



FAKULTET ZA INFORMATIKU I MENADŽMENT

Nenad Šćekić

FLASH DATA COMPUTING

- Diplomski rad -



FAKULTET ZA INFORMATIKU I MENADŽMENT

FLASH DATA COMPUTING

- Diplomski rad -

Mentor:
Dr doc Angelina Njeguš

Student:
Nenad Šćekić

Br. indeksa:
184/2004



FAKULTET ZA INFORMATIKU I MENADŽMENT

UNIVERZITET SINGIDUNUM
FAKULTET ZA INFORMATIKU I MENADŽMENT
Beograd, Danijelova 32

Kandidat: Nenad Šćekić

Broj indeksa: 184/2004

Smer: Projektovanje i programiranje

Tema: Flash Data Computing

Zadatak: Analiza centralizacije procesorske moći kroz postojeću tehnologije, kao i testiranja postojećih tehnologija za rešavanje zadatka koji podrazumevaju veliku količinu računarskih resursa.

.

MENTOR

Dr doc Angelina Njeguš

DEKAN

Prof. dr Milan Milosavljević

Sadržaj

1.1	Computing.....	- 1 -
1.2	Zašto grid computing ?	- 1 -
1.2.1	Time sharing	- 1 -
1.3	Time-sharing kao usluga.	- 3 -
1.4	Computer Utiliti.....	- 3 -
1.5	Util computing se u praksi.....	- 3 -
1.6	Analiza.....	- 4 -
1.6.1	Analiza super računara:.....	- 5 -
1.6.2	Analiza super grid računara:.....	- 5 -
1.7	Super računar – u kom pravcu ići	- 5 -
1.8	Implementacije GRID-a.....	- 6 -
2	Globus Toolkit	- 7 -
2.1	Kako se to odnosi na gt4(Globus toolkit 4)	- 10 -
2.1.1	Kratak opis Veb servisa	- 11 -
2.1.2	Primer upotrebe veb servisa.	- 11 -
2.1.3	Veb servis arhitektura	- 13 -
2.1.4	Veb servis – Serverska strana	- 14 -
2.1.5	WSRF: The Web Services Resource Framework	- 14 -
2.1.6	WSRF: Sve je u stanjima	- 15 -
2.1.7	Pristup resursima sa stanjem	- 16 -
2.2	The Globus Toolkit 4	- 20 -
2.2.1	Arhitektura	- 20 -
3	Flash VM	- 21 -
3.1	Istorijat Flash-a.....	- 21 -
3.2	Primer programiranja u ActionScript-u.....	- 24 -
3.3	Tipovi podataka	- 25 -
3.3.1	ActionScript 2	- 25 -
3.3.2	ActionScript 3	- 26 -
3.4	Rasprostranjenost Flasha.	- 28 -
3.4.1	Dostupnost flash vm	- 28 -
3.4.2	Instalirane verzije flash vm.....	- 29 -

3.4.3	Statistika korišćenja interneta	- 29 -
4	Flash kao grid	- 30 -
4.1	1. Slučajevi korišćenja DataGrid sistema	- 30 -
4.1.1	1.1 Priprema <i>Grid</i> kontrole.....	- 30 -
4.1.2	1.2 Inicijalizacija	- 31 -
4.1.3	1.3 Izvršenje.....	- 32 -
4.2	2. Konceptualni model.....	- 34 -
4.3	3. Sekvencijalni dijagrami.....	- 35 -
4.3.1	3.1 Priprema <i>Grid</i> kontrole.....	- 35 -
4.3.2	3.2 Inicijalizacija	- 35 -
4.3.3	3.3 Izvršenje.....	- 36 -
5	Dalji razvoj	- 37 -

1.1 Computing

Computing, Computation je u opštem smislu bilo koji tip obrade informacija. Uključujući proces u kome se ljudski proces razmišljanja prevodi-pretvora u računanje u užem smislu.

Utility Computing je skup računarskih resursa (procesora, skladišnog prostora, itd) koji je dostupan svima na korišćenje kao što je dostupna voda, struja ili telefonska mreža. Pod pojmom computing grid podrazumeva se kreiranje mreže računarskih resursa koji bi bili dostupni svima na korišćenje. Kreiranje ovog sistema je još uvek paradigma, postoje razni oblici u implementaciji koji postižu određene rezultate, ali svi oni su daleko od konačnog cilja kreiranje kompletnog grid computinga. Sam standard za grida još uvek nije formiran, iako sve implementacije se u nekom obliku oslanjaju na već postojeće standarde.

1.2 Zašto grid computing ?

Ideja o postojanju računara kao sredstva koje bi ubrzalo, kao i omogućilo bilo kakvu obradu podataka je odavno poznata. Vreme koje je potrebno za rešavanje bilo kog problema je presudan faktor. Računar koji bi omogućili kratko vreme rešavanja nekog određenog problema je najčešće preskup. Pojava super računara predstavlja jedno rešenje za probleme koji sadrže veliku količinu podataka. Napomenuo bih da odnos cene usled brzog razvoja računara je u konstantnom padu kao i povećavanje brzine rada. Može se smatrati da je značajan pad cene računara prilikom pojave nove generacije. Naravno da ovo pravilo je još više izraženo kada se uzmu u obzir super računari. Podatak koji najviše iznenađuje je da sama cena jednog super računara posle par godina postaje zanemariva jer troškovi održavanja takvih sistema prevazilaze cenu kupovine. Neretko razvijene zemlje doniraju stariju generaciju svojih super računara drugim zemljama jer održavanje takvih sistema postaje apsolutno neopravdano. Također razvojem tehnologije sve češće se traže rešenja na komplikovanije probleme i obradu veće količine podataka. **Ne napreduju samo računari već količina i obim problema koje je potrebno rešavati.**

1.2.1 Time sharing

Početkom razvoja računarske tehnologije cena jednog računara je bila izuzetno visoka da bi mogao da je priušti prosečan korisnik. 60tih kada je počela ekspanzija računara a cena jednog računara je bila isuviše prevelika da bi bila u vlasništvu jedne osobe. Ako bi smo izuzeli upotrebu u vojsci gde su unapred bili konstruisati računari za obavljanje zadataka specijalne namene. Cena jednog računara je bila previsoka a njegovo korišćenje je bilo posvećeno rešavanju jednog problema. Zapravo korišćenje računara je bilo vrlo ne efikasno jer je opsluživao samo jednog korisnika.

Usled velike ne iskorišćenosti računara primenjen je koncept **time-sharinga** (Deljeno vreme). Na jedan računar bi se prijavio određen broj terminala svi oni bi čekali svoj red na izvršavanje svog zadatka. Pretpostavimo da se na jedan računar prijavi deset terminala. Glavni računar, main-frame, bi redom proveravao da li neki terminal ima neki zadatak i izvršavao ga. Koncept je imao smisla i praktičnu primenu jer je vreme čekanja procesora na konkretan zadatak bilo znatno veliko. Računar bi čekao na unos korisnika, kao i na spore periferije trake, diskove, štampače itd. Daljim razvojem brzine procesora kao i padom cena, računari su postali

znatno pristupačni. Tako da je korisnik bio u mogućnosti da sebi priušti računar solidnih karakteristika i svu njegovu procesorsku moć koja bi bila na raspolaganju samo njemu. Što je u određenom momentu predstavljalo veći napredak jer u primeni jednog glavnog računara i velikom broja terminala ponuđeno vreme jednom terminalu i korisniku bi bilo znatno manje nego kada bi sam korisnik posedovao sam svoj računar.

Promenom ideje o centralizaciji korisnici su osetili boljitak. Ali pojavila se novo polje ne iskorišćenosti. Svi računari u vlasništvu korisnika su također imali veliko vreme ne iskorišćenosti. Što je znatno povećalo ne iskorišćenost ukupne računarske moći svih pojedinačnih računara.

Tokom 70setih pojavio se znatan broj firmi koji su pokušali da kreiraju super računar sastavljen od većeg broja računara i nudili im svoje resurse klijentima. Korisnicima ovih usluga bi se naplaćivalo vreme pozivne linije u vremenu, količina skladišnog prostora u kilobajtima kao i vreme računanja određenog zadatka u sekundama. Iako je način naplate bio dobro osmišljen nije se pokazao previše isplativim jer je brzina razvoja računarske tehnologije činilo da ovi projekti budu apsolutno ne isplativi. Skrenuo bih pažnju da u slučaju korišćenja u vojne i državne svrhe kada novac naravno nije problem, bili su kreirani razni mehanizmi povezivanja jer potreba za izračunavanjem velike količine podataka predstavljala nužnost. Naravno bez obzira na cenu i efikasnost takvih rešenja nakon par godina usled brzog razvoja tehnologija oni bi postajali neisplativo preskupi za održavanje i naravno njihove karakteristike bi postale zastarele. Pošto je informatika u jeku svog napredovanja bez obzira na uloženu količinu novca, svaki sistem bi bio osuđen na neefikasan odnos novca/karakteristika i vrlo brzo bi postao ili zastareo ili preskup za korišćenje.

