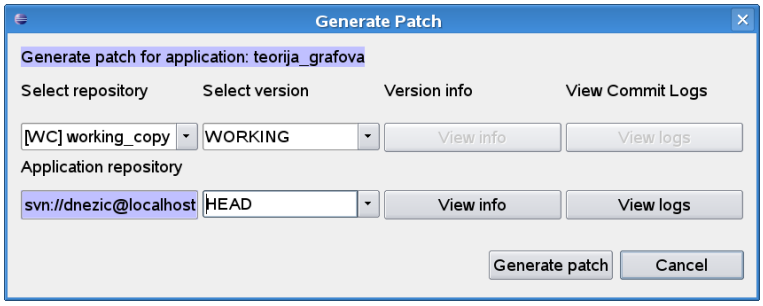
Slika 4.28: Generiranje diff datoteke (1)

U padajućem izborniku *Select repository* možemo izabrati između trenutne aplikacije koja se nalazi na Subversion repozitoriju i između jedne ili više radnih kopija ukoliko su definirane za aplikaciju koju želimo zakrpati (slika 4.29).



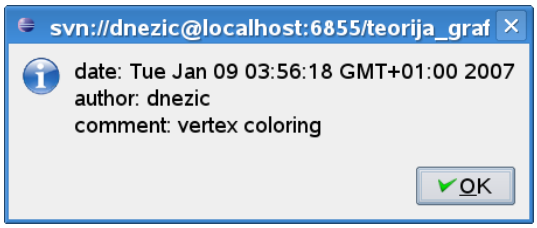
Slika 4.29: Generiranje diff datoteke (2)

Elementi padajućeg izbornika s prefiksom "[WC]" označavaju radne kopije. Ukoliko odaberemo aplikaciju sa Subversion repozitorija, a ne radnu kopiju, odabirom revizije s manjim brojem od posljednje revizije (*HEAD*), moći ćemo pokrenuti ispitni paket s točno određenom verzijom aplikacije. Odaberemo li radnu kopiju, onda se to odnosi na reviziju radne kopije *WORKING,* odnosno izmjene koje nisu još potvrđene na Subversion repozitoriju (poglavlje 3.4). Na slici 4.30 odabrana je radna kopija, kao izvorišni repozitorij, pa će se generirati zakrpa koja će opisivati razlike između revizije *WORKING* i revizije *HEAD*.



Slika 4.30: Generiranje diff datoteke (3)

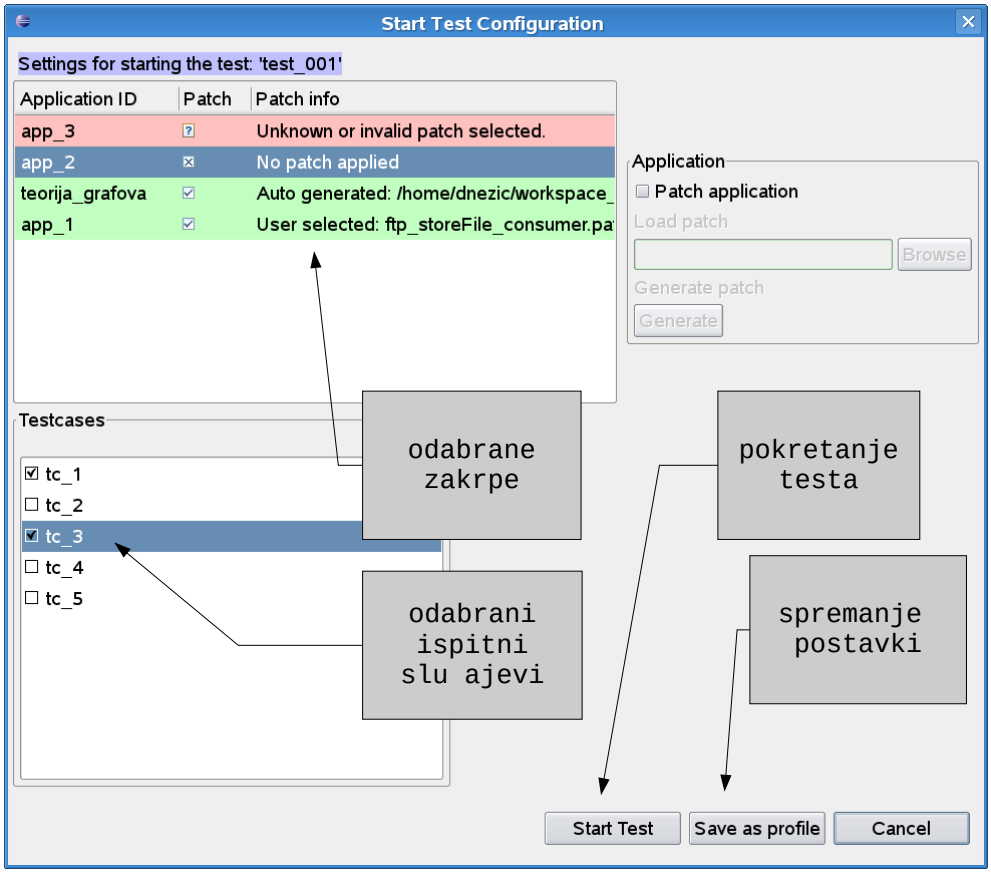
U padajućem izborniku Subversion repozitorija, mogu se vidjeti i odabrati sve potvrđene revizije aplikacije, od posljednje revizije (*HEAD*) pa sve do revizije 1. Za svaku reviziju mogu se opcijom *View info* vidjeti informacije o toj reviziji (slika 4.31).



