



УНИВЕРЗИТЕТ У НОВОМ САДУ ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: Себастијан Новак

Број индекса: 13012

Тема рада: Реализација подсистема за аутоматско тестирање веб базираних апликација у оквиру Intent+ тест платформе

Ментор рада: др Мирољуб Поповић

Нови Сад, месец, 2014



КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:			
Идентификациони број, ИБР:			
Тип документације, ТД:	Монографска документација		
Тип записа, ТЗ:	Текстуални штампани материјал		
Врста рада, ВР:	Завршни (Bachelor) рад		
Аутор, АУ:	Себастијан Новак		
Ментор, МН:	др Мирослав Поповић		
Наслов рада, НР:	Реализација подсистема за аутоматско тестирање веб базираних апликација у оквиру Intent+ тест платформе		
Језик публикације, ЈП:	Српски / латиница		
Језик извода, ЈИ:	Српски		
Земља публиковања, ЗП:	Република Србија		
Уже географско подручје, УГП:	Војводина		
Година, ГО:	2014		
Издавач, ИЗ:	Ауторски репринт		
Место и адреса, МА:	Нови Сад; трг Доситеја Обрадовића 6		
Физички опис рада, ФО: (поглавља/страна/ цитата/табела/слика/графика/прилога)	7/41/14/9/21/0/0		
Научна област, НО:	Електротехника и рачунарство		
Научна дисциплина, НД:	Рачунарска техника		
Предметна одредница/Кључне речи, ПО:	тестирање, веб апликације, тестирање веб базираних апликација		
УДК			
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад		
Важна напомена, ВН:			
Извод, ИЗ:	<p>У раду је описана реализација подсистема за аутоматизовано тестирање веб базираних апликација.</p> <p>Подсистем је пројектован за уграђивање у Интент+ окружење за тестирање.</p> <p>Подсистем користи јава аплет да би дошао до контроле корисниковог система. Јава аплет такође користи Пајтон базирано окружење - Селениум, за контролу корисниковах интернег прегледача.</p> <p>Јава аплет има могућност извршавања Пајтон скрипти, зато што има уграшен Пајтон интерпретер реализован у Јави - Џајтон.</p>		
Датум прихватања теме, ДП:			
Датум одбране, ДО:			
Чланови комисије, КО:	Председник:	професор др Миодраг Темеринач	
	Члан:	др Јелена Ковачевић	Потпис ментора
	Члан, ментор:	професор др Мирослав Поповић	



KEY WORDS DOCUMENTATION

Accession number, ANO:			
Identification number, INO:			
Document type, DT:	Monographic publication		
Type of record, TR:	Textual printed material		
Contents code, CC:	Bachelor Thesis		
Author, AU:	Sebastian Novak		
Mentor, MN:	professor PhD Miroslav Popović		
Title, TI:	Implementation of a subsystem for automated testing of web based applications within the Intent+ Integrated Test Environment.		
Language of text, LT:	Serbian		
Language of abstract, LA:	Serbian		
Country of publication, CP:	Republic of Serbia		
Locality of publication, LP:	Vojvodina		
Publication year, PY:	2014		
Publisher, PB:	Author's reprint		
Publication place, PP:	Novi Sad, Dositeja Obradovica sq. 6		
Physical description, PD: (chapters/pages/ref./tables/pictures/graphs/appendices)	7/41/14/9/21/0/0		
Scientific field, SF:	Electrical Engineering		
Scientific discipline, SD:	Computer Engineering, Engineering of Computer Based Systems		
Subject/Key words, S/KW:	Testing, web applications, web based application testing		
UC			
Holding data, HD:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia		
Note, N:			
Abstract, AB:	<p>This paper describes an implementation of a subsystem for automated testing of web based applications. The subsystem is designed to be easily integrated into the Intent+ Integrated Test Environment.</p> <p>The subsystem utilizes a Java applet to gain access and control over the user's system. The Java applet also used Python based framework for controlling the user's browsers – Selenium. The applet is able to execute Python scripts as it also has an embedded JVM based Python interpreter – Jython.</p>		
Accepted by the Scientific Board on, ASB:			
Defended on, DE:			
Defended Board, DB:	President:	professor PhD Miodrag Temerinac	
	Member:	PhD Jelena Kovačević	Menthor's sign
	Member, Mentor:	professor PhD Miroslav Popović	

Zahvalnost

Zahvaljujem se mentoru dr Miroslavu Popoviću i asistentu Dejanu Stefanoviću na svoj pruženoj pomoći i savetima u toku izrade ovog rada.

Takođe se zahvaljujem profesorici, dr Jeleni Kovačević, bez čijeg odobrenja za rad u okviru Intent+ tima, izrada ovog rada bi bila značajno otežana.

SADRŽAJ

1.	Uvod.....	1
2.	Teorijske osnove	2
2.1	Selenium i Selenium WebDriver.....	2
2.2	Selenium IDE	3
2.3	RT-Intent.....	4
2.4	Jython.....	5
2.5	Google Web Toolkit – GWT	6
2.6	MVP projektantski šablon	6
2.7	Java aleti.....	7
2.8	JSNI - Sprega Jave i JavaScript-a.....	8
2.9	Apache Commons IO	8
2.10	Metaprogramiranje	8
2.10.1	Refleksija	9
2.11	Poliglotno programiranje	10
2.12	Global Interpreter Lock i konkurentno programiranje u Jython-u.....	11
3.	Koncept rešenja	13
3.1	Uvod	13
3.2	Prototip 1 – Java aplet sa ugrađenim Jython interpreterom i Seleniumom	15
3.2.1	Java -Python sprega.....	16
3.2.2	UnitTest API za ispitivanje i rezultati ispitivanja	17
3.3	Prototip 2 – Java program za komunikaciju sa Intent poslužiocem	18
3.3.1	Komunikacija sa Intent+ poslužiocem	18
3.3.2	IntentInterface – apstrakcija komunikacije sa Intent+ poslužiocem	18

4.	Programsko rešenje	21
4.1	GWT Korisnička sprega	22
4.1.1	View	22
4.1.1.1	ExecutionTab – kartica za izvršavanje scenarija za ispitivanje	24
4.1.1.1.1	Učitavanje sadržaja korisnikove datoteke sa izvornim kodom	24
4.1.1.1.2	AceEditor – JavaScript baziran editor za isticanje sintakse programskog koda.....	24
4.1.1.2	TestSelectionTab – kartica za prijavu na Intent+ i izbor plana ispitivanja .	26
4.1.2	Presenter	27
4.1.3	Izvoz GWT metoda u JavaScript	28
4.1.4	JavaScriptToJava sprega	29
4.2	MiniExecutorAplet	29
4.2.1	Instanciranje Jython interpretera i PythonWrapper-a.....	31
4.2.2	JythonFactory.....	31
4.2.3	Podešavanje sistema	31
4.3	PythonWrapper	32
4.3.1	Ručno izvršavanje scenarija za ispitivanje	32
4.3.2	Propagacija izuzetaka	33
4.3.3	Sigurnosna ograničenja	33
4.4	TestResults	34
5.	Ispitivanje	35
6.	Zaključak	41
7.	Literatura i reference	43

SPISAK SLIKA

Slika 2.1 WebDriver API	2
Slika 2.2 : Selenium IDE snima radnju korisnika	3
Slika 2.4 : RT-Intent	4
Slika 2.3: Mogućnost generisanja koda na osnovu zabeleženih radnji.....	4
Slika 2.5: Bazična MVC aplikacija bez asinhronne komunikacije Controller-a i View-a	6
Slika 2.6: MVC bazirana aplikacija sa asinhronom komunikacijom <i>View</i> i <i>Controller</i> modula.....	7
Slika 2.7: MVP projektantski šablon	7
Slika 3.1: Koncept rada podsistema.....	14
Slika 3.2: Grafička korisnička sprega za lakše ispitivanje protipa 1	15
Slika 3.3 : Izveštaj sa rezultatima ispitivanja	16
Slika 3.4 : Java-Python sprega.....	17
Slika 3.5: Prototip 2 – Konzolna aplikacija - sprega sa Intent+ sistemom	20
Slika 4.1: Nivoi komunikacije podistema.....	21
Slika 4.2: Izgled kartice za izvršavanje ispitivanja nakon neuspelog izvršenja scenarija za ispitivanje	25
Slika 4.3: Izgled kartice za izvršavanje ispitivanja nakon uspešnog izvršenja scenarija za ispitivanje	25
Slika 4.4 : Početni izgled kartice, prijava na Intent+	26
Slika 4.5: Nakon uspešne prijave, prikaz planova ispitivanja	26
Slika 4.6: Izvršavanja svih scenarija za ispitivanje u okviru izabranog plana za ispitivanje	27
Slika 4.7: Komunikacija GWT-a i apleta	30

Slika 5.1: JConsole – pregled stanja programskih niti unutar virtuelne mašine	39
Slika 5.2: VisualVM – alat za vizualizaciju trenutnog stanja virtuelne mašine.....	39

SPISAK TABELA

Tabela 2.1: JVM jezici	11
Tabela 3.1: Pregled realizovanih prototipova u toku istraživanja	13
Tabela 3.2 : Spisak usluga Intent+-a koje IntentInterface koristi.....	18
Tabela 3.3: API za komunikaciju sa Intent+ poslužiocem.....	19
Tabela 4.1: View sprega.....	23
Tabela 4.2: Presenter sprega.....	28
Tabela 4.3: API za kontrolu PythonWrapper-a	31
Tabela 5.1: Scenariji ispitivanja	37
Tabela 5.2: Rezultati ispitivanja	38

SPISAK LISTINGA PROGRAMSKIH KODOVA

Listing 2.1: Primer primene metaprogramiranja u ANSI-C.....	9
Listing 2.2: Primer primene refleksije u Javi	10
Listing 4.1: Izvoz GWT metoda.....	28
Listing 4.2: Propagacija izuzetka bačenog iz Python-a	33

SKRAĆENICE

GWT	- Google Web Toolkit, Programabilne sekvencijalne mreže
JVM	- Java Virtual machine, Javina virtuelna mašina
MVP	- Model View Presenter, projektantski šablon
JSNI	- JavaScript Native Interface, GWT sprega JavaScript-a i Java
DTO	- DataTransferObject, objekat sa informacijama iz baze podataka, model entiteta iz odgovarajuće tabele
WS	- Web Service, web servis
GIL	- Global Interpreter Lock, ugrađeni globalna isključiva brava(eng. Mutex) u određenim interpreterima, za automatsku zaštitu pri konkurentnom izvršavanju koda.

1. Uvod

Sve veće potrebe za razvojem programske podrške, uslovile su ubrzavanje iste, samim tim postavile nove standardne na nivou zahteva koje programska podrška koja se razvija mora da ispunjava. Jedan od čestih zahteva jeste nezavisnost složene programske podrške od platforme na kojoj se izvršava. Univerzalnu prenosivost se ostvaruje uz pomoć korisničke sprega koja se izvršava unutar internet pregledača, odnosno izradom rešenja u obliku mrežno bazirane (eng. web) aplikacije.

Zadatak ovog rada je realizacija podsistema (koji će biti ugrađen u Intent okruženje) za automatsko pokretanje i izvršavanje automatskog ispitivanja unutar internet pregledača upotrebom Selenium okruženja.

Testovi se preko web servisa automatski dobavljuju od Intent poslužioca, te izvršavaju u zadatim internet pregledačima (Firefox, Chrome). Rezultati testova se posredstvom veb servisa vraćaju nazad Intent poslužiocu.

Cilj podsistema za ispitivanje jeste da se izvršava unutar internet pregledača kako bi se što lakše mogao koristiti u različitim okruženjima.

Ispitivanje ispravnosti rada programskog rešenja je obavljeno ispitivanjem rada Intent okruženja.

2. Teorijske osnove

Ovo poglavlje opisuje teorijske osnove na kojima je rad zasnovan kao i tehnologiju upotrebljenu za realizaciju.

2.1 Selenium i Selenium WebDriver

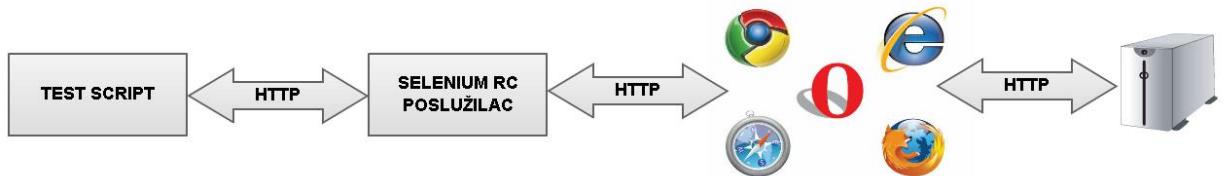
Selenium predstavlja okruženje za ispitivanje mrežnih (eng. Web) aplikacija. Nudi skup mrežnih rukovalaca (eng. WebDrivers) za upravljanje većinom popularnih internet pregledača.

Selenium WebDriver omogućava programsku kontrolu internet pregledača uz pomoć univerzalnog klijentskog API-a , koji se oslanja na specifično napisane upravljačke programe za kontrolu konkretnih internet pregledača – slika 2.1.



Slika 2.1 WebDriver API

Ranije implementacije su zahtevale direktnu komunikaciju sa Selenium RC poslužiocem, koji je prihvatao zahteve za kontrolu internet pregledača i imao ulogu posrednika između aplikacije koja izvršava ispitivanje i samog internet pregledača – slika 2.2.