Napomenjem da 60tih godina prošlog veka nije bio prisutan koncept ličnih(personalnih) računara. Uglavnom računari su bili u vlasništvu vojske, raznih laboratorija ili znatno manjem broju Univerzitetima. Vremenom uočen je obrazac korišćenja računara. Najme računar bi većinu svog vremena čekao na unos korisnika ili pak sam procesor bi čekao odgovor neke od sporih spoljnih komponenti kao što su razne trake, diskovi, štampači itd. Time-sharing se nametnuo kao efikasniji način iskoristiti procesorsko vreme. Time-sharing je omogućio deljenje vremena jednog računara na više korisnika. Na takav način da bi računar redom ispitivao da li je neki korisnik poslao nešto na obradu. Također dok bi sam računar čekao na svoje spore uređaje mogao je da ponudi svoje resurse drugom korisniku.

Kasnih 60tih 70 desetih, kompjuterski terminali su bili multipleksirani u velike institucionalne main frame računare. Koji su sekvencionalno proveravali da li terminal ima neki zahtev za obradu. Kasnije su interkonekcije podržavale interapte, a neke od njih su podržavale paralelno prenos podataka kao na primer IEEE 488 standard koj dozvolja da 15 uređaja dele jednu 8-bitnu paralelnu vezu. Tako da su vremenom na Univerzitetima koristili terminali znatno potsećali na korišćenje današnjih ličnih računara.

Povećavanjem procesorske moći ranih 80, koncept time-sharinga je zapostavljen jer je cena procesora drastično smanjila tako da je računar postao pristupačniji većini korisnika. Prednost ličnog računara je očigledna jer bi se sva procesorska moć (naravno znatno slabija u odnosu na jedan time shering računar) bila predana jednom korisniku njegovim potrebama. Sve do pojave interneta time-sharing je retko korišćen i može se reći da je bio zaboravljen kao koncept. Treba napomenuti da je internet ponovo vratio koncept time-sharinga. Skupi korporativne farme servera koji koštaju milione mogu da opslužuju hiljade klijenata i da svi dele zajednički resurs. Kao i ranije serveri funkcionišu u periodničnom trendu nakon dugog perioda ne aktivnost dolazi period povećane aktivnosti prilikom koje nijedan od korisnika ne primećuje zakašnjenje. Naravno u slučaju prevelikog

broja posetilaca kao i zahteva server može postati previše zauzet.

1.3 Time-sharing kao usluga.

Ranih 60tih nekolicina kompanija je ponudila time-sharing uslugu kao servis za poslovanje. Korisnici be se povezivali telefonskom linijom dial-up-om. Terminali su podržavali od 30 do 120 karaktera u sekundi. Korisnicima su bili ponuđeni terminali, razni alati, prostorno skladištenje, štampanje itd. Korisnicima bi se naplaćivalo vreme zauzeća linije, procesorsko vreme u sekundama, i po kilobajtu skladištenog prostora. IBM 360, PDP-10 I SDS 940 su najčešći sistemi koji su bili korišćeni za tu svrhu.

1.4 Computer Utiliti

70tih se dosta vremena ulagalo u centralizaciju računarskih resursa. Kreiranje raznih usluga koje bi omogućile iskorišćavanje resursa kao opšte dobro. Međutim taj koncept se nije mogao održati usled prevelike brzine tehnološkog razvoja i cena održavanja takvih sistema. Tržište za util computing nije postojalo. Sve do pomeranja ka klijent-server centralizaciji i kreiranjem server farmi.

Internet je ponovo kreirao tržište za util computing. Util computing je oblik virtualizacije u kome količina prostora i procesorske moći može da nadmašiti jedan time-sharing računar ili super računar. Više servera u pozadini grupiše svoju moć i kreira zajedničke resurse. U praksi to mogao da bude dedukovani klaster posebno kreiran za određenu namenu kojoj bi bio iznajmljen ili predstavljao određenu verziju super računara za namenski zadatak. Tehnika pokretanja jednog zadatka na više računara je poznata kao **distribuirano računanje**. A pod termin Grid computing se podrazumeva određen oblik distribuiranog računanja. Gde su činioci (*nodes*) geografsko udaljeni ili u više različitih domena. Da bi obezbedio util computing kompanija može da ponudi svoje resurse javnosti na prodaju.

Jedan od najčešćih modela je volunteer computing aplikacije je centralni server koji daje zadatke činiocima (nodovima). Klijenti bi na dobrovoljnoj bazi instalirali potreban softver koji bi omogućio kreiranje grid-a.

Drugi model bi bila firma koja bi prodavala svoje resurse kada oni nebi bili korišćeni.

1.5 Util computing se u praksi

Najčešće se koristi za specijalne namene. Ovakav sistem ima znatno nižu početnu cenu ili kroz volunteer computing može biti dostupan bez ikakve nadoknade. Zapravo računarski resursi su najčešće iznajmljeni. Korisnici koji imaju potrebu za velim računarskim resursima ili im se takve potrebe javljaju u određenim intervalima korišćenjem util computinga mogu da izbegnu probleme koji bi nastali prilikom fizičkog nabavljanja i sastavljanja velikog broja računara.

Sajt berze bi bio adekvatan primer. Zato će mo razmotriti problem koj bi se javljao. Nakon otvaranja berze u 10 časova posetioci sajta su u najvećoj meri brokeri koji gledaju trenutno stanje akcija. Sve do zatvaranja berze u 14 časova količina poseta se nebi znatno

menjala i konstantno bi imala poprilično isto opterećenje, tako da u tom vremenskom intervalu možemo smatrati da broj posetilaca sajtu predstavlja normativnu veličinu. Nakon zatvaranja berze. Serveri berze izvršavaju određene proračune kao i javno objavljivanje vrednosti akcija na kraju tog radnog dana berze. Broj posetilaca se naglo povećava nakon zatvaranja berze. Znatno veći broj posetilaca želi da vidi kako su njihove akcije prošle tog dana kao i brokeri koji bi sumirali taj dan. Međutim to je najkritičniji momenat, jer to je momenat kada serveri berze trebaju da u što kraćem vremenskom intervalu generišu statistiku analizu i zatvore i provere sve transakcije.

Prilikom gradjenja infrastrukture mogu se opredeliti za kupovinu velikog broja računara koji bi omogućili nesmetan rad prilikom najveće posete. Cena i troškovi bi bili ogromni a potreba klijenta

1.6 Analiza

Postoje dva načina kreiranja super-računara:

- Posebno konstruisanje računara na harverskom nivou.
- Kreiranje super računara softverskim putem. Udruživanje više različitih računara u jednu logičku celinu.

Glavna razlika između jednog super računara i grid super računara se sagleda u tome da jedan super računar ima jedno napajanje jednu grafičku kartu dok grid super računar je sastavljen od mnoštvo kompletnih računara.

Posmatrano sa aspekta ideje kreiranja super računara od velikog broja računara. Grid computing još uvek nije doživeo svoju pravu ekspanziju. Standard za Grid computing još uvek nije kreiran iako se konkretne implementacije koriste određenim standardima prilikom kreiranja. Nedostaci postojećih rešenja su:

1.6.1 Analiza super računara:

Super računar	Prednost	Mana
Brzina	✓	
Cena		✓
Pouzdanost	✓	
Troškovi nabavke		✓
Troškovi održavanja		✓
Vek trajanja		✓
Potrošnja električne energije		✓
Heterogenost hardvera	✓	

1.6.2 Analiza super grid računara:

Super računar	Prednost	Mana
Brzina		✓
Cena	✓	
Pouzdanost		✓
Troškovi nabavke	✓	
Troškovi održavanja	✓	
Vek trajanja	✓	
Potrošnja električne energije	✓	
Heterogenost hardvera		✓

Glavni nedostaci grid super računara su komplikovana instalacija, prevelika heterogenost. Kao i opšta ne obaveštenost krajnjih korisnika koji bi trebalo da samo inicijativno postanu volonteri. Naravno u slučaju kada se unutar neke organizacije kreira grid i dalje ostaje problem komplikovane instalacije. Mali broj nodova

Glavne prednosti su da bilo koj hardver bi mogao da bude jedan od potencijalnih činioca koji bi bili uključeni u grid. Naravno usled prevelike heterogenosti to može da predstavlja i nedostatak. Zato se ili kreira namenski operativni sistem ili koristi Java kao platformo nezavisna rešenje.