Slika 4.31: Informacije o odabranoj reviziji

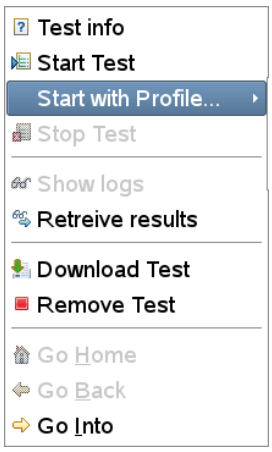
Opcijom *Generate patch* izvršit će se generiranje *diff* datoteke na temelju odabranih postavki. Nakon što smo generirali sve zakrpe možemo pristupiti odabiru ispitnih slučajeva i pokretanju ispitnog paketa. Na slici 4.32, u popisu aplikacija, možemo uočiti da je za aplikaciju `app_3` odabrana zakrpa koja nije u cjelosti definirana ili nevaljana, u tom slučaju potrebno je odabrati validnu zakrpu ili isključiti primjenu zakrpe za tu aplikaciju.

Kako bi mogli pokrenuti ispitni paket potrebno je odabrati barem jedan ispitni slučaj. Nakon što su svi uvjeti za pokretanje ispitnog paketa ispunjeni, ispitni paket se može pokrenuti akcijom *Start Test* ili se trenutne postavke mogu spremiti kao *ispitni profil* opcijom *Save as profile* (slika 4.33).



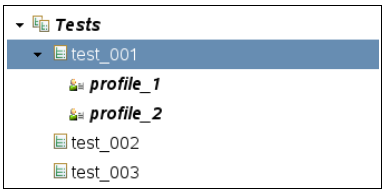
Slika 4.32: Odabir postavki pokretanja ispitnog paketa

Na taj način, pri slijedećem pokretanju ispitnog paketa, možemo odabrati profil prema kojem će se ispitivanje automatski podesiti, što će nam uštediti nešto vremena (slika 4.34).



Slika 4.34: Ispitni profili (1)

Svaki ispitni paket može imati više ispitnih profila (slika 4.35). Svaki ispitni profil može se uređivati ili ukloniti. Zatvaranjem Eclipse razvojnog okruženja, ispitni profili se spremaju u postojanu memoriju, kako postavke, ne bi ponovnim pokretanjem Lusca Controllera, bile izgubljene. Osim ispitnih profila, sačuvane su aplikacije i njihove radne kopije s pripadajućim korisničkim imenima i lozinkama.



Slika 4.35: Ispitni profili (2)

4.5.10. Prikaz zapisa o provođenju ispitivanja

Ukoliko je ispitni paket u stanju *RUNNING*, u novom *pogledu* mogu se opcijom *Show logs* prikazivati zapisi koje ispitni paket tijekom svog rada generira (slika 4.36). Ukoliko je istovremeno pokrenuto više operacija pregleda zapisa (za više ispitnih paketa), za svaku od njih kreira se posebna dretva koja se brine za njihovo dohvaćanje. Dretva proziva Luscu svake 2 sekunde te šalje upit s rednim brojem zapisa koji je posljednji dohvaćen. Lusca prosljeđuje sve zapise s većim brojem od poslanog, koji (ako) su do trenutka zahtjeva generirani. Ovakav način dohvata zapisa osigurava primitak svih zapisa bez obzira kada je zahtjev upućen. Također, pri višekorisničkom radu osigurava se dopremanje svih zapisa korisnicima.