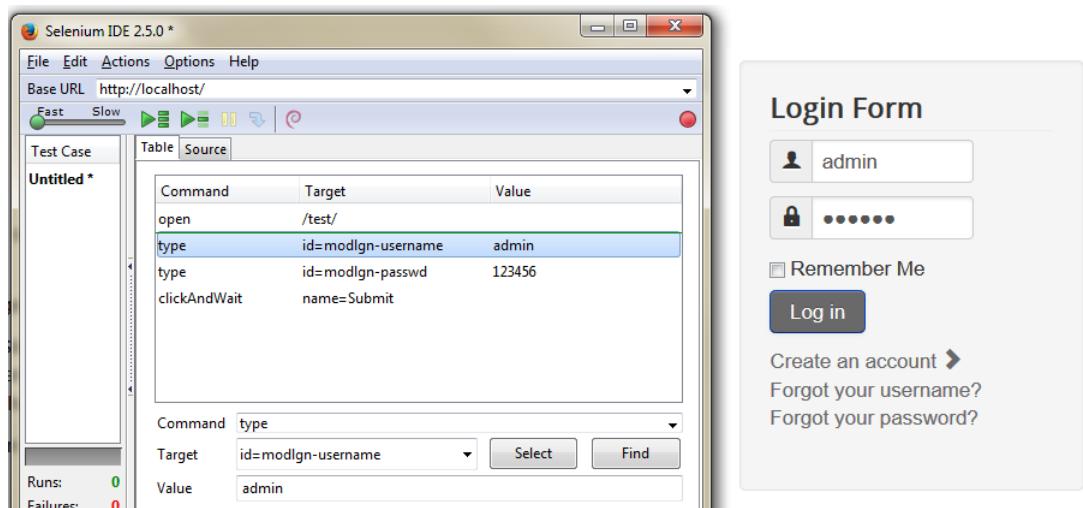


Slika 2.2: Rad sa *Selenium RC*

U sadašnjoj verziji, mrežni rukovalac u sebi sadrži implementacije upravljačkih programa za kontrolu Firefox, Internet Explorer, Safari i Opera internet pregledača. Svi se koriste na potpuno isti način, preko univerzalnog API-a, jer je upravljanje sada u potpunosti apstrahovano. Za Chrome internet pregledač postoji implementacija upravljačkog programa od strane treće strane, kao zvaničan deo Chromium projekta [1].

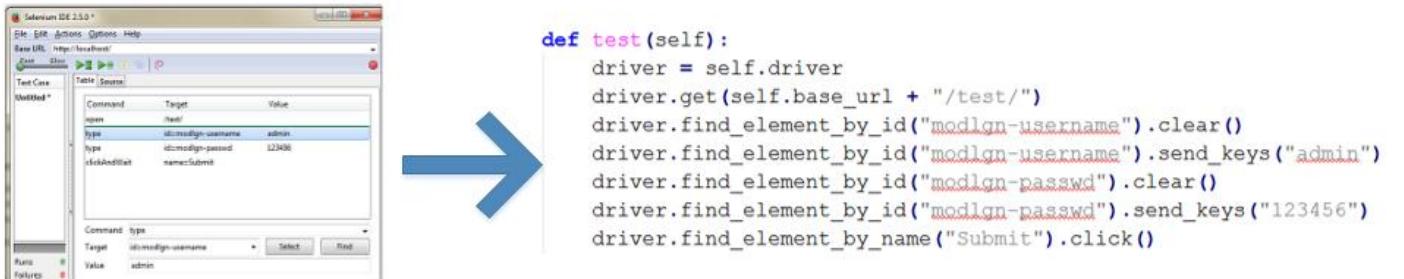
2.2 Selenium IDE

Selenium IDE (slika 2.3) je integrisano razvojno okruženje koje omogućava generisanje skripti za ispitivanje snimanjem akcija korisnika kao i kontrolisano izvršavanje skripti. Implementiran je kao dodatak za Firefox internet pregledač.



Slika 2.2 : Selenium IDE snima radnju korisnika

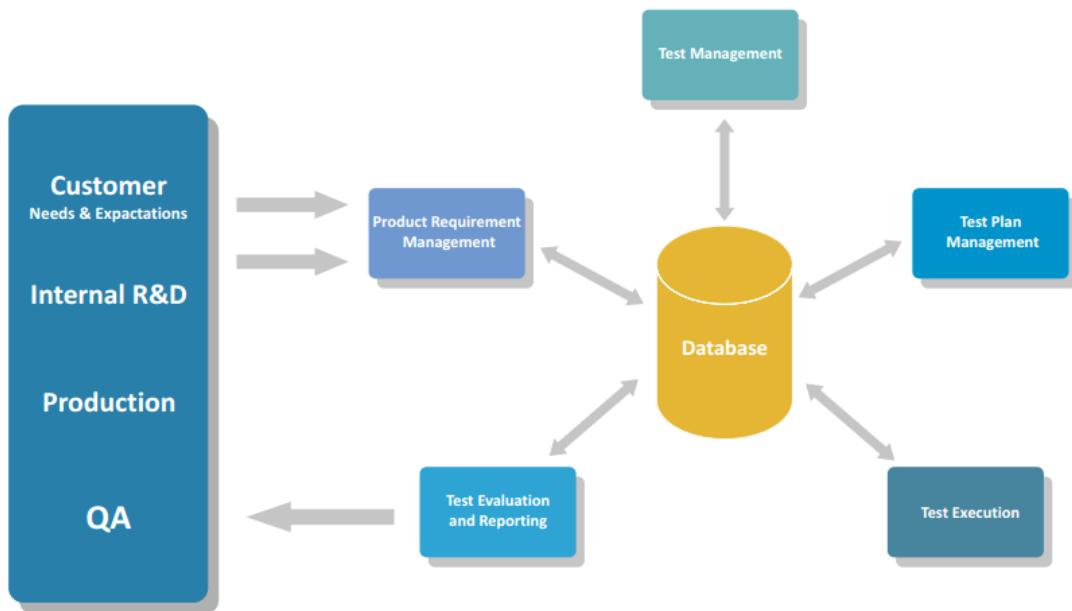
Na osnovu snimljenih radnji korisnika je moguće generisani izvorni kod za kontrolu internet pregledača – slika 2.4.



Slika 2.3: Mogućnost generisanja koda na osnovu zabeleženih radnji

2.3 RT-Intent

RT-Intent je složeno i sveobuhvatno integrisano okruženje za ispitivanje koje omogućava rukovođenje i izvršavanje ispitivanja u toku razvoja i održavanja programske podrške, kao i za kontrolu kvaliteta tokom njenog životnog veka.



Slika 2.4 : RT-Intent

RT-Intent se sastoji od nekoliko modula:

- Intent-RM – zadužen za stvaranje zahteva za stvaranje, modifikaciju i održavanje programskih proizvoda
- Intent-TM – pokriva rukovođenje scenarija ispitivanja, omogućava korisnicima da definišu scenarije za ispitivanje kao i neophodne korake za izvršavanje istih i skladišti ih u centralnu bazu podataka.
- Intent-TPM – Nudi mogućnost podešavanja sistema za ispitivanje kao i rukovođenje planom ispitivanja.
- Intent-TE – Modul zadužen za procenjivanje rezultata ispitivanja, grešaka i održavanja veze rezultata ispitivanja sa zahtevima.
- RT-Executor – Alat za izvršavanje ručni scenarija za ispitivanje kao i automatskih. RT-Executor kontroliše opremu neophodnu za izvršavanje datog scenarija za ispitivanje, samo izvršavanje testa i rukuje sa rezultatima ispitivanja.
- RT-DB – Centralna baza podataka za skladištenje podataka celog RT-Intent okruženja.

Cilj ovog rada jeste da, po ugledu na RT-Executor, ponudi usluge ispitivanja mrežno baziranih programskih rešenja, izvršavanjem scenarija za ispitivanje kontrolom internet pregledača, kao i generisanja izveštaja o rezultatu ispitivanja.

2.4 Jython

Jython je implementacija Python programskog jezika za Java platformu.

Omogućava obostranu interakciju Python modula i klasa sa Java paketima i klasama, kao i Java klasa sa Python modulima i klasama. Jython kod se uvek prevodi u Java bajtkod.

U zvaničnoj Jython dokumentaciji [\[2\]](#), posebno su istaknute sledeće mogućnosti:

1. Dinamičko prevođenje Jython koda u Java bajt kod
2. Mogućnost da Python klasa nasledi Java klasu, implementira Java interfejs
3. Opcionalna statička kompilacija - za pravljenje apleta, servleta, binova
4. Podršku za Python jezik i sve standardne Python biblioteke

2.5 Google Web Toolkit – GWT

Google Web Toolkit – skraćeno GWT, omogućava pisanje složenih korisničkih sprega za veb baziranih programskih rešenja u Javi.

GWT programski kod napisan u Javi, prevodi u Javascript.

Ono što GWT nudi, jeste Model-View-Presenter (MVP) projektantski šablon za izradu mrežno baziranih aplikacija, kao i lako ugrađivanje novih i izmena postojećih komponenti.

GWT nudi podršku za određen broj standardnih Java API-a, omogućajući razvoj složenih JavaScript aplikacija, bez velikog poznavanja JavaScript programskog jezika, kao i asinhronu komunikaciju sa poslužiocem (AJAX), takođe bez poznavanja istog, jer će se sav Java kod prevesti u optimizovan JavaScript kod.

2.6 MVP projektantski šablon

MVP šablon je nastao kao posledica sve složenijih mrežnih aplikacija.

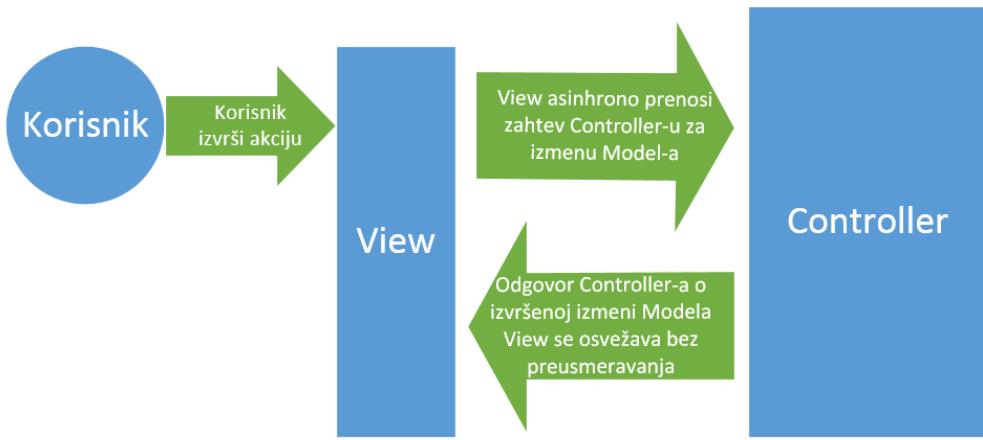
Nastao kao derivat Model-View-Controller šablonu, nudi mogućnosti raslojavanja odgovornosti Controller modula.

Tradicionalne MVC bazirane veb aplikacije zahtevaju da na svaku akciju korisnika odreaguje Controller. Controller po potrebi menja Model i obaveštava View da je došlo do izmene, kako bi se prikaz podataka osvežio. Izmena informacija vezanih za Model zahteva ponovo slanje stranice korisniku, kako bi se osvežio ceo View – slika 2.5.



Slika 2.5: Bazična MVC aplikacija bez asinhronne komunikacije Controller-a i View-a

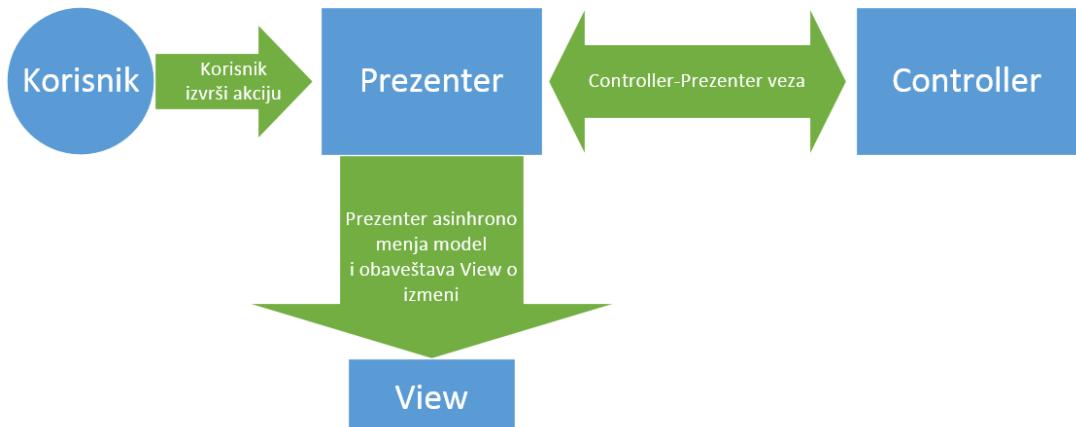
Ponovo učitavanje stranice je za iole složenije stranice neprihvatljivo, što sa korisničke strane, što i sa tehničke. Rešenje jeste da se ugradi deo upravljačke logike u sam View, kako bi mogao da asinhrono da prenosi korisničke akcije Controller modulu i da isto tako asinhrono sačeka odgovor i da se po potrebi osveži izmenjeni deo, bez potrebe ponovnog učitavanja stranice – slika 2.6.



Slika 2.6: MVC bazirana aplikacija sa asinhronom komunikacijom *View* i *Controller* modula

Ovim se deo odgovornosti *Controller* modula prebacuje na sam *View*, ali *Controller* je i dalje čvrsto vezan za samu korisničku spregu.