1.7 Super računar – u kom pravcu ići

Pod pretpostavkom da postoji oblik udruživanja procesorskih moći unutar jednog super računara ili pak softversko kreiranje super računara korišćenjem grid-a glavni probleme kao i ne doslednost su da dosadašnja rešenja ne koriste potpunu moć svih računara.

Ideja koja se nameće je kreiranje jedinstvene platforme koj bi omogućila korišćenje svih ne iskorišćenih procesorskih ciklusa ili ukupne procesorske moći. Naravno u praksi to postaje paradigma zbog velike različitosti počevši od samih računara, operativnih sistema i načina njihove povezanosti ili ne povezanosti. Paradigma je skupiti ukupnu procesorsku moć i nju ponuditi kao opšte dobro.

Bez obzira na koje se rešenje koristi. Jedan super računar uvek po svojim performansama brži od bilo kog ličnog računara kao i od dosadašnjih implementacija bilo kog grid-a. Razlog je sasvim jednostavan. Ni jedna zajednica grida nije udružila veliki broj svojih činioca(nodova). Prosečne cifre o kojima govorimo su reda veličine od 100-10 000 što čini manje od promila ukupne procesorske moći personalnih računara.

U budućnosti sve ređe ćemo viđati super računare kao jedan brz procesora. Poslednjih godina već uveliko se uvode računari sa više procesora. Mehanizmi kao softverska rešenja na nivou operativnog sistema ili pak softvera koji će spajati više procesora u jedan samo su prvi korak ka pravoj implementaciji grid-a. Jedno je sigurno, budućnost donosi operativne sisteme koji će umeti da koriste više jezgara, kao i alate programerima da olakša i omogući i popularizuje paralelno programiranje. Tek nakon toga će postojati dobra platforma i alati koji će omogućiti da se kreira i standardizuje grid. Alati kao i operativni sistemi koji podržavaju paralelno programiranje odavno postoje ali su daleko ne popularni. Upravo iz razloga što se izvršavaju ili na posebnim hardverima ili u retko korišćenim operativnim sistemima.

Mehanizmi, modeli, alati za korišćenje distribuiranog računanja i paralelnog programiranja su već osmišljeni, ali najveći problem su ne kompatibilnost sa personalnim računarima. Zahtevaju posebne operativne sisteme, posebne programske jezike(alate).

Da li postoji neki način i da li postoji neki mehanizam koji bi omogućio pokretanje i izvršavanje zadatka na udaljenim računarima, a da pritom ne zahteva posebnu aktivnost korisnika, je pitanje koje se nameće. Sa druge strane garantuje da daj zadatak nije pokrenut od strane neke zlonamernog korisnika koja bi mogla ili ka iskoristi tu moć u svoju svrhu ili pak da sazna privatne informacije vlasnika računara.

1.8 Implementacije GRID-a

Microsoft nudi svoju verziju grid hpc-a. koji zahteva da su računari pokrenuti korišćenjem Windows operativnog sistema kao i uređivanje infrastrukture kao što je instaliranje Aktivnog direktorijuma, Domena kontrolera. Domen kontroler i sama zahtevi ne predstavljaju problem u lokalnom korišćenju ali je nemoguće iskoristiti microsoftovo rešenje za globalnu primenu. Napominjem da ovaj problem nije samo u Microsoft-ovoj implementaciji već predstavlja globalni nedostatak.

Da bi se kreirao grid neophodno je kreirati virtualnu mašinu koja bi bila dostupna na svim operativnim sistemima. kao i u organizacijama kao i na desktop računarima. Na ono našta bih skrenu pažnju a o čemu će se više govoriti je korišćenje FLASH VM koja je dostupna na preko 99% računara u svetu. što predstavlja naj prisutniju aplikaciju-virtualnu mašinu u svetu.

Ne treba zaboraviti GPL rešenja kao što su Beowulf ili Rock i Globus koji ćemo obraditi kako radi.

Bez obzira na tip ili verziju mana svih dosadašnjih sistema je komplikovano korišćenje, zahteva pristanak korisnika, znanje za kreiranje i instalaciju. Naravno teoretski virtualizacija može da pomogne u praktičnom delu razmeštanja nodov. Naime, ono što bi virtualizacija praktično olakšala bi bilo prebacivanje virtualnih mašina unutar grida ili

prestavljala mehanizam za daljinsko prebacivanje virtualnih grid mašina na udaljene računare, gde bi sama virtulna mašina predstavljala platformu kao virtualni host-platforma. Ono što ovaj koncept čini lošim je zahtevanje posebne aktivnosti od korisnika da pristane da pokrene virtualnu mašinu. Naravno ova ideja samo olakšava pristupačnost i portabilnost ali opet ne nudi opštost grida.

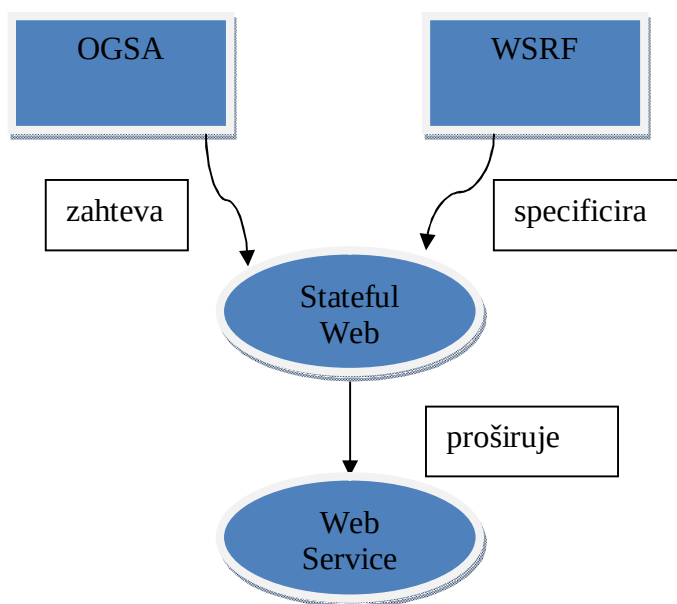
Pretpostavljam da ćemo u budućnosti videti dva načina distribucije grid:

- Grid će biti dostupan kao servis koji će biti instaliran u samom operativnom sistemu i zahtevaće od korisnika da pristane na njegovo korišćenje. Ne treba da nas začudi postojanje verzija windows ili osx operativnih sistema koji bi bili ponuđeni korisniku apsolutno besplatno, a za uzvrat bi automacki nakon podizanju operativnog sistema pokretali servis za grid.
- Sam Grid-nod će biti instaliran na virtualnoj mašini. Što bi od korisnika zahtevalo preuzimanje jednog fajla koji kojim bi pokretali u virtualnoj mašini.

2 Globus Toolkit

Globus toolkit je skup alata koji omogućavaju računaru da postane deo grid-a. O sam nije grid niti je konačni standard. Već kao i ostala rešenja predstavlja određen oblik implementacije paradigme grida.

Osnovni koncepti Globus toolkita. u OGSA, WSRF i GT4



Firura 1 Odnos između OGSA, WSRF i Web Servisa.

Grid aplikacija se obično sastoji od više različitih komponenti. Na primer tipična grid aplikacija bi imala:

- VO Management Service: Koja bi nodove i korisnike smeštala u neku od virtualnih organizacija.
- Resource Discovery and Management Service (Mehanizam za otkrivanje resursa i upravljanje servisima): Da bi aplikacija koja se izvršava na gridu mogla da otkrije resurse koji odgovaraju njenim potrebama, i upravljala njenim potrebama..

- Job Management Service (Servis za upravljanje poslovima) : Da bi korisnici mogli da pošalju zadatak na grid.
- I gomilu drugih servisa koji bi se brinuli o sigurnosti , upravljanju podacima, itd..

Svi ovi servisi međusobno komuniciraju. Na primer, Job Management Service može da konsultuje Resource Discovery Service da bi pronašao već izračunat resurs koji bi odgovarao zahtevima zadatog posla. Sa velikim brojem servisa i čestom komunikacijom postoji velika verovatnoća stvaranja haosa. Pitanje koje se postavlja je šta bi se dogodilo kada bi svaki proizvođač radio sopstvenu implementaciju svog sistema za upravljanje poslovima na totalno drugačiji način. Kada bi svako prijavljivao ne samo drugačiju funkcionalnost već i drugačije interfejsse. Praktično bi bilo nemoguće omogućiti rad svih delova da rade zajedno.