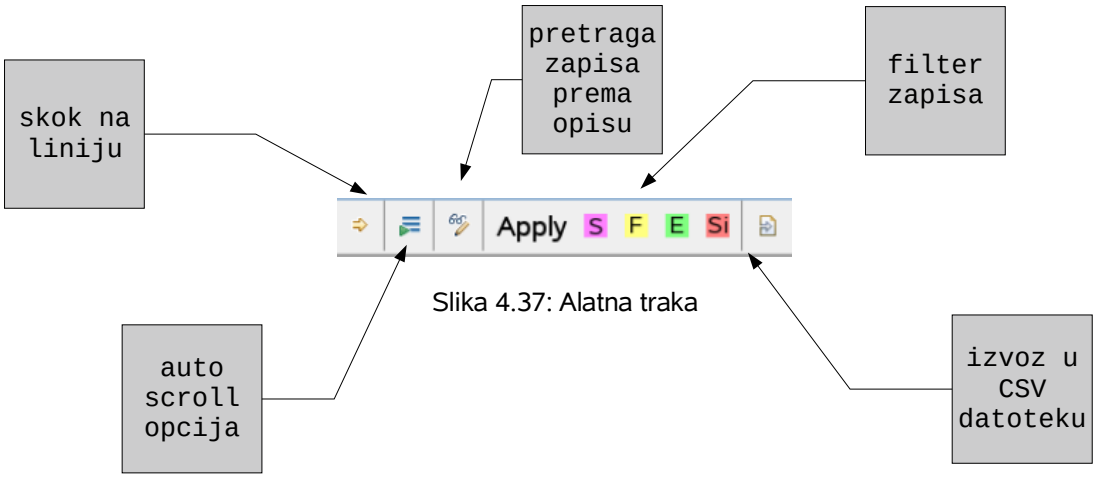
Problems | Javadoc | Declaration | Lusca View | test_001 | test_002 | test_003

Test logs — Apply S F E Si

Line	Timestamp	Info	Type
0	03:56:10.15 04.09.2007	Just some info	file
1	03:56:10.15 04.09.2007	Just some info	event
2	03:56:10.15 04.09.2007	Just some info	testcase conclusion
3	03:56:10.15 04.09.2007	Just some info	testcase execution
4	03:56:10.15 04.09.2007	Just some info	testcase start
5	03:56:10.15 04.09.2007	Just some info	testcase end
6	03:56:10.15 04.09.2007	Just some info	testcase start
7	03:56:10.15 04.09.2007	Just some info	file
8	03:56:10.15 04.09.2007	Just some info	testcase execution
9	03:56:10.15 04.09.2007	Just some info	testcase conclusion
10	03:56:10.15 04.09.2007	Just some info	testcase end
11	03:56:10.15 04.09.2007	Just some info	file
12	03:56:10.15 04.09.2007	Just some info	testcase start
13	03:56:10.15 04.09.2007	Just some info	testcase conclusion
14	03:56:10.15 04.09.2007	Just some info	testcase start
15	03:56:10.15 04.09.2007	Just some info	script status
16	03:56:10.15 04.09.2007	Just some info	testcase preparation
17	03:56:10.15 04.09.2007	Just some info	file
18	03:56:10.15 04.09.2007	Just some info	file

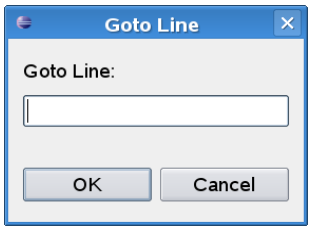
Slika 4.36: Prikaz logova

Svaki zapis prikazan je rednim brojem, datumom i vremenom generiranja (*Timestamp*) u lokalnom vremenu Lusca Controllera, opisom, te vrstom (*Type*). Pokretač ispitnog okruženja šalje zapise s vremenom naznačenim prema UTC (*Universal Time Coordinated*) zoni. Boja svake linije ujedno označava i tip zapisa radi lakšeg raspoznavanja. Alatna traka (slika 4.37) omogućava izvršavanje operacija koje mogu pomoći pri pregledavanju generiranih zapisa.