Ovakav *Controller* u sebi sadrži aplikativnu logiku kao i obradu logiku za prezentaciju podataka. Ideja MVP šablona jeste da se onaj deo *Controller* modula koji je vezan za prezentaciju podataka izdvoji, da bi se odvojio sloj aplikativne logike od prezentacionog – slika 2.7.



Slika 2.7: MVP projektantski šablon

2.7 Java apleti

Java aplet je program dostavljen korisniku u vidu bajtkoda. Apleti su projektovani da se izvršavaju u okviru internet stranice, sa jednom ključnom razlikom u odnosu na ostale

standardne elemente internet stranice: Java apleti se ne izvršavaju unutar pretraživača, već na Javinoj virtuelnoj mašini, u procesu nezavisnog od samog internet pregledača.

Izvršavajući se u posebnom procesu, aplet može da pristupi sistemu za upravljenje datotekama i vrši izmene, kao i sve druge radnje na koje ima jedan program koji se izvršava na datom računaru. Ovo je već na početku bilo uočeno kao velika opasnost ako se ne kontroliše na neki način, zato većina internet pregledača java aplete ograniči (eng. sand-boxing), da nemaju takve privilegije. Aplet, ako je sertifikovan, ima pravo da pita korisnika za neograničen pristup njegovom računaru i jedino tada ga dobija.

2.8 JSNI - Sprega Java i JavaScript-a

Budući da se programski kod napisan u GWT okruženju kompajlira u JavaScript, omogućeno je pisati JavaScript kod unutar Java metoda. Metoda napisana u Javi može da u sebi sadrži JavaScript kod, pod uslovom da ima modifikator native u svojoj deklaraciji. Modifikator native nije deklaracija JSNI nativne metode, već samo naglašavanje da je kod u metodi napisan u čistom JavaScript-u koji se u slengu zove native .

Zbog ove mogućnosti, moguće je iz JavaScript okruženja zvati Java metode, što je jedno od ključnih mogućnosti neophodnih za realizaciju rešenja opisanog u ovom radu. GWT ovu mogućnost naziva JavaScript Native Interface.

2.9 Apache Commons IO

Apache Commons IO nudi set funkcija, za rad sa datotekama i direktorijumima kao i jednostavnije verzije Java klasa za rad sa tokovima podataka. Pravljenje kopija željenih direktorijuma i datoteka, preimenovanje i slično. Apache Commons IO je deo rešenja zbog značajnog broja operacija sa datotekama i direktorijumima.

2.10 Metaprogramiranje

Metaprogramiranje se formalno definiše kao pisanje programa koji pišu programe, ali je metaprogramiranje u praksi mnogo širi pojam. Šta sve metaprogramiranje podrazumeva u praksi je detaljno opisano u [\[5\]](#) i [\[6\]](#) , ukratko bi se moglo reći da se svako rešenje koje manipuliše izvornim kodom van normalne upotrebe, smatra metaprogramiranjem.

Na primer, u C-u, jedna od primena metaprogramiranja jeste tokom pravljenje generičkih struktura i funkcija koje rade sa njima, slično STL-u i šablonskim klasama u C++-u. U programskom kodu ispod prikazan je definicija, poziv i rezultat izvršavanja jednog makroa za generisanje steka za proizvoljan tip podataka.

```

#define Stack(type, STACK_SIZE) \
typedef struct type##Stack \
{ \
    type mem[STACK_SIZE]; \
    int topIndex; \
} type##Stack; \
Stack(int, 20)

```

Nakon obrade preprocesora, dobićemo:

```

typedef struct intStack
{
    int mem[20];
    int topIndex;
} intStack;

```

Listing 2.1: Primer primene metaprogramiranja u ANSI-C

Većina današnjih modernih programskih jezika visokog nivoa mehanizme za primenu metaprogramiranja. U ovom radu se metaprogramiranje primenjuje kroz upotrebu refleksije u Javi.

2.10.1 Refleksija

Većina principa objektno-orientisanog programiranja je opšte poznata, tj principi kao što su apstrakcija, enkapsulacija, nasleđivanje, polimorfizam, dok je refleksija često nešto za šta se nije ni čulo ili nije dobro razumelo.

Refleksija omogućava inspekciju i modifikaciju ponašanja aplikacije u vremenu izvršavanja. Upotrebom refleksije moguće je dobiti informacije i pristup metodama i poljima neke klase iako nisu javno vidljive. Moguće je po potrebi i promeniti vidljivost željenih članova klase. Ovo se često koristi kada je potrebno ispitati i delove klase koji nisu javno vidljivi.

Refleksija omogućava pravljenje novih objekata, pozivanje njihovih metoda, ne znajući njihovu klasu u vremenu prevođenja, niti imena metoda koje treba pozvati nad datim objektima. Ovo se koristi i u rešenjima za objektno-relaciono mapiranje (Hibernate, Java Enterprise Beans), gde se imena polja i metoda za pristup dinamički dobavljaju pri izvršavanju i generisanju koda za rad sa relacionom bazom podataka.

Da bi bilo jasnije, ispod je dat primer upotrebe Javinog API-a za refleksiju. Imamo jednu jednostavnu klasu sa samo jednom metodom koja ne vraća ništa, dve funkcije ispod

omogućavaju kreiranje novog objekta proizvoljne klase i pozivanje proizvoljne metode tog objekta. Više primera i informacija se može naći u zvaničnoj dokumentaciji [\[7\]](#).

```
public class SimpleExample{
    public void Hello(){
        System.out.println("Hi there!");
    }
}

static Object createObject(String className){
    return Class.forName(className).newInstance();
}

static void callMethod(Object obj, String method, Object params, Class[]
paramTypes){
    Method m = obj.getClass().getMethod(method, paramTypes);
    m.invoke(obj, params);
}
Object simpleObject = createObject("SimpleExample");
callMethod(simpleObject, "Hello", null);
```

Listing 2.2: Primer primene refleksije u Javi

Refleksija naravno nosi i određenu cenu, budući da ceo kod nije poznat u vremenu prevođenja, on se mora interpretirati u vremenu izvršavanja, bez mogućnosti ikakve optimizacije, što obično znači oko 700-1000 puta sporije izvršavanje [\[8\]](#). Refleksija je moćan alat koji može mnogo toga doneti, ako se ispravno koristi.

2.11 Poliglotno programiranje

Pod poliglotnim programiranjem se obično podrazumeva upotrebu nekog jezika posebne namene uz glavni, jezik opšte namene. Tipičan primer je razvoj Java EE web aplikacija, gde je Java jezik opšte namene, a jezici posebne namene su obično SQL, XML i JavaScript.

Napredovanjem tehnologije i sve većim usvajanjem metodologija brzog razvoja programske podrške, poliglotno programiranje dobija novo značenje – upotreba više jezika opšte namene u okviru jednog programskog rešenja. Kako bi imali što lakši kod za održavanje, on mora biti kratak, jednostavan i razumljiv, što postižemo apstrakcijom i korišćenjem jezika visokog nivoa. Dinamični jezici se često nameću kao logičan izbor (Haskell, Python, Ruby, Perl). Sa druge strane jedna od najvećih prednosti Java, jeste njena platforma, Java virtuelna mašina, na kojoj su portovane ili originalno za nju napisane, bajt-kod verzije interpretera i prevodilaca popularnih programskih jezika visokog nivoa i neki od njih su prikazani u tabeli 1:

r. broj	Naziv programskog jezika	Implementacija u JVM
1	Haskell	JHaskell
2	Ruby	JRuby
3	Python	Jython
4	Scheme	JScheme

Tabela 2.1: JVM jezici

Kombinovanje Java i drugih programskih jezika baziranih na JVM omogućava pojednostavljenje standardnih delova koda, viši nivo apstrakcije i generalno lakši kod za održavanje.

U ovom radu ovo omogućava integraciju skriptinga u Java EE veb aplikaciju, imajući na raspolaganju JVM implementaciju Python interpretera - Jython.

2.12 Global Interpreter Lock i konkurentno programiranje u Jython-u

Određen broj skripting jezika (Python, Ruby) imaju ugrađen mehanizam za sprečavanje problema tokom konkurentnog izvršavanja, sa jedne strane omogućava laku integraciju postojećih rešenja koja nisu projektovana za konkurentno izvršavanje, dok sa druge, znato otežava pisanje paralelnih programa. Ovaj mehanizam se zasniva na uzajamno isključivim bravama koje ne dozvoljavaju da se isti kod izvršava u više od jedne niti. Pod zaštitom je sav kod, osim specifično napisanih delova koda koji zahtevaju rad sa ulazno-izlaznim uređajima, vremenski zahtevne obrade podataka, koji se izvršavaju van GIL-a [3].

Fredrik Lundh je u svom članku o sinhronizaciji programskih niti u Python-u [9] pokrio ponašanje CPython interpretera tokom konkurentnog izvršavanja. Njegova opažanja su sažeta u dokumentaciji Jython-a [3], gde je istaknuto da su u CPython interpreteru sledeće operacije atomske :

1. Čitanje i modifikacija sadržaja promenljive ili polja objekta neke klase
2. Čitanje i modifikacija sadržaja globalne promenljive

-
- 3. Pristup elementu iz liste
 - 4. Modifikacija liste (dodavanje, brisanje elemenata liste)
 - 5. Pristup i modifikacija vrednosti rečnika

Budući da Python biblioteke i kod specifično napisan za CPython interpreter se oslanja na ove mogućnosti, Jython takođe garantuje atomsko izvršavanje navedenih operacija. S tim da Jython nema GIL. Problem konkurentnog pristupa je rešen implementacijom Python rečnika kao konkurentnih heš mapa. Ovim je omogućena eksploracija višeprocesorskih sistema, ali treba imati u vidu da ovo nije idealno rešenje. Jer na primer, iteracija nije atomska, tako da su mrtve petlje, trke do podataka, moguće pri konkurentnom izvršavanju Python koda u Jython okruženju.

Preporučen način za realizaciju konkurentnog izvršavanja u Jython okruženju jeste upotreba Javinog API-a i Java klase koje garantuju bezbedni konkurentni pristup i modifikaciju.

3. Koncept rešenja

3.1 Uvod

U toku istraživanja tehnologija i metoda za izradu programskog rešenja, rađeni su jednostavni primeri - prototipovi određenih segmenata rešenja - za dokaz validnosti koncepta rešenja (eng. proof of concept).

Tok izrade konačnog rešenja je u potpunosti zavisio od upotrebljivosti ovih prototipova, čija je svrha bila da dokažu da je određene probleme moguće rešiti na dati način. Tokom istraživanja, neki od prototipova su se međusobno nadograđivali, dok su drugi bili razvijani nezavisno kako bi se ispitivale različite metode i tehnologije, u cilju nalaženja što boljeg rešenja.

Pregled realizovanih protipova je dat u tabeli 1.

r. br.	Oblik prototipa	Opis i svrha
1	Java aplet	Java aplet sa ugrađenim Jython interptererom, Selenium bibliotekama i upravljačkim programima za internet pregledače. Samostalno se podešava pri samom pokretanju.
2	Java klijent za veb servise	Konzolni java program, svrha mu je demonstracija rešenja povezivanja sa Intent poslužiocem

Tabela 3.1: Pregled realizovanih prototipova u toku istraživanja

U tabeli 1 su navedeni samo oni prototipovi koji su deo konačnog rešenja, tako da će kroz objašnjenje svakog od njih, biti objašnjeno i samo rešenje.

Kao što je u uvodu napomenuto, jedno od zahteva jeste nezavisnost od platforme.

U okviru podsistema, ugrađen je Jython, kako bi se Python datoteke sa scenarijom ispitivanja napisane u moglo izvršavati po zahtevu Java apleta, posredstvom PythonWrapper modula, bez potrebe prethodnog podešavanja Python interpretera na računaru koji koristi ovaj podsistem.

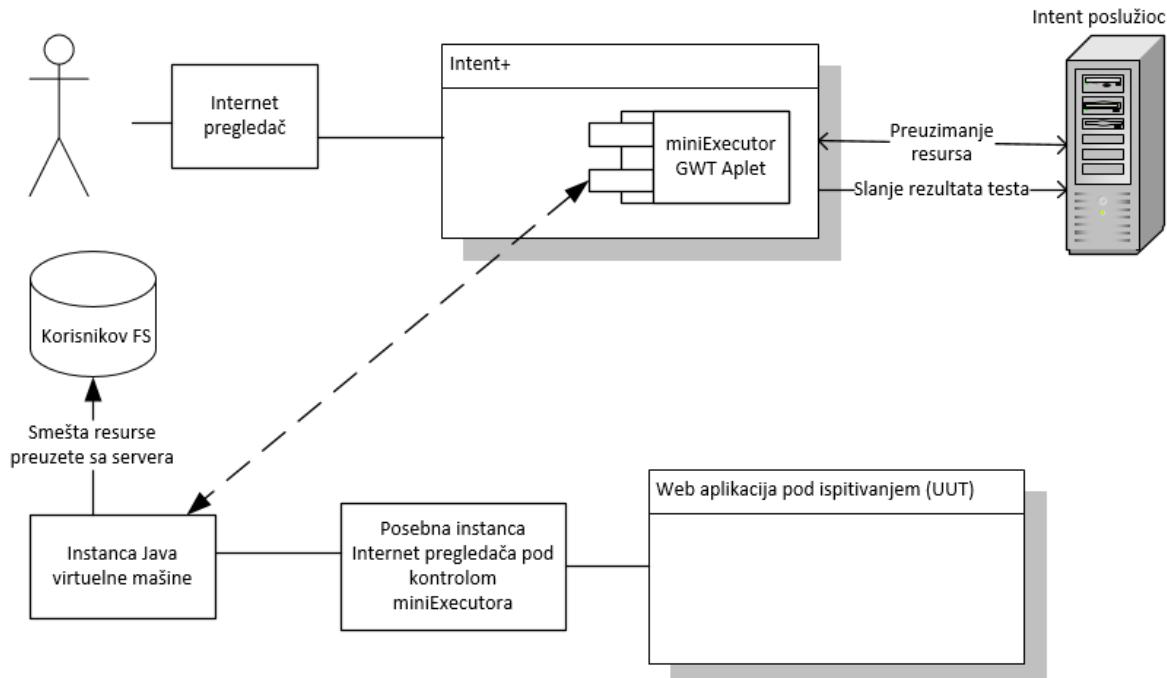
Kontrola internet pretraživača je realizovana upotrebom Seleniumovog mrežnog rukovaoca, kao što je prethodno objašnjeno u drugom poglavlju.