Rešenje koje se nameće je standardizacija tj definisanje osnovnih interfejsa za svaki servis. Na primer, Uzevši na primer Svetsku globalnu mrežu. Jedan od razloga zašto je Internet tako popularan je i što se zasniva na standardima zasnovanim na (HTML, HTTP, itd) dogovorenih od strane svih glavnih učesnika (Microsoft, Netscape, itd). U slučaju da je svaka kompanija rešila da uradi svoju implementaciju. Svako ko bi želeo da koristi Mikrosoftovo rešenje morao bi da očekuje da je tvorac sajta napravio posebnu verziju za taj pretraživač. Međutim usled dogovorenog standarda apsolutno je nebitno koji alat je korišćen za kreiranje internet stranice kao što je nebitno sa kojim internet pretraživačem gledate sadržaj. Dakle html je dogovoren standard kao “zajednički jezik” za sve pretraživale i sajtove. **Standardizacija je jedini oblik koji omogućava širu upotrebu.**

Open Grid Services Architecture (**OGSA**), razvijen od strane Global Grid Forum, teži ka definisanju standarda otvorene arhitekture za grid zasnovana rešenja. Cilj OGSA je da standardizuje sve servise na jednu grid aplikaciju (job management services, resource management services, security services, itd.) garantujući niz standardnih interfejsa za servise ovog tipa. Standard je još uvek u izradi. Ipak OGSA već ima spisak zahteva koji moraju biti ispunjeni za standardne interfejsse. Drugim rečima OGSA je otišla samo daleko u definisanju najvažnijih servisa koji su neophodni za kreiranje i pokretanje grid aplikacije. **Dakle OGSA zahteva “stateful” servise.**

Ipak , Potreba za posrednikom pri kreiranju i prebacivanju zadataka ukazuje da mora da postoji neki standardi način komunikacije. Na primer kada bi OGSA (na primer) definisao interfejs za slanje posla JobSubmitInterface i metodu submitJob. Mora da postoji neki standardni način za pokretanje metode ako bi smo želeli da arhitektura dosegne industriski nivo. Tako da su mogli da budu odabrani razni načini korišćenjem(CORBA, RMI, ili čak tradicionalni RPCa). Iz razloga koji će biti objašnjeni kasnije odabran je **web servis kao noseća tehnologija.**

Veb servis arhitektura je bila definitivno najbolje rešenje izuzev činjenice da je bilo neophodno kreirati takav veb servis koji bi bio stateful. Nažalost ne postoji ni jedan standardan način da se to postigne pa je nešto moralo posebno da se uradi na tom problem.

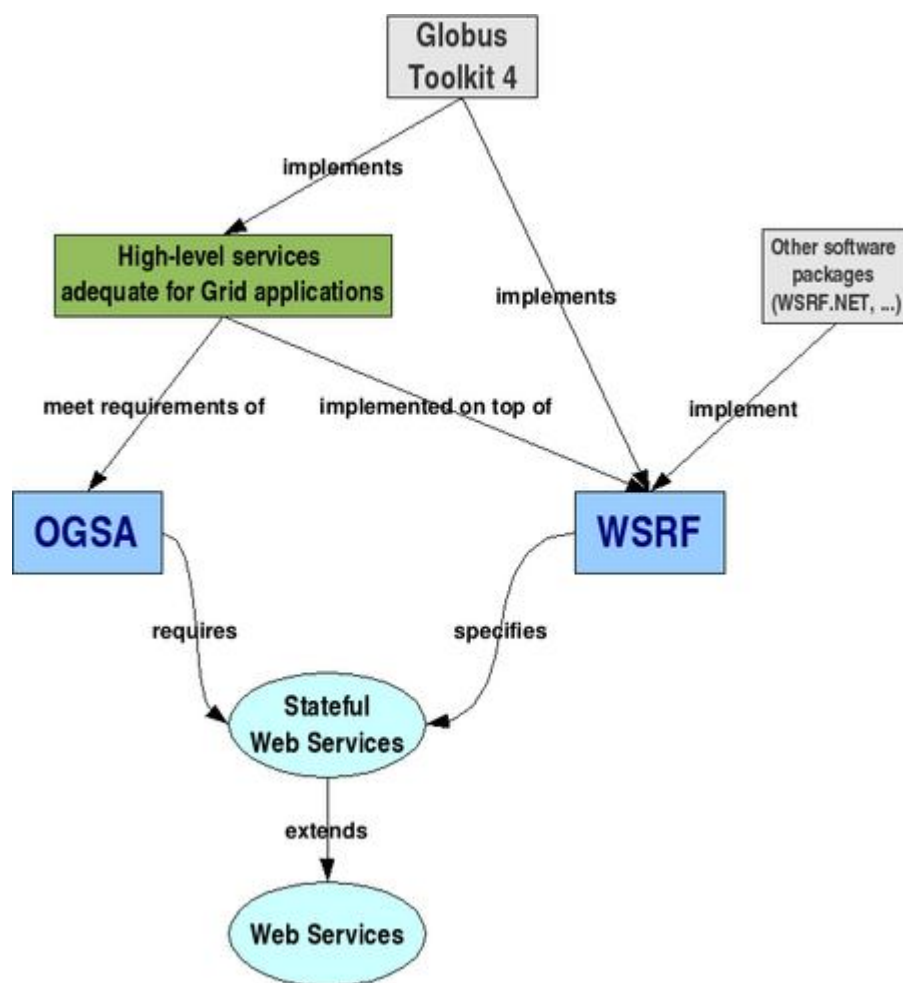
Web Services Resource Framework

Ceo Web Services Resource Framework (WSRF), je specifikacija od strane OASISa. WSRF specificira kako kreirati veb servis da bude stateful, naravno dodajući mnoštvo korisnih stvari. Odnos između OGSA i WSRFa je jednostavan WSRF nudi stateful servise

koje OGSA treba ili OGSA je arhitektura, **WSRF je infrastruktura na kojoj je arhitektura napravljena.**

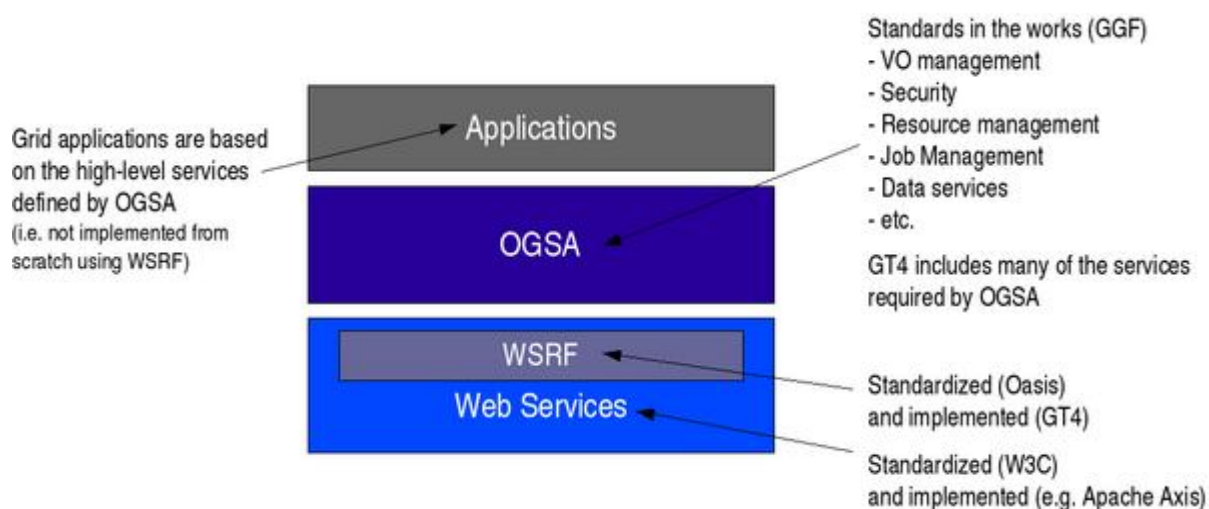
2.1

Kako se to odnosi na gt4(Globus toolkit 4)



Firura 2 Odnos između OGSA, GT4, WSRF i Web Servisa u [3]

Globus toolkit je skup alata, razvijenih od strane Globus alijanse, koju možemo da koristimo u programiranju grid-zasnovanih aplikacija. Sam toolkit sadrži par servisa višeg nivoa koji se mogu koristiti u kreiranju grid aplikacije. Zapravo ovi servisi nude ispunjenje svih apstraktnih zahteva koje je propisala OGSA. Drugim rečima Globu toolkit predstavlja set alata koji pružaju otkrivanje, traženje resursa, infrastrukturu za slanje poslova, bezbednost itd. Iako GGF još uvek radi na specifikaciji standarda, ne može se reći da je GT4 ispunio sve zahteve ili ih pak je ispunjava. Većina ovih servisa je implementirano na gornjem sloju WSRF-a. Sam toolkit implementira neke servise koji nisu na WSRF-u i oni se nazivaju "non-WS components". Za sada sam GT4 implementira sve standarde WSRF-a. To je vrlo važno jer sve ostalo bi trebalo da bude izvršavano na višim slojevima. Tako da dokle god ne bude bio kreirana potpuna "Grid Nirvana", WSRF je samo ne ophodan korak. Vrlo je teško tvrditi u kom pravcu će dalje ići razvoj, jedno je sigurno WSRF je samo prvi korak. Treba napomenuti da WSRF nije jedina implementacija postoje i druge kao što je WSRF.NET.