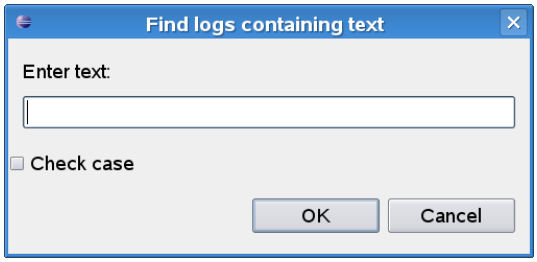
Slika 4.37: Alatna traka

Prva opcija je *Goto Line* (slika 4.38), kojom se, u malom izborniku, može direktno pozicionirati na liniju s upisanim rednim brojem. Redni broj linije mora biti u rasponu dotad primljenih zapisa.



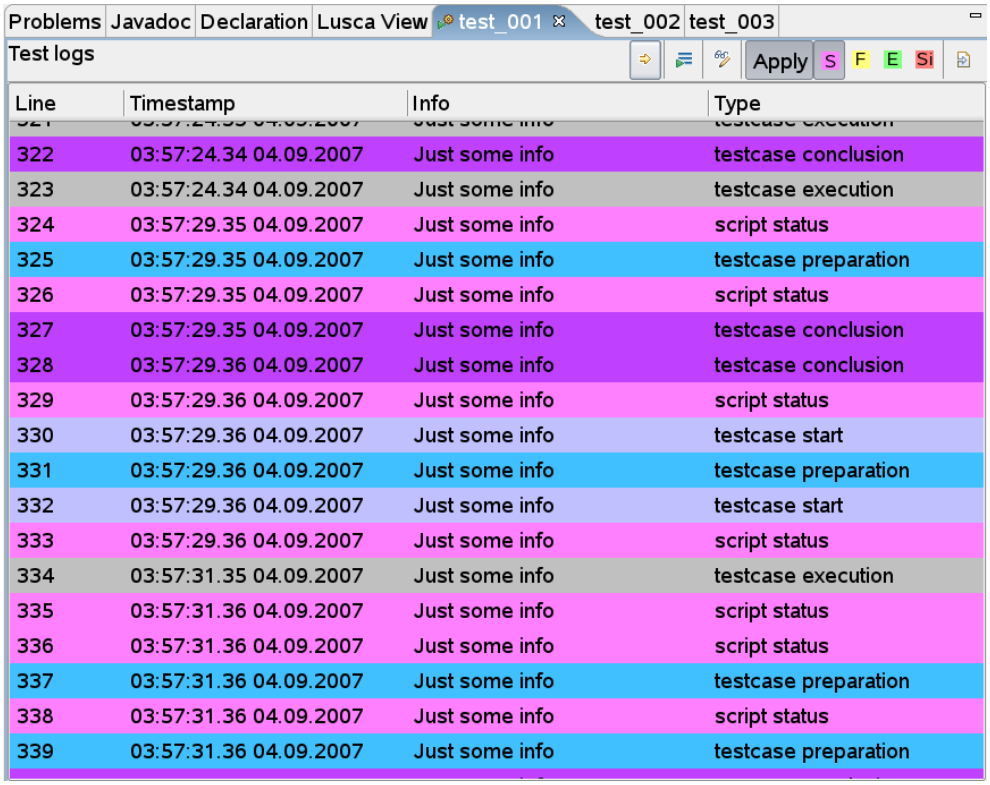
Slika 4.38: Skok na liniju određenu brojem

Slijedeća opcija je *Auto Scroll,* koja će, ukoliko je izabrana, automatski pozicionirati pokazivač na posljednju pristiglu liniju. Zapisi se mogu pretraživati prema svojem opisu (slika 4.39). Korisnik može upisati riječ, a tražilica će se sekvencijalno pozicionirati na sve linije koje zadovoljavaju dani kriterij, dok ne pretraži sve pristigle zapise. Da bi neki zapis bio uspješno pronađen, upisani tekst mora biti podskup opisa zapisa.