Od ponuđenih upravljačkih programa za internet pregledače, jedino je Firefox upravljački program platformski nezavisni, budući da je realizovan kao modul koji se ugrađuje u sam internet pregledač. Upravljački programi za ostale internet pregledače su implementirani u vidu binarne izvršne datoteke koja izvršava naredbe dobijene preko WebDriver API-a. Ovo u podsistemu uvodi platformsku zavisnost. Za kontrolu svih internet pregledača, osim Firefox-a, potrebno je koristiti upravljački program koji odgovara dатој platformi.

Uloga podsistema jeste da dobavlja scenarije za ispitivanja sa Intent+ poslužiocima, te da ih izvrši i rezultate pošalje Intent poslužiocu.

Podsistem je realizovan kao Java aplet, zbog potreba samostalnog preuzimanja datoteka, pokretanja drugih izvršnih programa i pristupa samom sistemu na kom se izvršava. Šablon za realizaciju spore između Java i Python okruženja je detaljno opisan u [\[3\]](#).

Koncept rada podsistema je prikazan na slici 3.1.



Slika 3.1: Koncept rada podsistema

3.2 Prototip 1 – Java aplet sa ugrađenim Jython interpreterom i Seleniumom

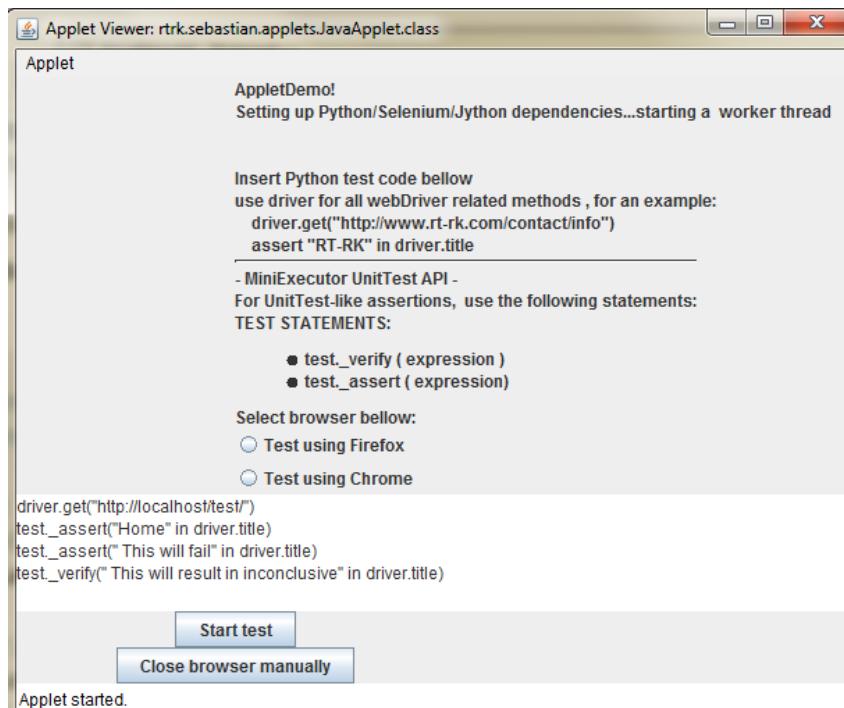
Prvi korak ka izradi rešenja je bila izrada Java apleta koji bi mogao da kontroliše internet pregledač na osnovu korisnički zadatih scenarija za ispitivanje. Scenariji za ispitivanje se zadaju u Python programskom jeziku i aplet uz pomoć Jython-a izvršava kod scenarija za ispitivanje.

Komunikacija Java i Python objekata se ostvaruje uz pomoć sprege, koja omogućava da se metode Python objekata pozivaju u Javi.

Radi što lakšeg ispitivanja funkcionalnosti, odnosno ispravnosti protipa, realizovana je prosta grafička korisnički sprega – slika 10.

Kod unet u tekstualno polje se preuzme klikom na dugme Start test , prosleđuje se PythonWrapper modulu, koji ga izvršava u željenom internet pregledaču i generiše izveštaj sa rezultatima ispitivanja – slika 11.

Kao što je napomenuto u [poglavlju 2.2](#) – moguće je uneti automatski generisan kod za ispitivanje, budući da se koristi isti identifikator za kontrolu internet pregledača.



Slika 3.2: Grafička korisnička sprega za lakše ispitivanje protipa 1

Po ugledu na UnitTest rešenja za testiranje, dostupne su funkcije assert i verify za provere iskaza u toku ispitivanja. Pri zadavanju iskaza, na raspolaganju je skup funkcija definisan u Selenium WebDriver API-u [4].

```
driver.get("http://localhost/test")
test._assert("Home" in driver.title)
test._assert(" This will fail" in driver.title)
test._verify(" This will result in inconclusive" in driver.title)

 test - Notepad
File Edit Format View Help
MINI EXECUTOR TEST REPORT
-----
Tests executed on : Google chrome browser
Ran: 3 tests
Passed: 1 tests
Failed: 1 tests
Inconclusive: 1 tests
=====
Passed tests:
-----
test._assert("Home" in driver.title)
=====
Failed tests:
-----
test._assert(" This will fail" in driver.title)
=====
Inconclusive tests:
-----
test._verify(" This will result in inconclusive" in driver.title)
=====
```

Slika 3.3 : Izveštaj sa rezultatima ispitivanja

3.2.1 Java -Python sprega

Jython omogućava implicitnu konverziju Python objekta u Java objekat.

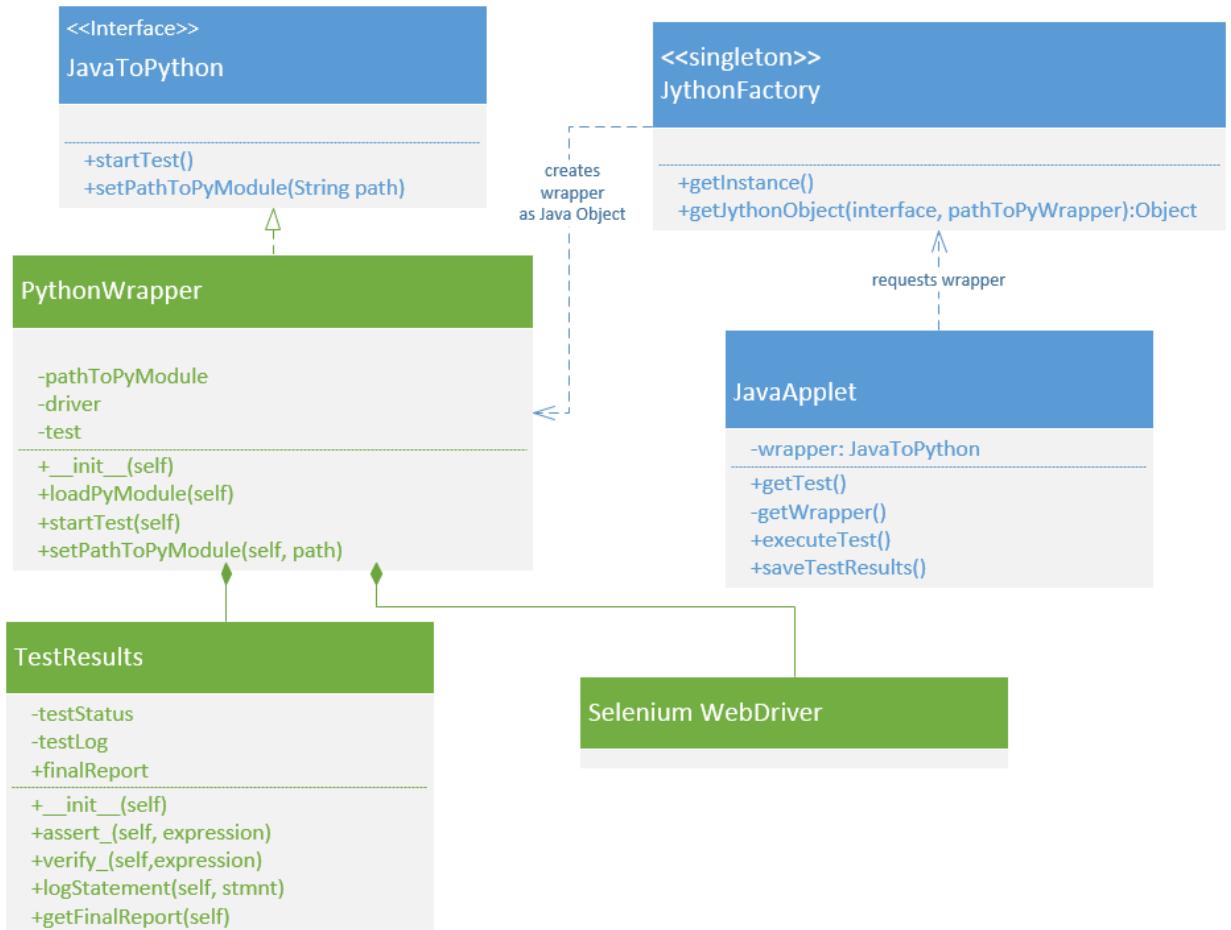
Sprega je realizovana kroz Java interfejs koga Python klasa implementira, što omogućava da se Python objekat koristi preko Java interfejsa.

Klasa koja koristi usluge Jython interpretera i proizvodi zeljene objekte je projektovana po fabričkoj metodi, jer omogućava konverziju Python objekta proizvoljne klase u Java objekat.

Java aplet je zadužen za dobavljanje Python scenarija za ispitivanje koga treba izvršiti, dok će posebna Python klasa, (pod kontrolom Java apleta), dinamički učitavati i izvršavati dobavljene scenarije za ispitivanje. UML dijagram ove sprege je prikazan na slici 12.

Sprega definiše usluge PythonWrapper modula:

1. Dobavljanje željenog internet pregledača za izvršavanje ispitivanja
 2. Održavanje internet pregledača otvorenog do završetka svih ispitivanja
 3. Izvršavanje zadatih scenarija za ispitivanje
 4. Vraćanje izveštaja o rezultatima ispitivanja



Slika 3.4 : Java-Python sprega

3.2.2 UnitTest API za ispitivanje i rezultati ispitivanja

Funkcijama za ispitivanje se pristupa preko test identifikatora, koje su realizovane po ugledu na unittest modul iz standardnih Python biblioteka.

Izrazi čija se istinitost proverava se zadaju kao parameter samih funkcija.

Sam identifikator je zapravo referenca na objekat koji će voditi računa o izvršenim ispitivanjima i generisati izveštaj na kraju ispitivanja.

3.3 Prototip 2 – Java program za komunikaciju sa Intent poslužiocem

3.3.1 Komunikacija sa Intent+ poslužiocem

Uz Intent+ mrežnu aplikaciju, dostupan je i mrežni servis (eng. Web service), koji nudi usluge za lako dobavljanje informacija o scenarijima, planovima i skriptama za ispitivanje, kao i upis rezultata u Intent+ bazu podataka. Upotreba usluga se vrši generisanjem klijenta za željeni servis, te se konkretnim uslugama pristupa pozivanjem metoda generisanog klijenta.

Prenos podataka se obavlja putem DTO – Data Transfer Object, objekti koji predstavljaju modele podataka iz baze, odnosno entiteta iz baze podataka. DTO klase se generišu automatski na osnovu šeme Intent+ poslužioca. Usluge koje se koriste su date u tabeli 3.2.

Komunikacija sa Intent+ poslužiocom je apstrahovana u posebnoj klasi – `IntentInterface` – koja obavlja svu komunikaciju sa Intent+ poslužiocem i pakuje informacije iz DTO objekata u oblik zgodan za slanje u JavaScript.

r. br.	Metode WS klijenta od interesa	Opis i svrha
1	<code>String sayHello()</code>	Provera dostupnosti Intent+ poslužioca.
2	<code>Tuser Login(String username, String password)</code>	Služi za prijavljivanje na Intent+, dobijamo DTO sa podacima korisnika ako je logovanje uspešno.
3	<code>List<Texecutiontestplan> LoadExecutionTestPlans (Long TU_ID, Long TTS_ID);</code>	Dobavlja listu planova za ispitivanje za trenutno ulogovanog korisnika.
4	<code>List<Ttestcase> LoadTests (Long TETP_ID, Long TTP_ID, Long TU_ID);</code>	Dobavlja listu scenarija za izvršavanje za traženi plan ispitivanja.
5	<code>List<Ttestscriptfiles> GetTestScriptFiles (Long TTC_ID)</code>	Dobavlja listu datoteka sa izvornim kodom za konkretni scenario ispitivanja.
6	<code>Long SaveTest(...)</code>	Omogućava čuvanje rezultata ispitivanja u Intent+ bazu.