Firura 3 Odnos izmedju OGSA, WSRF i GT4

2.1.1

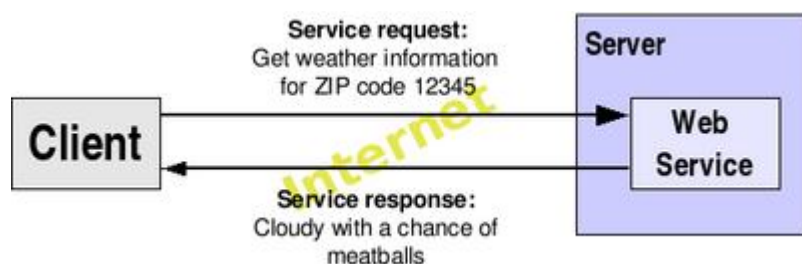
Kratak opis Veb servisa

Web servisi su još jedan oblik tehnologije distribuiranog računanja. Oni nam dozvoljavaju da kreiramo klijent/server aplikacije. Iako veb servisi se oslanjaju na postojeću tehnologiju kao HTTP oni se ne odnose na internet pretraživače i HTML. Naj generalnija podela bi glasila **veb (HTML) je sajt za ljude a veb servis (SOAP) za softver.**

2.1.2

Primer upotrebe veb servisa.

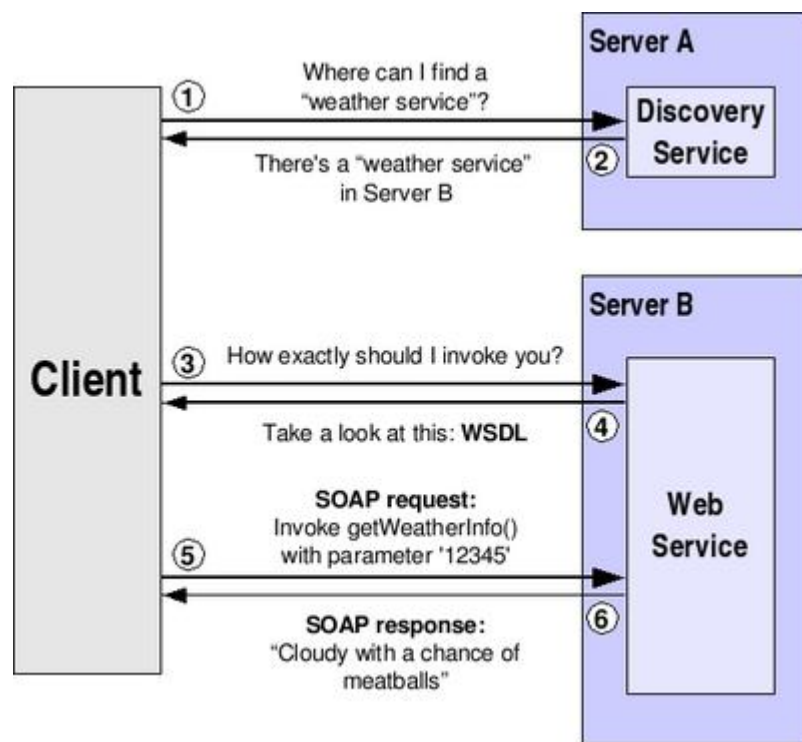
Klijentski program koj bi hteo da pristupi bilo kojoj informaciji prvo mora da kontaktira veb servis, i postavi mu upit o vremenoj prognozi. Server vraća vremensku prognozu kao odgovor kroz veb servis.



Veb servisi nude određene prednosti nad ostalim tehnologijama (RMI, CORBA, EJBs. itd) tako što je platformski nezavistan jezik, pošto koristi standardni XML jezik. Ovo znači da svaki klijent pisan u c++ može da koristi veb servis u javi koja je pokrenuta na linux-u. Mane veb servisa su:

- Brzina prenosa podataka je znatno manja s obzirom da se ne prenose binarni podaci. Ono što se dobije na portabilnosti izgubi se na efikasnosti.

Sami veb servisi ne dozvoljavaju prevlik broj opcija kao što na primer CORBA poseduje (perzistentnost, transport itd. Naravno u skorije vreme veb servis specifikacija i implementacija kao na primer WSRF uvodi neke od potrebnih funkcionalnosti. Prednosti u brzini rada COBRA ili EJB-a su značajne ali računati u svojoj komunikaciji moraju da poseduju prethodno međusobno znanje. Tako da primena ovakve komunikacije preko interneta ne bi postigla značajniji uspeh jer zahteva određene pripreme. Sa druge strane korišćenjem veb servisa klijent ne mora da zna kako će izgledati sama komunikacija dok je ne zatraži. Dakle u internoj komunikaciji veb servisi nisu idealno rešenje, dok u komunikaciji preko interneta oni predstavljaju praktičnu neophodnost.

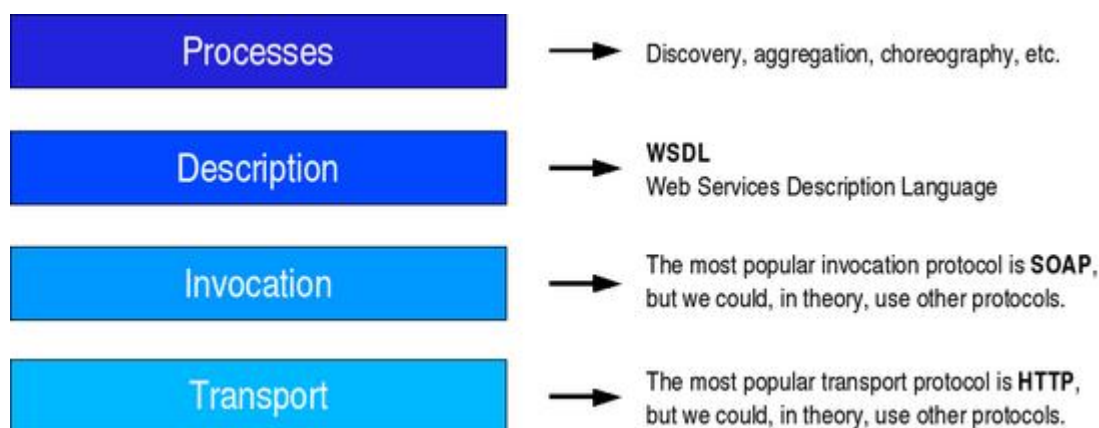


Firura 4 Tipican rad jednog veb servisa

1. Kao što je ranije napomenuto mi nemamo pretstavu o veb servisu koji ćemo pozvati. Tako da u prvom koraku otkrivamo veb servis koji odgovara našim potrebama. Recimo da želimo da komuniciramo sa servisom koji bi nam dao informaciju o vremenskoj prognozi u Srbiji. To će mo postići kontaktiranje servisa za otkrivanje(discovery service) koji je i sam veb servis.
2. Servis za otkivanje nam vraća odgovor, sa obaveštenjem koji serveri mogu da nam obezbede željene servise.
3. Sada kada znamo lokaciju, mi ne znamo sta je sve potrebno da bi smo ga iskoristili. Naravno da znamo da je u pitanju servis za vremensku prognozu, ali nemamo pretstavu kako se zove metoda i kog tipa bi bili parametri. Zato ćemo kontaktirati veb servis koji će ga opisati(WSDL).
4. Veb servis odgovara jezikom nazvanim WSDL.

5. Sada konačno znamo gde je lociran i kako treba pozvati servis. Sam poziv je izvršen korišćenjem jezika SOAP. Šaljemo SOAP zahtev veb servisu da želimo vremensku prognozu za određen grad.
6. Veb servis će odgovoriti SOAP porukom i poslati nam vremensku prognozu ili poslati grešku ukoliko je SOAP zahtev bio ne pravilan.