Slika 4.39: Pretraga za linijama koje sadrže u info dijelu upisani niz

Pristigli zapisi mogu biti filtrirani, radi lakšeg snalaženja, po svojem tipu. Korisnik može odabrati iz alatne trake tipove zapisa koje želi zadržati u *pogledu*, to su redom: skripte (S), datoteke (F), događaji (E) i signali (Si). Nakon što su odabrani tipovi logova koji se žele zadržati u *pogledu*, potrebno je aktivirati filter pritiskom na dugme *Apply* (slika 4.40).



Slika 4.40: Uporaba filtera zapisa

Svi pristigli logovi mogu se izvesti u CSV, *Comma Separated Values* datoteku, pa korisnik može naknadno detaljnije proučiti pristigle zapise.

5. Zaključak

Lusca je okruženje za ispitivanje u ranoj fazi razvoja. Kako bi se korištenje Lusca ispitnog okruženja učinilo jednostavnijim i približilo novim korisnicima bilo je potrebno implementirati korisničko sučelje. Implementirani su korisničko sučelje i uređivač ispitnih paketa u vidu Eclipse dodataka, kako bi se iskoristila popularnost i brojne mogućnosti Eclipse platforme. Lusca ispitno okruženje svojim *lakim* dizajnom i implementacijom u jeziku *python*, pruža mnogo mogućnosti za daljnje proširivanje i nadogradnju. Iz tih razloga bilo je potrebno osmisliti protokol za komunikaciju koji je također jednostavno proširivati i nadograđivati, sukladno daljnjem rastu Lusca ispitnog okruženja. Za osnovni protokol komunikacije između Eclipse dodatka i Lusce odabran je HTTP, kojim se vrši prijenos podataka u JSON formatu. Prednost ovakvog načina komunikacije je velika prilagodljivost i jednostavnost implementacije te kompatibilnost s velikim brojem platformi i tehnologija za koje se mogu izrađivati nova sučelja. Upravo takav način komunikacije Lusca Controllera s Luscom je ujedno i njegova najveća mana. Lusca Controller mora učestalo prozivati HTTP zahtjevima Luscu, iako se stanje njenih objekata nije mijenjalo. Prednost ovakvog pristupa je u tome što Lusca ne mora voditi brigu o obavještavanju svih registiranih upravljačkih sučelja o nastalim promjenama, a mana je beskorisno opterećivanje Lusce zahtjevima. Dodatni problem može nastati u slučaju većeg broja korisnika spojenih na *Test Console* protokol. Sa strane krajnjeg korisnika to ne predstavlja veliki problem, s obzirom da mu nije važna velika promptnost pri dohvatu podataka. Dodatno, pri operaciji praćenja zapisa generiranih izvršavanjem testa, potrebno je učestalo dohvaćati veću količinu informacija, što može dodatno zagušiti Eclipse dodatak i Luscu. Stoga je potrebno razmisliti, u nekim daljnjim primjenama, o implementaciji binarnog protokola. Treba uzeti u obzir i činjenicu da Eclipse razvojno okruženje, po svojoj svrsi, ipak nije namijenjeno ovakvim operacijama te je dosta tromo.

6. 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. Sun Microsystems: *Tuning Garbage Collection with the 1.4.2 Java Virtual Machine,* URL: http://java.sun.com/docs/hotspot/gc1.4.2 (01/09/07).
3. Java (programming language), URL: http://en.wikipedia.org/wiki/Java_(programming_language) (01/09/07)
4. James Gosling: *James Gosling's Letter to the Java Community,* URL: http://www.sun.com/software/opensource/java/gosling_letter.jsp (31/08/07)
5. Sun Microsystems: *About Java Technology,* URL: http://www.sun.com/java/about (31/08/07)
6. Ceki Gülcü: *Short introduction to log4j,* URL: http://logging.apache.org/log4j/1.2/manual.html (29/08/07)
7. OSGi Alliance: *About OSGi,* http://www.osgi.org/about/faqs.asp?section=1 (01/09/07)
8. Jim des Rivieres, Wayne Beaton: *Eclipse Platform Techical Oveview,* URL: http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html (30/08/07)
9. David Gallardo: *Developing Eclipse plug-ins,* URL: http://www.ibm.com/developerworks/java/library/os-ecplug (30/08/07)
10. Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato: *Version Control with Subversion,* O'Reilly, 2004.
11. Diff, URL: http://en.wikipedia.org/wiki/Diff (30/08/07)
12. Introducing JSON, URL: http://www.json.org (30/08/07)

Dodatak A – XML shema 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>
```