Tabela 3.2 : Spisak usluga Intent+-a koje `IntentInterface` koristi

3.3.2 IntentInterface – apstrakcija komunikacije sa Intent+ poslužiocem

Apletu je neophodno omogućiti sledeće usluge:

1. Prijava korisnika

2. Dobavljanja planova za ispitivanje dodeljenih ulogovanom korisniku
3. Dobavljanje scenarija za ispitivanja za konkretni plan
4. Dobavljanje skripti za kokretni scenario za ispitivanje
5. Upis rezultata ispitivanja u Intent+ bazu podataka

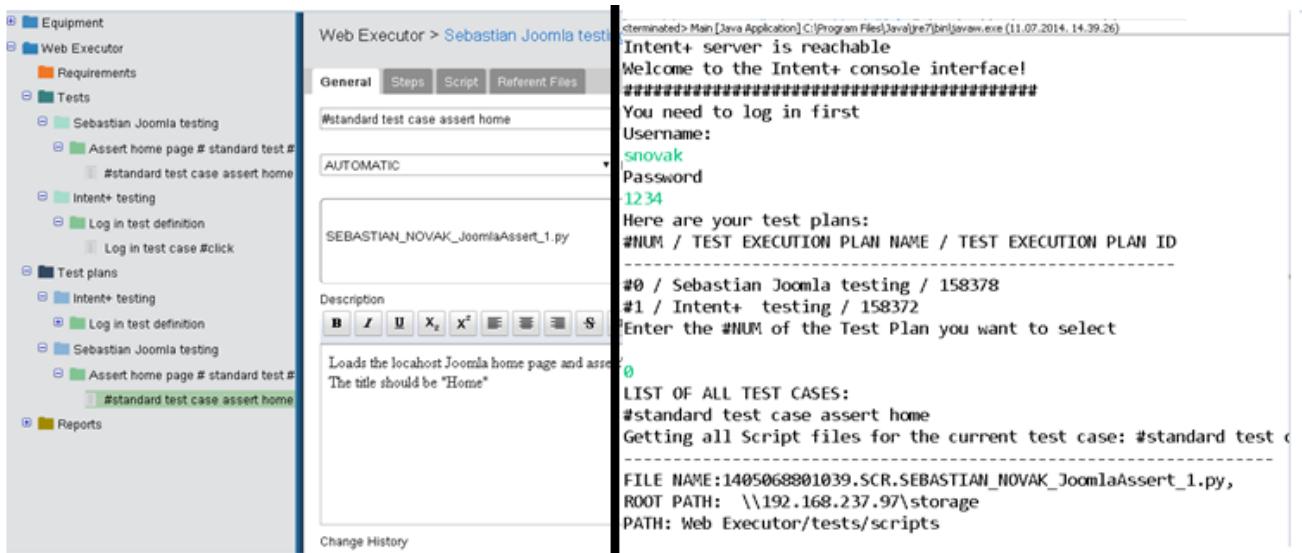
Aplet će informacije dobijene od IntentInterface-a predstavljati na korisničkom interfejsu. Zbog ograničenosti veze između JavaScript-a i apleta, najpre u vidu vrstama i tipovima objekata koji se mogu bezbedno obostranu prenositi, IntentInterface sve informacija vraća kao tekst (moguće razmene tipova su predstavljene u table 5). Kolekcije DTO objekata se spajaju u jedan string koga JavaScript može da razdvoji na jednostavan način.

Usluge API-a su predstavljene u tabeli 4.

r. br.	Naziv metode IntentInterface-a	Opis i svrha
1	<code>void login(String username, String password) throws WebServiceException, ConnectException, InvalidLoginException</code>	Vrši prijavu na Intent+ sistem, u slučaju greške, baca se odgovarajući izuzetak sa informacijama o grešci.
2	<code>String getTestPlansAsStr() throws NotLoggedInException, ServerUnreachableException, NoAssignedTestExecutionPlans</code>	Dobavlja planove ispitivanja za trenutno ulogovanog korisnika, u slučaju greške baca se odgovarajući izuzetak. Ako korisnik nema nijedan dodeljen plan, takođe će biti bačen izuzetak.
3	<code>String getTestCasesAsStr(int testPlanIndex) throws NotLoggedInException</code>	Dobavlja listu scenarija za ispitivanje za zadati plan ispitivanja za trenutno ulogovanog korisnika kao jedan string.
4	<code>String getTestScriptsAsStr(int testCaseIndex) throws NotLoggedInException</code>	Dobavlja listu scenarija za izvršavanje za traženi plan ispitivanja.
5	<code>String getTestScriptAsStr(int testScriptIndex) throws MalformedURLException, IOException, DirectoryException, NotLoggedInException</code>	Putem HTTP-a vrši preuzimanje sadržaja tražene skripte. Datoteka se smešta u privremeni direktorijum korisnika, zatim se njen sadržaj očitava i šalje nazad apletu.
6	<code>SaveTestResults(boolean status, String report) throws NotLoggedInException</code>	Apstrahuje čuvanje rezultata ispitivanja.

Tabela 3.3: API za komunikaciju sa Intent+ poslužiocem

Prototip 2 je realizovan kao konzolna aplikacija koja uz koristi usluge IntentInterface-a i prikazuje rezultate izvršavanja u konzoli . Na slici 13, sa leve strane je predstavljen izgled same Intent+ web aplikacije, dok su sa desne strane predstavljeni podaci preuzeti sa Intent+ poslužioca koristeći usluge IntentInterface-a.



Slika 3.5: Prototip 2 – Konzolna aplikacija - sprega sa Intent+ sistemom

4. Programsko rešenje

Cilj rešenja, odnosno podsistema, jeste da pruži dodatnu funkcionalnost Intent+ okruženju: Mogućnost dodavanja novih scenarija za ispitivanje veb baziranih aplikacija, izmenu postojećih, kao i njihovo izvršavanje i prikupljanje rezultata direktno iz Intent+ okruženja.

Podsistem će biti ugrađen u Intent+ okruženje i da bi postupak ugrađivanja bio što lakši, korisnička sprega rešenja je, kao i Intent+ korisnička sprega, zaosnovana na GWT-u.

Java aplet koji predstavlja srce podsistema je skriven od korisnika i kontroliše se uz pomoć korisničke sprege, koja njegove metoda poziva posredstvom JSNI-a.

Nivoi komunikacije su predstavljeni na slici 4.1.



Slika 4.1: Nivoi komunikacije podsistema

Aplet pre početka rada, mora da podesi sistem korisnika, odnosno da na njega smesti određene resurse neophodne za njegovo izvršavanje.

Da bi podsistem imao mogućnost kontrole Chrome internet pregledača, mora da smesti potreban upravljački program na računar korisnika. Osim upravljačkog programa, na računar korisnika se smeštaju datoteke Selenium okruženja i Python datoteke koje su deo samog podsistema.

Komunikacija između apleta i GWT-a korisničke sprege se odvija putem asinhronih poziva funkcija odnosno callback mehanizma.

GWT strana koja poziva metodu apleta, uz poziv mora da prosledi naziv globalno vidljive JavaScript funkcije koju će aplet pozvati po završetku svoje metoda. Da bi se omogućio brz povratak GWT metodi, da bi ona mogla da nastavi da reaguje na akcije korisnika, sve metode apleta koje se pozivaju iz GWT dela, se izvršavaju u posebnoj niti.

4.1 GWT Korisnička sprega

Grafička korisnička sprega je bazirana na GWT šablonu, prethodno predstavljenom i objašnjrenom u [poglavlju 2.6](#). Podsistem poseduje View i Presenter, dok Controller , napisan samo radi testiranja realizovanog podsistema, jer će se kasnije ugraditi u Intent+ web aplikaciji i biti pod kontrolom njenog Controller modula. View je implementiran kao widget koji se može dodati željenom kontejneru. Važno je napomenuti da View pri učitavanju dodaje i HTML kod java apleta u RootLayoutPanel – glavni kontejner za sve komponente GWT korisničke sprege.

4.1.1 View

Ceo View je modelovan jednom klasom – MiniExecutorPanel, koja služi kao kontejner za glavne delove aplikacija, organizovanih po karticama:

1. Kartica za izvršavanje testa - ExecutionTab
2. Kartica za biranje skupa testova prijavom na Intent+ sistem - TestSelectionTab

Uloga View modula jeste da prenosi korisničke naredbe prezenteru i prikaz podataka. MiniExecutorPanel implementira spregu View, koja sadrži spisak metoda koje su neophodne prezenteru za efektivnu kontrolu MiniExecutorPanel-a. Klikom na bilo koje dugme, obrada svake korisničke radnje se prosleđuje na obradu prezenteru pozivom njegove metode.

Spisak metoda koje MiniExecutorPanel implementira da bi omogućio kontrolu prezenteru je dat u tabeli 4.1.

r. br.	Deklaracija metode u View sprezi	Opis i svrha
1	void setPresenter(Presenter p)	Čuvanjem referencu na svog prezentera, MiniExecutorPanel može da pri svakoj akciji korisnika zove odgovarajuću metodu prezentera.
2	void bind()	Deo MVP šablonu, omogućava vezivanje View-a i njegovog prezentera. Dodaje rukovaće korisničkih akcija, koji će pozivati metode prezentera.
3	public Widget asWidget()	Deo MVP šablonu, omogućava prezenteru da smesti View u korenski ili neki roditeljski kontejner.
4	void showOK()	Signaliziranje uspešno izvršenog ispitivanja
5	void showError()	Signaliziranje neuspešno izvršenog ispitivanja
6	getSelectedBrowser()	Dobavljanje izabranog internet pregledača u kom će se izvršavati scenario za ispitivanje
7	JavaScriptObject getApplet()	Dobavljanje MiniExecutor apleta, kako bi prezenter mogao da posredstvom JSNI-a poziva njegove metode.
8	void setStatusText(String text)	Postavljanje izveštaja o detaljima izvršenog testa
9	void setEditorContent(String pyCode)	Postavljanje sadržaja editora.
10	String getEditorContent()	Dobavljanje sadržaja editora.
11	void setErrorAnotation(int lineNumber, int lineColumn, String errorMsg)	Postavljane anotacije u editoru, na dатој liniji koda, sa datom porukom.
12	void clearErrorAnotation()	Uklanjanje anotacije iz editora.
13	void setTestExecutionPlans(String[] testExecutionPlans)	Postavljanje sadržaja komponente za izbor plana ispitivanja.
14	void setTestCases(String[] testCases)	Prikaz scenarija za ispitivanje koji će se izvršiti za izabrani plan ispitivanja.

Tabela 4.1: View sprega

Radi dekompozicije grafičke sprege, kartice su modelovane posebnim klasama, koje u sebi ne sadrže nikakvu logiku, već predstavljaju kolekciju kontrola korisničke sprege koje su pod kontrolom MiniExecutorPanel-a.

4.1.1.1 ExecutionTab – kartica za izvršavanje scenarija za ispitivanje

Realizovan nasleđivanjem klase Composite Widget jer pakuje sve elemente korisničke sprege koji treba da se nađu u kartici za izvršavanje ispitivanja. Bez ovog pakovanja, svaki od ovih elemenata bi se morao kontruisati unutar konstruktora samog MiniExecutorPanel-a, što bi njegov konstruktor učinilo dosta nepreglednim. Ne sadrži nikakvu logiku, ima get/set metode za sve delove korisničke sprege koje MiniExecutorPanel može da preuzme i prosledi akcije korisnika Presenter-u. Izgled ove kartice je prikazan na slikama 4.2 i 4.3.

Glavni delovi ove komponente jesu:

1. Glavni meni sa alatkama, za izvršavanja koda za ispitivanja u izabranom pretraživaču, kao i mogućnost učitavanja sadržaja datoteke sa izvornim kodom za ispitivanje i smeštanje tog sadržaja u editor.
2. Editor je implementiran uz pomoć AceEditor-a [\[11\]](#).
3. Statusni meni, grafički prikaz statusa izvršenog testa i izveštaj izvršenog testa.

4.1.1.1.1 Učitavanje sadržaja korisnikove datoteke sa izvornim kodom

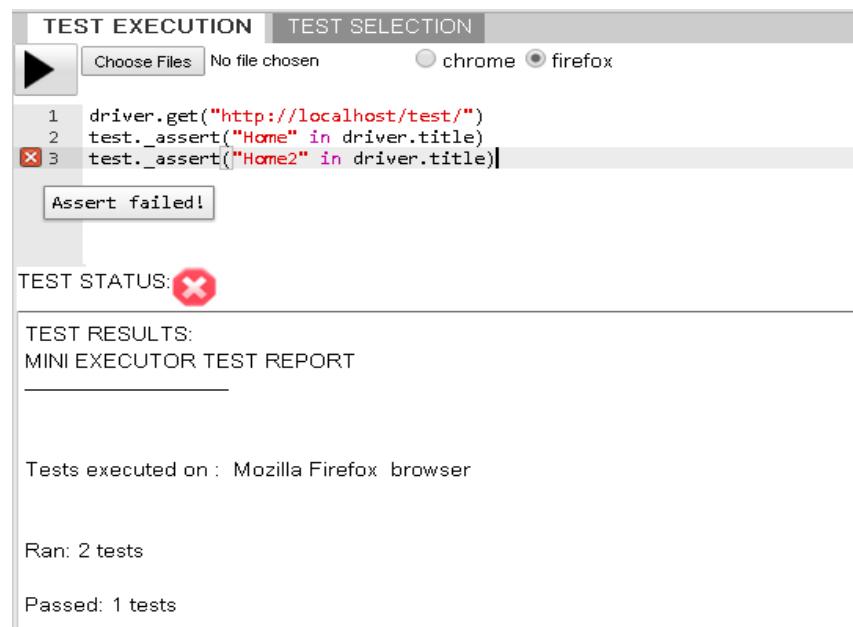
Korisnik klikom na dugme “Choose files” (slike 15 i 16) može da otpremi datoteku MiniExecutor aplikaciji. Da se korisnik ne bi šetao sa jedne strane na drugu, otpremanje datoteke se vrši asinhrono preko AJAX-a, sadržaj datoteke se kasnije preuzima sa poslužiocu kao odgovor servera na AJAX zahtev i postavlja kao sadržaj editora. Ovo je sve već implementirano u biblioteci GWTUpload [\[12\]](#), pa je otpremanje realizovano integracijom iste u MiniExecutor.