2.1.3 Veb servis arhitektura

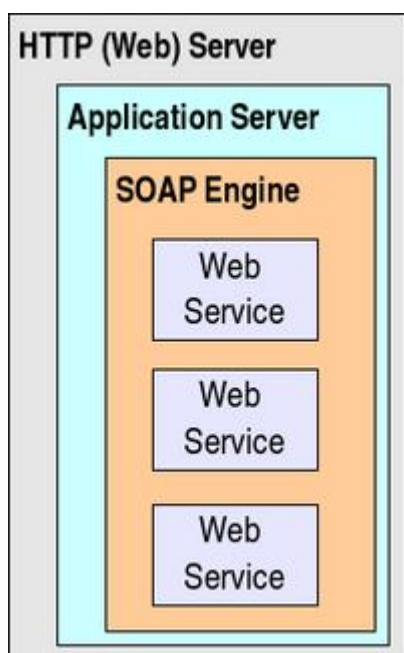


Firura 5 Veb servis arhitektura.

- Service Processes: Prvi deo arhitekture generalno uključuje više od jednog veb servisa. Na primer otkrivanje pripada jednom delu arhitekture, pošto nam omogućava da lociramo određeni servis od kolekcije veb servisa.
- Service Description: Jedna od najzanimljivijih zadataka je da oni su samo opisni, Svaki servi koj ste lozirati možete od njega zatražiti da opiše samog sebe i ukaže koje operacije on podržava i kako ga možete pozvati. Ovaj mehanizam je obezbedjen jezikom za opisivanje veb servisa(Web Services Description Language - WSDL).
- Service Invocation: Pozivanje servisa generalno bilo koj oblik distribuiranog servisa kao što su COBRA ili EJB (Enterprise Java Bean) uključuje slanje poruka izmedju klijenta i servera i kako bi server treba da formatira svoj odgovor. U teoriji možemo da isporistimo bilo koj način za pozivanje veb servisa na primer XML-RPC ili XML. Ipak SOAP je najčešći izbor prilikom kreiranja web-servisa.
- Transport: Na posetku sve ove poruke moraju na neki način putovati izmedju servera i klijenta najčešći izbor za komunikaciju je HTTP (HyperText Transfer Protocol), isti protokol koj se koristi za tradicionalni veb. U teoriji mi možemo da koristimo bilo koj ali u praksi i zbog rasprostranjenosti on je izbor.
- Veći deo veb servis arhitekture je standardizovan od strane wwvc iste organizacije koja je odgovorna za XML, HTML, CSS, itd.

2.1.4

Veb servis – Serverska strana



Firura 6 Serverska strana

- Web service: Veb servis je deo softvera koj pruža niz operacija. U slučaju JAVA programiranja veb servis bi bio java klasa a operacije bi bile implementirane kao JAVA metode. međutim sam veb servis ne zna ništa o SOAP zahtevima i odgovorima i zato nam treba.
- SOAP mašina: koja je deo softvera koji zna kako da postupa sa SOAP zahtevima i odgovorima.
- Application server: Softvera koji omogućava prostor kome će pristupati većina klijenata. SOAP mašina radi kao aplikacija na aplikativnom serveru.
- Većina aplikacionih servera nudi HTTP funkcionalnost. Tako da smo u mogućnosti da koristimo servis samom instalacijom SOAP mašine. Naravno ukoliko to nije slučaj uvek možemo da... koristimo
- HTTP Server: Najčešće se naziva veb server (Apache, IIS).

2.1.5

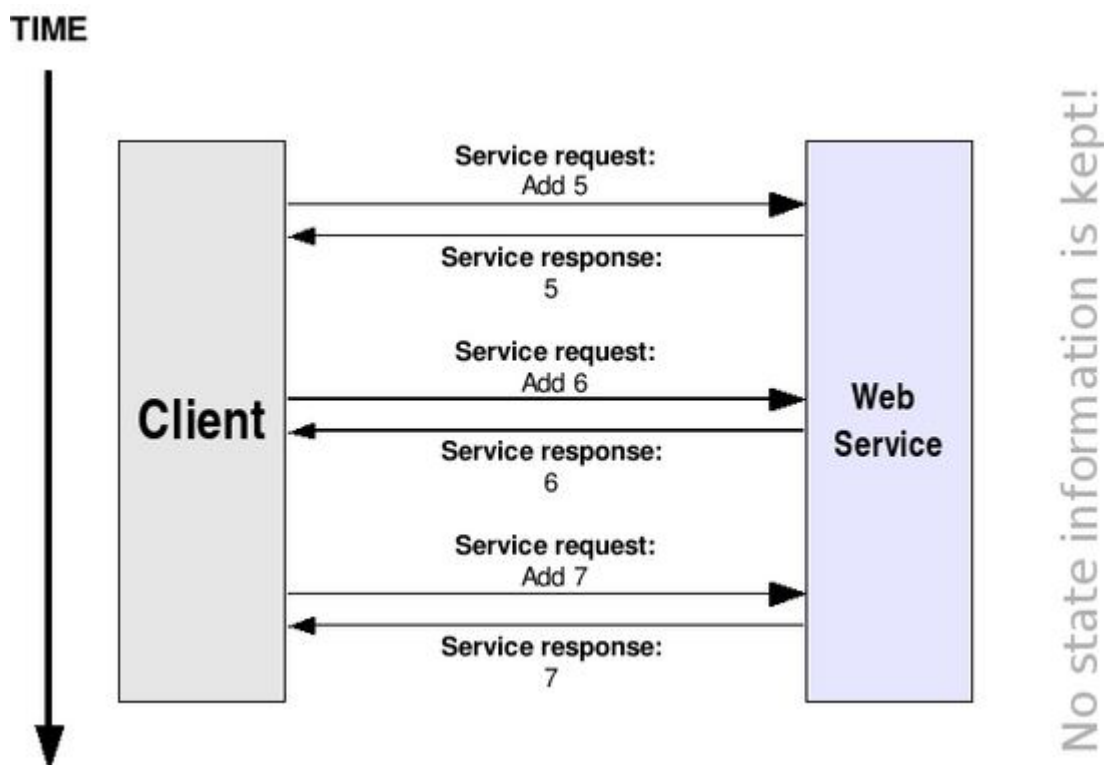
WSRF: The Web Services Resource Framework

Veb servisi su tehnologija koja omogućava kreiranje grid zasnovanih rešenja. Po svojoj rasprostranjenosti po internetu i lakoći pristupa oni su naj pristupačniji metod za distribuciju informacija. Međutim po svojoj specifikaciji oni poseduju dosta ograničenja koja predstavljaju veliki problem za kreiranje grid aplikacija. Pojavom WSRF-a koji poboljšava određene aspekte veb servisa i omogućava ih da budu prikladniji za grid aplikacije.

2.1.6

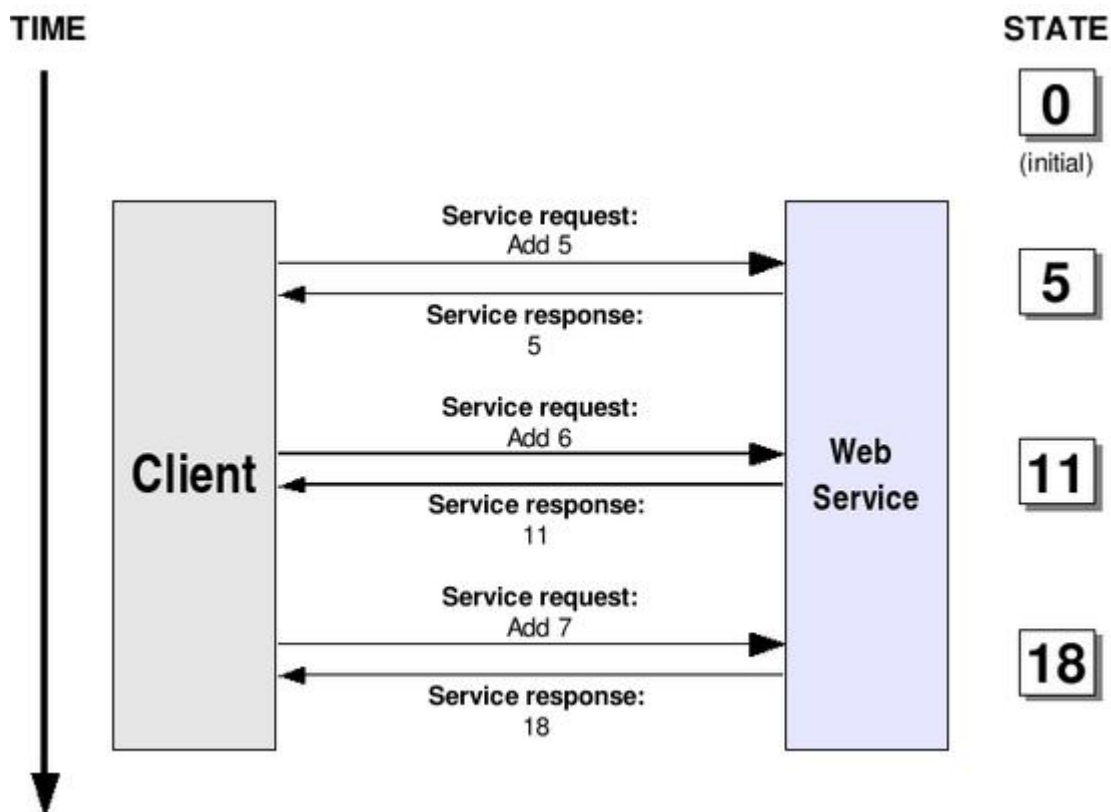
WSRF: Sve je u stanjima

Običan veb servis je bez stanja. To zapravo znači da sam ne može da zapamti informaciju, ili zadrži stanje između dva poziva. Na primer recimo da želimo da saberemo dva broja akumulativno. Prvo ćemo poslati broj pet. Veb servis će vratiti 5. U sledećem pozivu šaljemo broj 6 i očekivan rezultat bi bio 11 ali pošto veb servis nema predstavu o prethodnom pozivu on nam vraća 6. Isto se i dešava i sa trećim pozivom prilikom slanja broja 7. Ono što bi trebalo da bude ukupan akumulativni rezultat 18, od strane veb servisa dobijamo 7.



Firura 7 Pristup resursima bez stanja

Postoji veliki broj slučajeva i aplikacija u kojima servis ne mora da zna stanja. U slučaju korišćenja **u gridu stanja su neophodna**. Tako da idejno naš veb servis bi zadržavao stanja i radio na sl način.



Firura 8 Pristup resursima sa stanjem

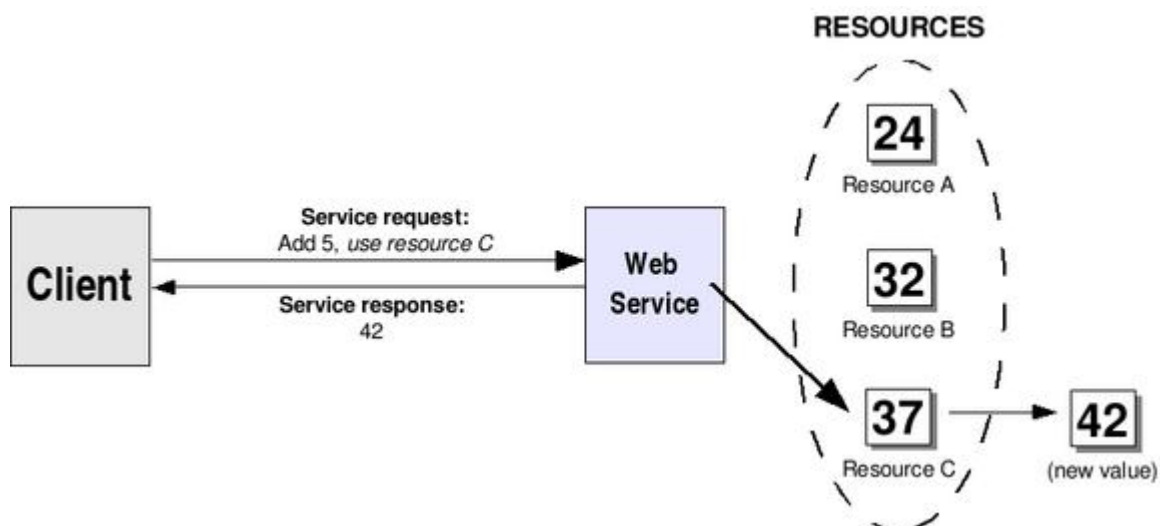
2.1.7

Pristup resursima sa stanjem

Dodajući veb servisu mogućnost da čuva informacije o stanjima je zapravo kompleksan problem. Rešenja za ovaj problem može da se zaobiće tako što bi se veb servis i informacije o stanjima čuvala odvojeno.

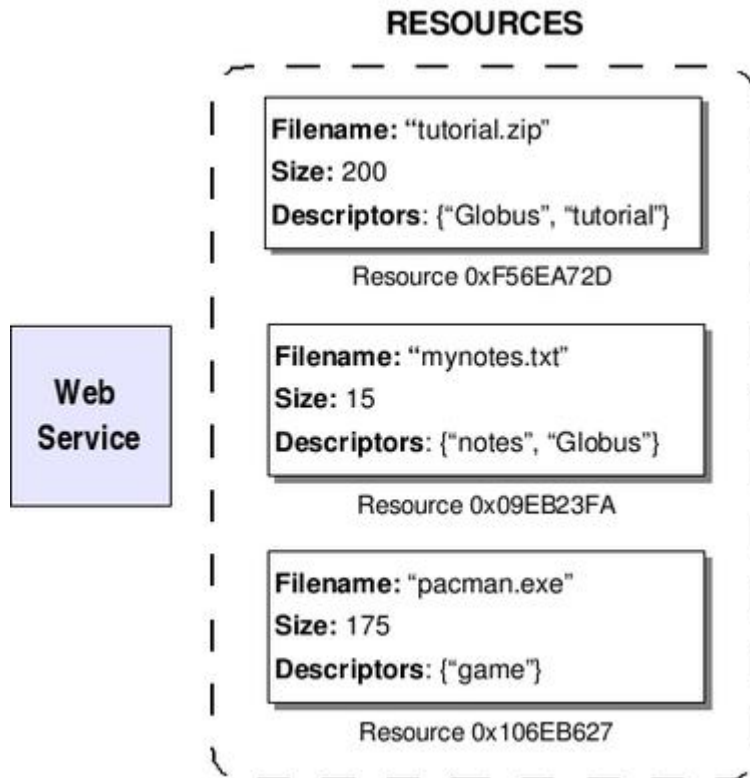
Umesto što bi smo stanja stavili u sam servis(čime bi smo dobili stateful servis, što je zapravo loša stvar.) mi će mo čuvati u odvojenom entitetu nazvano resurs. Koj će zapravo čuvati informacije o stanju. Svaki resurs će imati jedinstveni ključ, tako da kada god poželimo informaciju o stanju mi će mo reći veb servisu da koristi određeni resurs.

Na primer, uzmimo akumulativni primer. Kao što je u narednoj figuri, naš veb servis može da ima tri resursa(A,B,C) koje možemo da biramo. Ukoliko imamo intidžer koj želimo da pamtimo izmedju poziva i poziva, klijent jednostavno treba da specificira šta želi i metod za željeni resurs.



U figuri možemo da vidimo da klijent želi da uradi operaciju sabiranja sa resursom C. Kada veb servis primi zahtev za dodavanje, postaraće se da prikupi resurs C tako da sabiranje može da se desi. Sam resurs može da bude snimljen u memoriju, neki eksterni prostor, ili čak bazu. Također treba primetiti kako sam veb servis može da pristupi ka više od jednog resursa.

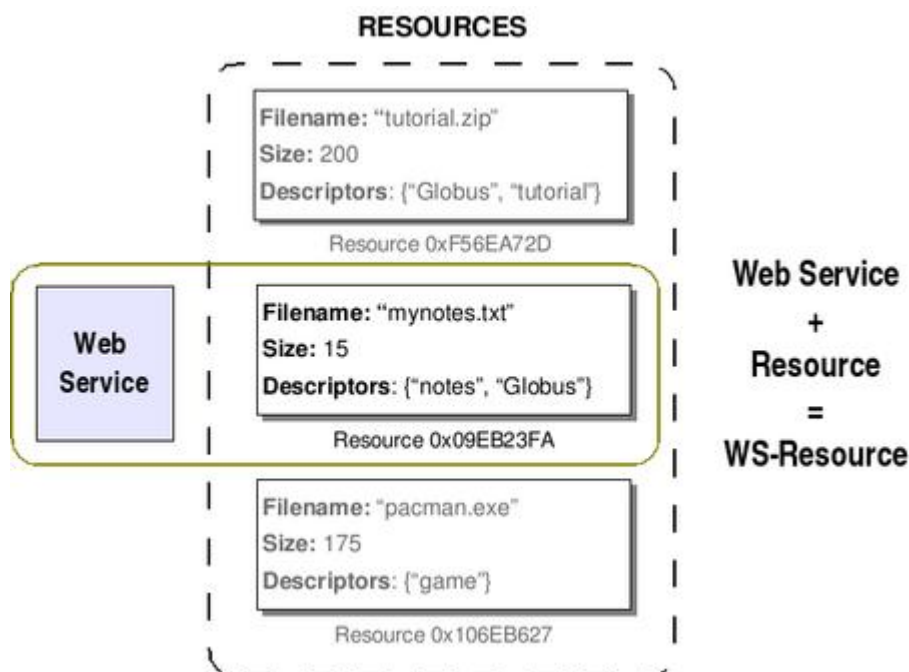
Naravno, resursi mogu da budu različitih oblika i veličina. Sam resur može da sadži više različitih vrednosti(ne samo intidžer kao što je pokazano u primeru). Naš resur može da sadrži i fajlove.