4.1.1.1.2 AceEditor – JavaScript baziran editor za isticanje sintakse programskog koda

Editor MiniExecutor-a je morao da podrži isticanje sintakse prikazanog programskog koda.

Zbog širokih mogućnosti, lako ugrađivanje u web aplikacije, izabran je popularni JavaScript baziran editor - AceEditor. Da bi se olakšalo rukovanje sa njim, napisan je AceWrapper, klasa koja će upotrebom JSNI metoda dodati AceEditor zadatom roditeljskom kontejneru i omogućiti da se usluge AceEditor-a koriste iz java. Svaka javno dostupna metoda ove klase, svoju uslugu realizuje pozivajući privatne JSNI metode, koje pristupaju odgovarajućim JavaScript metodama AceEditora.

Editor takođe omogućava označavanje linije poslednjeg neuspele naredbe za ispitivanje ili sintaksne greške (slika 15).



The screenshot shows a test execution interface. At the top, there are tabs for 'TEST EXECUTION' and 'TEST SELECTION'. Under 'TEST EXECUTION', there is a play button icon, a 'Choose Files' button, and a dropdown for browser selection with 'chrome' and 'firefox' options. The 'firefox' option is selected. Below this, a code editor displays three lines of test code:

```

1 driver.get("http://localhost/test/")
2 test._assert("Home" in driver.title)
3 test._assert("Home2" in driver.title)

```

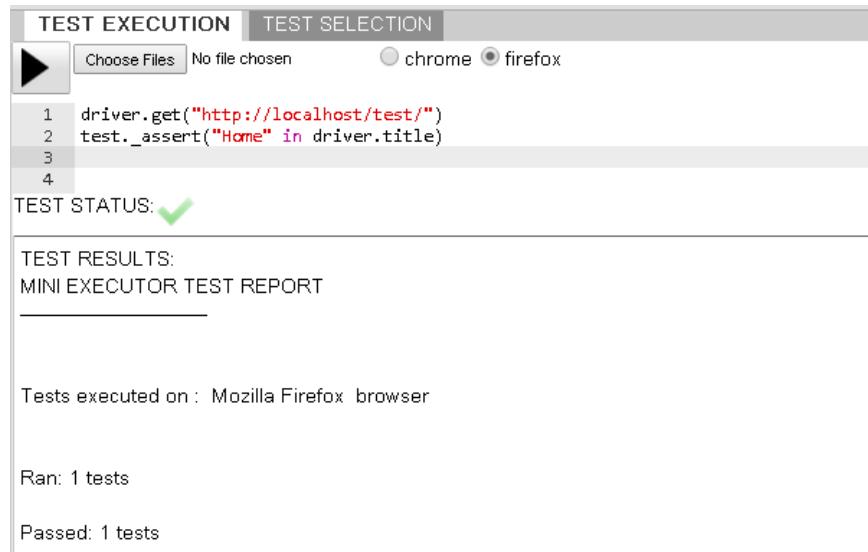
A red 'X' icon is placed next to the third line, indicating it failed. A message box below the code editor says 'Assert failed!'. The 'TEST STATUS:' section shows a red 'X' icon. The 'TEST RESULTS:' section is titled 'MINI EXECUTOR TEST REPORT'.

Details below the results table include:

- Tests executed on : Mozilla Firefox browser
- Ran: 2 tests
- Passed: 1 tests

Slika 4.2: Izgled kartice za izvršavanje ispitivanja nakon neuspelog izvršenja scenarija za ispitivanje

Iako postoji već gotovog rešenja za ugrađivanje AceEditor-a u GWT - AceGWT [13] , izabrana je implementacija softvrene integracije, budući da je postojeće rešenje preobimno i da se za potrebe MiniExecutor-a koristi samo mali deo mogućnosti AceEditora.



The screenshot shows a test execution interface similar to Slika 4.2. The 'TEST EXECUTION' tab is active. The code editor displays the same three lines of test code as in Slika 4.2. The 'TEST STATUS:' section now shows a green checkmark icon. The 'TEST RESULTS:' section is titled 'MINI EXECUTOR TEST REPORT'.

Details below the results table include:

- Tests executed on : Mozilla Firefox browser
- Ran: 1 tests
- Passed: 1 tests

Slika 4.3: Izgled kartice za izvršavanje ispitivanja nakon uspešnog izvršenja scenarija za ispitivanje

4.1.1.2 TestSelectionTab – kartica za prijavu na Intent+ i izbor plana ispitivanja

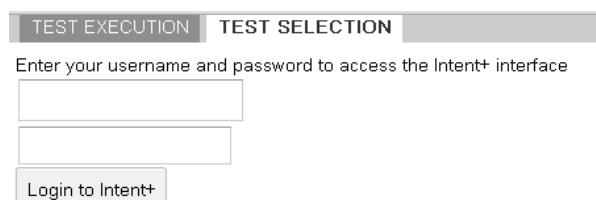
Ova kartica u sebi sadrži deo korisničke sprege neophodan za izbor i izvršavanja plana ispitivanja. Za početak rada neophodno je prijaviti se, kako bi se isti podaci mogli upotrebiti pri dobavljanju podataka sa Intent+ poslužioca. Pri uspešnoj prijavi na Intent+, preuzimaju se svi planovi za ispitivanje koji su dodeljeni prijavljenom korisniku.

Korisnik može da izabere koji od ponuđenih planova za ispitivanje želi da se automatizovano izvrši, nakon čega će se dobaviti svi scenariji za ispitivanje u okviru izabranog plana za ispitivanje i početi sa njegovim izvršavanjem. Korisnik takođe može izabrati u kojim od ponuđenih internet pregledača je potrebno da se scenariji za ispitivanje izvrše (Firefox, Chrome).

U toku izvršavanje plana ispitivanja, na korisničkoj sprezi će se prikazivati ukupan napredak i koliko je još scenarija za ispitivanje ostalo da se izvrši u okviru trenutnog plana.

Izvršavanje plana ispitivanja je moguće obustaviti klikom na dugme STOP.

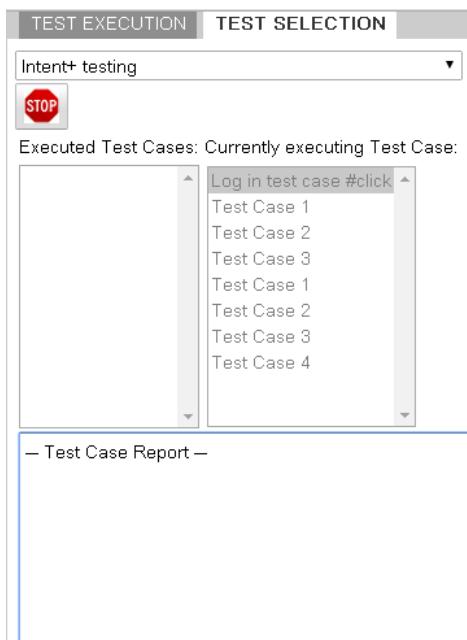
Kada se svi scenariji za ispitivanje unutar izabranog plana izvrše, moguće je sačuvati rezultate ispitivanja. Moguće je sačuvati pojedinačni rezultat izvršenog scenarija za ispitivanje ili celog plana, odnosno svih scenarija za ispitivanje.



Slika 4.4 : Početni izgled kartice, prijava na Intent+



Slika 4.5: Nakon uspešne prijave, prikaz planova ispitivanja



Slika 4.6: Izvršavanja svih scenarija za ispitivanje u okviru izabranog plana za ispitivanje

4.1.2 Presenter

Prezenter rukuje naredbama korisnika, kontroliše View, poziva metode apleta kako bi izvršio izvorni kod iz editora, prijavio korisnika u Intent+ sistem i započeo izvršavanje planova za ispitivanje i svih scenarija unutar konkretnog plana.

Prezenter implementira spregu koja daje spisak metoda koje View treba da poziva pri akciji korisnika. Spisak metoda definisanih u Presenter sprezi je dat u tabeli 5.

r.br	Deklaracija metode	Opis
1	<code>void go(final HasWidgets container)</code>	Deo MVP šablona, svaki prezenter mora imati ovu metodu, započinje povezivanje Prezentera, View-a i kontejnera u koji će se smestiti View
2	<code>void onSaveClick(int testIndex)</code>	Čuvanje rezultata pojedinačnog scenarija za ispitivanje
3	<code>void onSaveAllClick()</code>	Čuvanje rezultata svih izvršenih scenarija za ispitivanje
4	<code>void onRunClick()</code>	Započinje izvršavanje koda za ispitivanje unetog u editor

5	<code>void onLoginClick(String username, String password)</code>	Vrši prijavu na sistem na osnovu informacija prosleđenih iz View-a
6	<code>void executeTestPlan (int selectedPlanIndex)</code>	Dobavlja sve scenarija i skripte asocirane sa njima i počinje da ih izvršava

Tabela 4.2: Presenter sprega

4.1.3 Izvoz GWT metoda u JavaScript

Kao što je prethodno objašnjeno na početku ovog poglavlja, java aplet po završetku svojih metoda poziva globalno vidljive JavaScript funkcije. GWT metode prezentera nisu globalno vidljive, tako da je neophodni uraditi „izvoz“ ovih metoda u JavaScript.

Izvoz GWT metoda omogućava da se one pozivaju kao bilo koja druga JavaScript funkcija, na primer od strane neke JavaScript biblioteke ili kao u ovom radu, java apleta.

Pošto je ovo neretka situacija pri radu sa GWT-om, zvanična dokumentacija GWT projekta obezbeđuje šablon kako ovo treba da se realizuje [\[10\]](#).

Prema šablonu, potrebno je definisati globalne pokazivače na anonimne JavaScript funkcije koje pozivaju GWT metode koristeći njihov potpun potpis. Primer izvoza je dat u listingu 3.

```
private native void exportAppletCallbacks(MiniExecutorPresenter that) {  
    $wnd.testFinishedCallbackFncPtr = $entry(function(isSuccessful, lineNumber,  
errorMessage, results) {  
        that.@com.rtrk.intent.miniexecutor.client.presenter.MiniExecutorPresenter::test  
FinishedCallback(ZILjava/lang/String;Ljava/lang/String;)(isSuccessful, lineNumber,  
errorMessage, results);  
    });  
}
```

Listing 4.1: Izvoz GWT metoda

Java aplet će pozivom zadate JavaScript funkcije, pozivati ove anonimne JavaScript funkcije preko globalnih pokazivača. Tako da se apletu zapravo ne prosleđuju imena JavaScript funkcija koje treba da pozove, već nazivi globalnih pokazivača na anonimne JavaScript funkcije koje pozivaju GWT metode.

4.1.4 JavaScriptToJava sprega

Komunikacija između prezentera i java apleta je realizovana uz pomoć callback mehanizma i zasniva se na mogućnosti uvedenoj u verziji 6 Jave: LiveConnect. LiveConnect omogućava komunikaciju Java apleta i JavaScript okruženja, omogućavajući JavaScript okruženju da poziva metode Java apleta, odnosno Java apletu da poziva JavaScript funkcije. Za detalje komunikacije JavaScript okruženja i Java apleta, videti zvaničnu dokumentaciju o LiveConnect konceptu [\[14\]](#).

Prezenter je dužan da pri pozivu bilo koje metode java apleta kao parametar prosledi i ime globalno vidljive JavaScript funkcije koju će metoda apleta pozvati kada završi sa svojim radom.

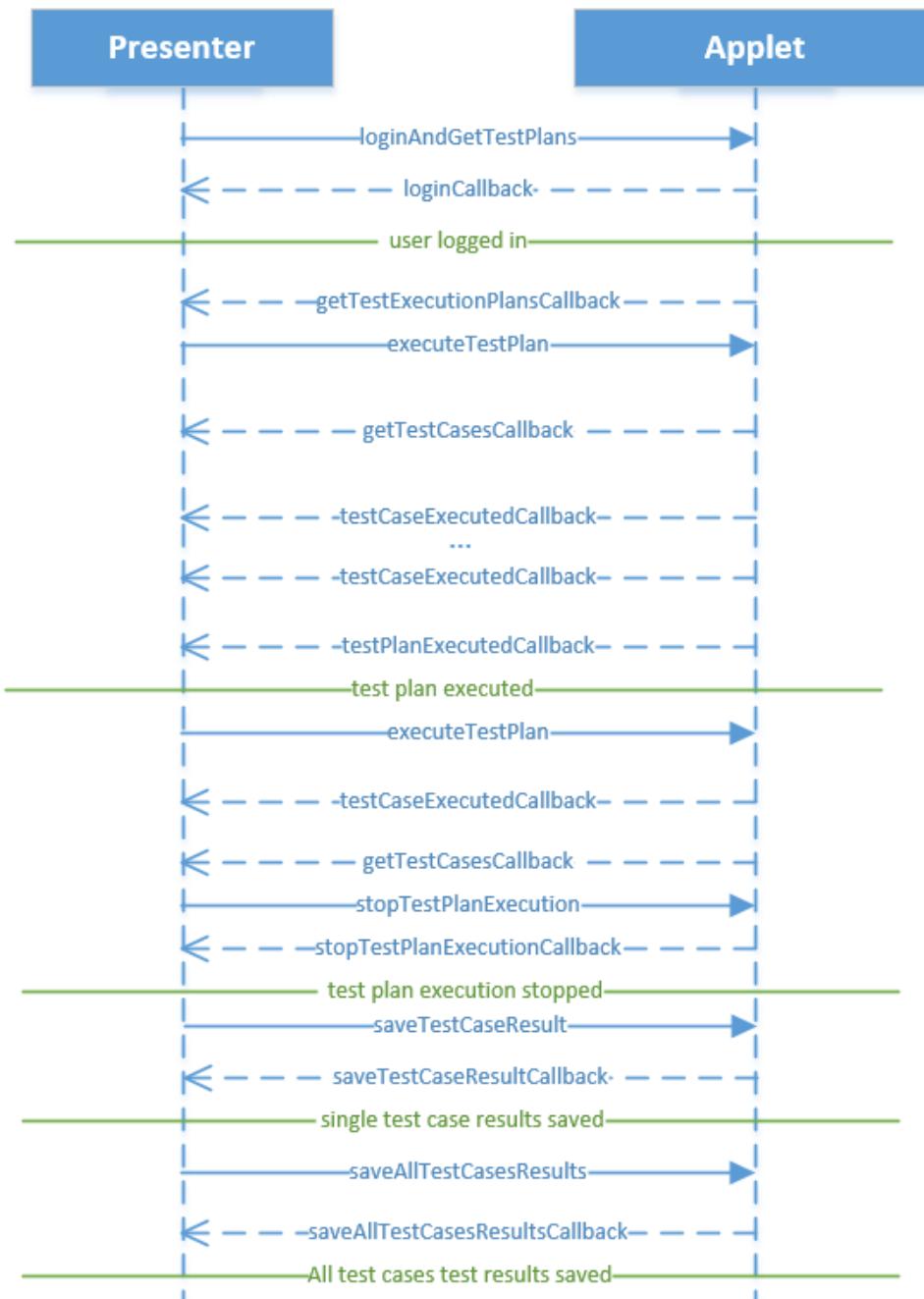
Dijagram na slici 4.7 prikazuje redosled i koncept poziva metoda java apleta i zadatih JavaScript funkcija pri njihovom završetku.

4.2 MiniExecutorAplet

Nakon inicijalizacije i podešavanja sistema, aplet je dostupan za upotrebu.

Uloge apleta jesu:

1. Instanciranje Jython interpretera i PythonWrapper-a.
2. Izvršavanje datog koda scenarija za ispitivanje
3. Prekid testa kod prvog neuspele provere u toku ispitivanja i vraćanje informacija o neuspeloj proveri.
4. Komunikacija sa Intent+ serverom



Slika 4.7: Komunikacija GWT-a i apleta

Korisne metode apleta su predstavljene u JavaScriptToJava sprezi, prikazane u tabeli 4.3.

Parametri ove metode jesu izvorni kod za ispitivanje, koga treba izvršiti, kod izabranog pretraživača i ime JavaScript funkcije koju treba pozvati nakon izvršavanja ispitivana, radi prikaza informacija o rezultatima izvršavanja scenarija za ispitivanje.

4.2.1 Instanciranje Jython interpretera i PythonWrapper-a

Aplet koristi usluge modula JythonFactory, koji je realizovan kao proizvođač željenih proizvoda. Proizvodi su Python objekti pretvoreni u Java objekte i kojima se pristupa preko IJavaToPython sprege.

JavaToPython sprega definiše API za kontrolu PythonWrapper-a, spisak metoda je naveden u tabeli 5.

r.br	Deklaracija metode	Opis
1	<code>String executeTest(String testCode, int selectedBrowsers)</code>	Izvršava kod scenarija za ispitivanje u izabranim internet pregledačima
2	<code>String setChromeDriverPath</code>	Pri podešavanju sistema, podešava putanju do chrome mrežnog rukovaoca
3		

Tabela 4.3: API za kontrolu PythonWrapper-a

4.2.2 JythonFactory

JythonFactory nudi usluge konverzija Jython objekata u zadate Java objekte, ako je to moguće (neophodno je da to Jython objekat nasleđuje željenu Java klasu u koju će biti kastovan).

Upotreboom refleksije, pravi se univerzalan mehanizam za konverziju, budući da se ime klase može zadati kao tekst.

Metoda `__to_java__` koju poseduje svaki Jython objekat, takođe koristeći refleksiju, obavlja konverziju Jython objekta u Java objekat.

4.2.3 Podešavanje sistema

Zbog pristupa sistemu datoteka korisnika, aplet je morao biti potpisani i ne može započeti rad dok korisnik ne dozvoli neograničen pristup njegovom sistemu.

Aplet pri inicijalizaciji, u posebnoj programskoj niti, pravi direktorijum za sebe u privremenom direktorijumu korisnika.

Unutar svog direktorijuma, aplet smešta PythonWrapper skriptu, Chrome upravljački program i datoteke Selenium okruženja. Rad sa sistemom datoteka korisnika i resursima apleta je realizovan u modulu Utility.

4.3 PythonWrapper

Uloga ovog modula je omogućavanje komunikacije između delova rešenja napisanih u Python-u i Javi. Po dobijenom zahtevu od apleta, učitava, izvršava i javlja rezultate datog scenarija za ispitivanje.

Značajne metode modula su deklarisane u sprez IJavaToPython i prikazane u tabeli 2.

PythonWrapper apletu vraća sledeće tipove podataka:

1. Podatke prostog tipa (boolean, int)
2. Python objekte konvertovane u Java objekte implicitnom konverzijom
3. Java objekte
4. Kolekcije Java objekata unutar Java ArrayList objekata

Jython omogućava da se unutar Python modula stvaraju Java objekti, da se koriste Java objekti dobijeni od neke Java metode i pravljenje kolekcije Java objekata uz pomoć ArrayList.

4.3.1 Ručno izvršavanje scenarija za ispitivanje

Kod se zadaje u obliku Stringa, gde je neophodno naredbe razdvojiti prelaskom u novi red. Učitani kod se unutar metode `executeTest` razbija na delove po redovima i šalje se interpreteru na izvršavanje metodom `exec(statement)`.

Metoda `exec(statement)` pri nailasku na sintaksu ili leksičku grešku, baca izuzetak koji se hvata i obrađuje unutar metode `executeTest`. Obradom greške izvlače se informacije o grešci – poruka interpretera kao i broj linije na kojoj se greška desila. Informacije o pojedinačnoj grešci su enkapsulirane unutar objekata klase `PyError`.

Metoda `getErrors()` vraća Java `ArrayList` objekata klase `PyError`, što predstavlja greške pronađene tokom poslednjeg izvršavanja metode `executeTest`.

Sve informacije o grešci se smeštaju u izuzetak klase `SyntaxErrorException` i on se baca apletu.

Pri nailasku na neuspelu proveru, `PythonWrapper` će napraviti izuzetak klase `AssertFailedException`, smestiti u njega informacije koja je provera neuspela i baciti ga apletu.

4.3.2 Propagacija izuzetaka

Kao što je prethodno rečeno, Jython omogućava da se u okviru Python okruženja prave Java objekti.

Kad je reč o izuzecima, nema razlike, prvo se naprave, napune odgovarajućim informacijama i zatim se bace. Jython poseduje mehanizam za propagaciju izuzetaka između Java i Python okruženja. Izuzetak koji se desi u Python okruženju a ne obradi se, će biti unutar klase `PyException`, koga može da uhvati Java okruženje koje ga je pozvalo i da kasnije iz njega izvuče pravi izuzetak.

```

try {
    pythonWrapper.executeTest(testCode);
} catch(PyException pyEx) {
    /* the PyException wraps the Java Exception that happened in Python */
    Object javaExp = pyEx.value.__tojava__(Exception.class);
    if (javaExp instanceof SyntaxErrorException){
        throw (SyntaxErrorException)javaExp;
    } else if (javaExp instanceof AssertFailedException){
        throw (AssertFailedException)javaExp;
    } else {
        throw (Exception)javaExp;
    }
}

```

Listing 4.2: Propagacija izuzetka bačenog iz Python-a

4.3.3 Sigurnosna ograničenja

Pri svom radu, MiniExecutor aplet pristupa mreži, preuzima datoteke i druge informacije sa Intent+ server, pristupa disku korisnika, smeštajući resurse neophodne za svoj rad u korisnikog direktorijum za privremene datoteke.

Zbog ovoga MiniExecutor aplet mora da bude potpisani i ovlašten aplet. Problem nastaje kada se metode apleta pozivaju iz JavaScript okruženja , gubi se status potpisanih i ovlašćenih apleta. Ovo je prevazideno upotrebom izvršavanja u privilegovanim režimu, gde pri prvom pozivu neke od metoda apleta, korisnik će morati da potvrdi dozvolu apletu da bude pozivan iz JavaScript okruženja, nakon čega će opet dobiti neograničen pristup. Važno je napomenuti da će Java, u svojim budućim verzijama, u potpunosti zabraniti pokretanje java apleta koji nisu potpisani zvaničnim sertifikatom.

4.4 TestResults

Tokom izvršavanja scenarija za ispitivanje, naredbe `test.verify__` kao i `test.assert__` se beleže unutar objekta klase `TestResults`. Korisnici pri pisanju scenarija za ispitivanje koriste identifikator `test` kako bi pristupili metodama za utvrđivanje istinosti određenih iskaza.

Metode za utvrđivanje jesu:

1. `test.assert__()`
2. `test.verify__()`

Beleži se kod cele naredbe i njen rezultat. U zavisnosti od rezultata i metode koja je izvršila datu naredbu, zabeleženi kod se kategorise na sledeći način:

1. Utvrđivanja koja su bila uspešna , assert i verify
2. Utvrđivanja koja su bila neuspešna, što predstavlja neuspešna utvrđivanja metode `assert__()`
3. Utvrđivanja koja su bila nejasna, odnosno, neuspešna utvrđivanja metode `verify__()`

Konačni izveštaj se generiše na osnovu zabeleženih naredbi za utvrđivanje i rezultata njihovog izvršavanja. PythonWrapper ovaj izveštaj preuzima pozivom metode `getTestResults()`, gde dobija ceo izveštaj unutar jednog String-a.

5. Ispitivanje

Ispitivanje je urađeno upotrebom Selenium razvojnog okruženja za snimanje akcija korisnika (prethodno predstavljeno u [poglavlju 2.2](#)) tokom upotrebe Intent+ mrežne aplikacije. Generisani Python kod je proširan sa naredbama za utvrđivanje tačnosti željenih iskaza. Prošireni kod za ispitivanje je dodat kao resurs u Intent+ mrežnoj aplikaciji, pod odgovarajućim scenarijom za ispitivanje.

Ako korisnik zatvori neki od kontrolisanih internet pregledača u toku ispitivanja, prijavljuje se greška.

Svi scenariji za ispitivanje se izvršavaju u Firefox i u Chrome internet pregledaču. Uslov da ispitivanje bude uspešno jeste da se dobiju isti rezultati u oba pregledača.

Izvršeni scenariji za ispitivanje su predstavljeni u tabeli 5.1, dok su rezultati predstavljeni u tabeli 5.2.

Naziv scenarija za ispitivanje	Opis	Očekivani ishod
Prijava sa ispravnim podacima na Intent+, klikom na dugme	Nakon unosa ispravnih, važećih podataka, korisničkog imena i lozinke, sledi klik na dugme za prijavu.	Uspešna prijava na Intent+, trenutna lokacija je pregled aktivnih projekata.
Prijava sa ispravnim podacima na Intent+, pritiskom tastera Enter	Pri podešavanju sistema, podešava putanju do chrome mrežnog rukovaoca	Uspešna prijava na Intent+, trenutna lokacija je pregled aktivnih projekata.

Pravljenje novog skupa scenarija za ispitivanje	Nakon uspešnog logovanja, potrebno je pristupiti nekom od trenutno aktivnih projekata koji su dodeljeni trenutno prijavljenom korisniku, zatim desnim klikom na pregled svih skupova scenarija za ispitivanje, napraviti novi skup scenarija za ispitivanje pod imenom TestSuite1 .	Pregledom svih skupova za ispitivanje, treba utvrditi da se među njima nalazi TestSuite1 .
Definisanje novog scenarija za ispitivanje	Nakon uspešnog napravljenog skupa scenarija za ispitivanje, potrebno je dodeliti scenarije skupu. Desnim klikom na ime skupa - TestSuite1 - bira se opcija za definiciju novog scenarija za ispitivanje – TestCase1	Utvrđivanje da se među definicijama scenarija za ispitivanje u okviru TestSuit1 skupa nalazi definicija scenarija za ispitivanje – TestCase1 .
Pravljenje novog plana za izvršavanje ispitivanja	Desnim klikom na TestPlans i izborom opcije new test plan, započinje se pravljenje novog test plana – TestPlan1	Utvrđivanje da se među planovima za izvršavanje ispitivanja nalazi TestPlan1
Dodela skupa scenarija za ispitivanje planu za izvršavanje ispitivanja	Nakon uspešno napravljenog skupa scenarija ispitivanja sa bar jednim scenarijom, moguće je dodeliti skup scenarija za ispitivanje određenom planu za izvršavanje ispitivanja. Prevlačenjem skupa scenarija za ispitivanje TestSuite1 na TestPlan1 će dodati sve	Utvrđivanje da se unutar TestPlan1 plana sada nalazi skupe scenarija TestSuite1 zajedno sa konkrenim scenarijom za ispitivanje - TestCase1

	scenarije za ispitivanje tom planu.	
Dodela scenarija za izvršavanje trenutno prijavljenom korisniku	Unutar plana za izavršavanje, potrebno je dodeliti TestCase1 trenutno prijavljenom korisniku	Utvrđivanje da je TestCase1 dodeljen trenutno prijavljenom korisniku.
Prijava na Intent+ unutar podsistema, preuzimanje spiska planova za ispitivanje	Upotrebom MiniExecutor podsistema vršiće se auto-ispitivanje, odnosno podistem će ispitivati sopstveni rad. Pokretanjem i postavljenjem podsistema na dva lokalna poslužilaca omogućeno je da na jedan podistem bude ispitivan od strane druge.	Unos validnih informacija za prijavu. Utvrđivanje da je u spisku planova za izvršavanje ispitivanja TestPlan1
Pokretanje izvršavanja plana za izvršavanje i provera upisanih rezultata u Intent+ aplikaciji	Nakon dobavljenog plana za izvršavanje ispitivanja, potrebno je pokrenuti ga, sačuvati njegove rezultate i proveriti da li će se rezultati pravilno upisati u Intent+ bazu podataka, odnosno prikazati u Intent+ aplikaciji.	Utvrditi da se rezultati ispitivanja uspešno upisani upisani u Intent+ bazu podataka.

Tabela 5.1: Scenariji ispitivanja

Naziv scenarija za ispitivanje	Rezultat ispitivanja
Prijava sa ispravnim podacima na Intent+, klikom na dugme	Uspešan
Prijava sa ispravnim podacima na Intent+, pritiskom tastera Enter	Uspešan
Pravljenje novog skupa scenarija za ispitivanje	Uspešan
Definisanje novog scenarija za ispitivanje	Uspešan
Pravljenje novog plana za izvršavanje ispitivanja	Uspešan
Dodela skupa scenarija za ispitivanje planu za izvršavanje ispitivanja	Uspešan
Dodela scenarija za izvršavanje trenutno prijavljenom korisniku	Uspešan
Prijava na Intent+ unutar podsistema, preuzimanje spiska planova za ispitivanje	Uspešan
Pokretanje izvršavanja plana za izvršavanje i provera upisanih rezultata u Intent+ aplikaciji	Uspešan

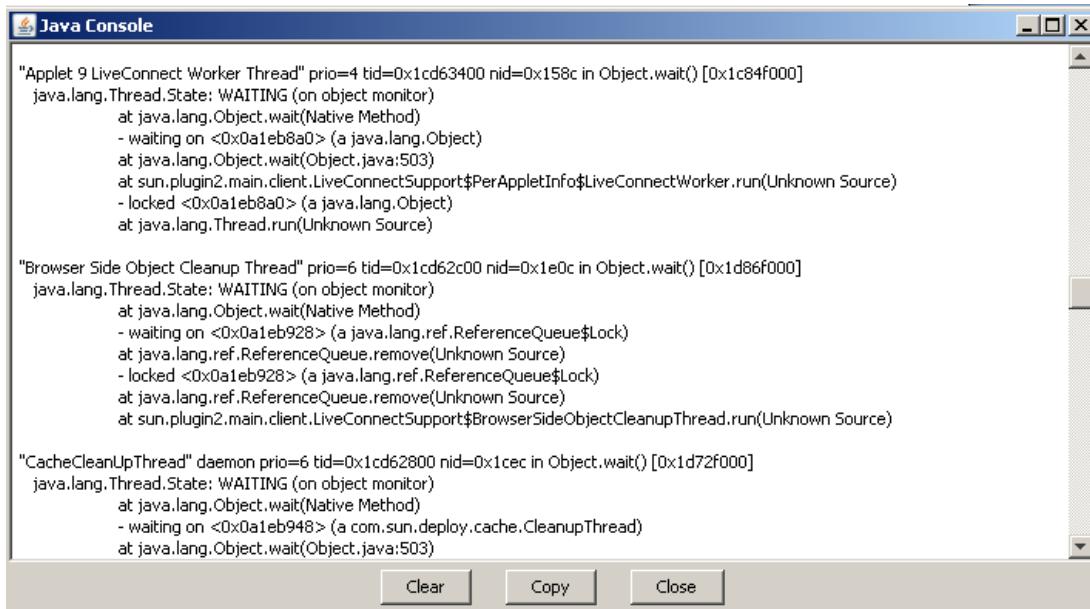
Tabela 5.2: Rezultati ispitivanja

Tokom ispitivanja, uočene su i mane realizacije sprege JavaScript okruženja i Java apleta.

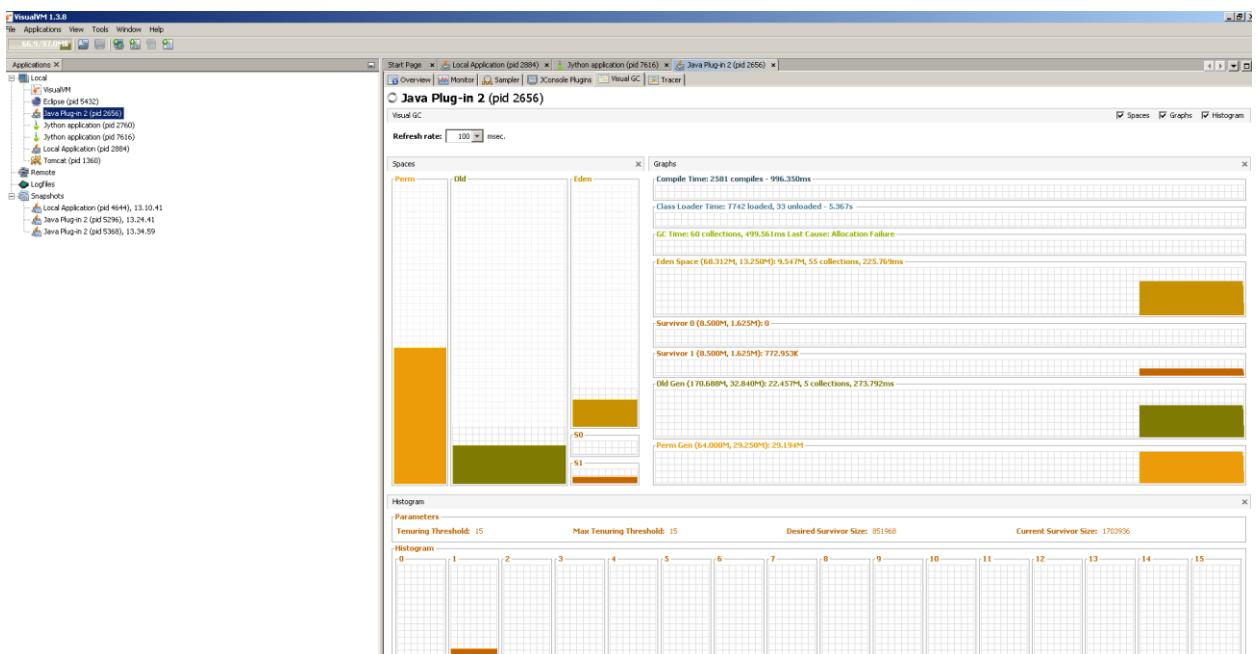
Pravljenje posebne programske niti za obradu zahteva poslatog iz JavaScript okruženja utiče na stabilnost sistema, budući da nije realizovan sistem za kontrolu i ograničavanja broja programskih niti. Konkretno, programske niti apleta, koje pozivaju funkcije iz JavaScript okruženja, se oslanjaju na servisnu programsku nit (eng. daemon thread) LiveConnect API-a.

Servisna nit izvršava poziv ka JavaScript okruženju i vrši interakciju sa internet pregledačem. Ako servisna programska nit bude prekinuta u toku rada ili čak stavljena na čekanje (što se može desiti zbog neefikasnog raspoređivanja vremena usled brojnih niti koje su pokrenuta na Java virtuelnoj mašini), internet pregledač će smatrati da je došlo do greške i da aplet više ne odgovara na zahteve korisnika, i nasilno okonačava rad apleta i Java virtuelne mašine.

Pri ispitivanju, analiza stanja i rada programskih niti je vršena uz pomoć JConsole i VisualVM alata (slike 5.1 i 5.2).



Slika 5.1: JConsole – pregled stanja programskih niti unutar virtuelne mašine



Slika 5.2: VisualVM – alat za vizuelizaciju trenutnog stanja virtuelne mašine

Na osnovu rezultata ispitivanja zaključuje se da se podsistem ispunjava zadatu funkcionalnost i da funkcioniše ispravno u ispitivanim uslovima rada.

6. Zaključak

U ovom radu je realizovan podsistem za ispitivanje mrežnih (eng. web) aplikacija upotrebom Selenium okruženja za ispitivanje. Selenium kao okruženje za ispitivanje, nudi programske rukovaće za upravljanje radom internet pregledača. Podsistem je projektovan kao ugrađena aplikacija (eng. widget), da bi se ugradio u mrežnu (eng. web) aplikaciju za planiranje i rukovođenje ispitivanja na projektima izrade i održavanja programske podrške – RT-Intent.

Iako Selenium okruženje nudi implementacije svog mrežnog rukovaoca (eng. WebDriver) u nekoliko programskih jezika, izabrana je Python implementacija, radi što lakšeg i jednostavnijeg pisanja koda za ispitivanje (odnosno kako bi se pisanje koda za ispitivanje svelo na pisanje skripti).

Za rad podsistema (lansiranje novih procesa - internet pregledača i kontrolu istih) bio je neophodan neograničen pristup računaru korisnika iz samog pregledača, što je postignuto uz pomoć potписаног Java apleta. Jython kao Python interpreter napisan u Javi, omogućio je apletu da izvršava Python skripte za ispitivanje na korisnikovom računaru.

Jedna od značajnijih poteškoća koja je premošćena je bila komunikacija GWT-a i java apleta. Naime, ako se metoda čak i potписанog apleta (koji je prethodno dobio neograničen pristup računaru korisnika) pozove iz JavaScript okruženja, aplet se smatra neovlaštenim i neophodno je da ponovo traži dozvolu korisnika za rad u privilegovanim režimima. Ovo je otvoreno pitanje i teško je prepostaviti budućnost samih potpisanih java apleta kao i njihove sprege sa JavaScript okruženjem.

Inicijalno zamišljena mogućnost izvršavanja na višestrukim platformama nije realizovana zbog poteškoća izvršavanja Jave u internet pregledačima na drugim platformama i generalnog

stava internet pregledača ovih platformi prema Java apletima i Javi uopšte. Tako da podsistem u potpunosti funkcioniše samo na Windows platformi.

Ispitivanje rada podistema je vršeno ispitivanjem rada Intent+ aplikacije, generisanjem koda za ispitivanje uz pomoć Selenium razvojnog okruženja, zadavanjem scenarija za ispitivanje upotrebom Intent+ aplikacije i njihovim automatskim izvršavanjem od strane podistema.

Realizacija podistema je podrazumevala razmenu objekata i drugih informacija između modula napisanih u tri različita programska jezika : Java, Python i JavaScript, što u današnje vreme predstavlja redovnu praksu. Nivoi apstrakcije na kojima se projektuje i realizuje isprojektovano su svakim danom sve viši. Dinamički jezici, skript i domenski specifični jezici kao i generatori koda postaju standardni delovi modernih rešenja.

Dalji rad na podsistemu će obuhvatiti realizaciju kontrolisanog izvršavanja scenarija za ispitivanje (korak po korak, tačke prekida), optimizaciju vremena učitavanja java apleta kao i realizaciju efikasnijeg mehanizma za kontrolu pravljenja i izvršavanja programskih niti deljenjem istih na zadatke (eng. tasks) koje će efikasno izvršavati odgovarajući raspoređivač zadataka (eng. task scheduler).

7. Literatura i reference

- [1] Chromium Project, <http://www.jython.com/archive/21/docs/whatis.html>
- [2] Jython dokumentacija, <https://wiki.python.org/jython/WhyJython>
- [3] Jython Book <http://www.jython.org/jythonbook/en/1.0/>
- [4] Selenium WebDriver API http://docs.seleniumhq.org/docs/03_webdriver.jsp
- [5] Neal Ford: *The Productive Programmer*, 2008
- [6] Kevin Hazzard, Jason Bock: *Metaprogramming in .NET* , 2012
- [7] Oracle: *The Reflection API*, <http://docs.oracle.com/javase/tutorial/reflect/>
- [8] Dennis Sosnoski: *Java programming dynamics*,
<http://www.ibm.com/developerworks/java/library/j-dyn0603/index.html> , 2003
- [9] Fredrik Lundh: *Thread Synchronization Mechanisms in Python*, 2007,
<http://effbot.org/zone/thread-synchronization.htm>
- [10] GWT Doc : Calling a Java Method from Handwritten JavaScript,
<http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsJSNI.html#calling>
- [11] AceEditor, <http://ace.c9.io/#nav=about>
- [12] GWTUpload, <https://code.google.com/p/gwtupload/>
- [13] AceGWT, <https://github.com/daveho/AceGWT#acegwt>

[14] LiveConnect: <https://jdk6.java.net/plugin2/liveconnect/>