Postavlja se logičko pitanje kako će klijent znati koj resur da iskoristi. URI može da bude dovoljan da izvrši adresiranje veb servisa, ali kako da se specificira sam resurs. Naravno

postoji više načina i principa kako se ovo može rešiti, ali recimo da je poželjan način korišćenjem WS-Addressing koji daje raznovrstan način adresiranja veb servisa (kada se uporedi sa klasičnim URI).

Spajanje veb servisa sa resursom naziva se WS-Resource. WS-Resource je zapravo krajnja putanja reference u WS-adresnom žargonu.



WSRF specification

Web Services Resources Framework koji se sastoji od pet različitih specifikacija. Sve one se na neki način odnose jedna prema drugoj u cilju pružanja upravljanja samim WS-Resources

WS-ResourceProperties

Sam resurs može i nemora da sadži tip resursa. Na primer u prethodnoj figuri svaki resur ima tri svojstva naziv fajla, veličinu ako i opis. WS-ResourceProperties opisuje svojstva resursa kao i pristup i oni su opisani u WSDL interfejsu.

WS-ResourceLifetime

Resur nema trivijalan životni vek. Drugim rečima, oni nisu statični entiteti kada se veb server pokrene i nestanu nakon gašenja. Resur može biti kreiran i uništen u bilo kom momentu. WS-ResourceLifetime pruža neko od osnovnih mehanizama za upravljenjem životnim ciklusom resursa.

WS-ServiceGroup

Pružá osnovne mehanizme grupisanja resursa u grupu. Grupisanje resursa nam olášava i povećava funkcionalnost jer možemo grupisati resurse i ponuditi ih kao jedan servis servis grupu.

WS-BaseFaults

Standardan naćin prijavljivanja greške prilikom WS-Service poziva.

WS-Notification

Omogućava klijentima mehanizam javljanja. Klijent mora da se prijavi(subscrib) na određeno dešavanje. WS-Notification će obavestiti sve klijente o promeni resursa prilikom promene.

WS-Addressing

Kao što je ranije pomenuto WS-Addressing specifikacija pruža nam mehanizam adresiranja do odgovarajućeg resursa. Za razliku od URI WS-Addressing je kolekcija Web servicea i resursi.

2.2 The Globus Toolkit 4

Posmatrano sa ovog aspekta WSRF deluje poprilično jednostavan i jasno definisan ali ko god da je programirao grid zasnovane aplikacije zna da sve ovo nije dovoljno za kreiranje grid aplikacije. WSRF je samo mali deo ali važan GT4 toolkita. WSRF je infrastruktura na kojoj je ceo toolkit zasnovan. Sam toolkit uključuje dosta komponenti koje se mogu koristiti u programiranju grid aplikacija.

2.2.1 Arhitektura

Globus toolkit se sastoji od dosta komponenti i sve komponente su podeljene u pet kategorija: Sigurnost, Upravljanje podacima, Upravljanjem izvršavanjem, informacionim servisom i standardnim „runtime“.

Iako sam Globus se fokusira na Web service sam toolkit uključuje i komponente koje nisu zasnovane na veb servisima. GridFTP komponente koje su prvo bile kreira zbog svoje nužnosti a kasnije proglašene GGF specifikacijom.

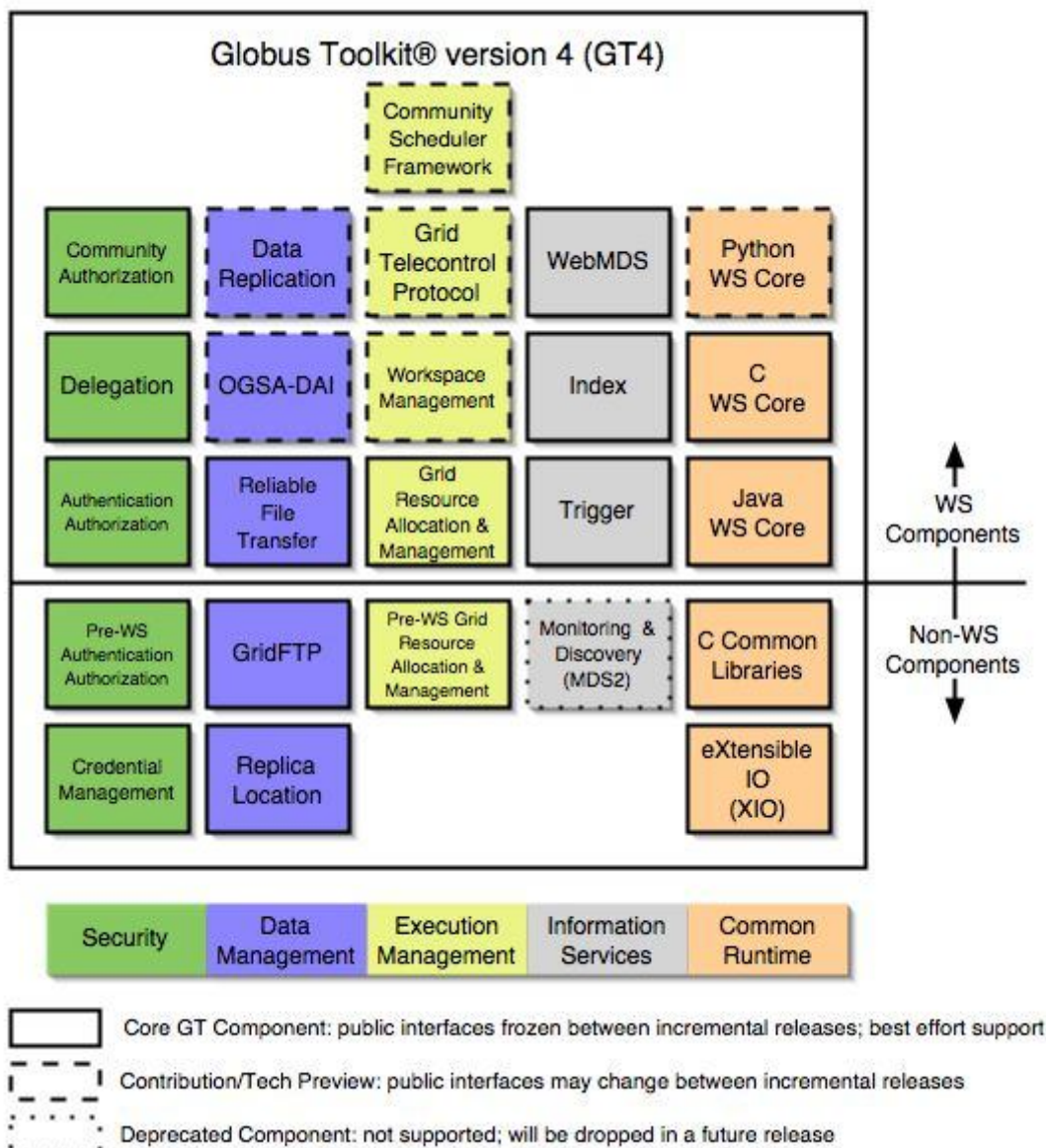


Figure 1 GT4 architecture

3 Flash VM

Zadatak ovog diplomskog rada je pronaći mehanizam za kreiranje grida korišćenjem bilo kog mehanizma tj. alata koji je prisutan na svim računarima ili pak u što većoj meri računara.

3.1 Istorijat Flash-a

Adobe Flash (ranije poznat kao **Shockwave Flash** ili **Macromedia Flash**) nastao je 1996 i postao vrlo popularan alat za kreiranje animiranog i interaktivnog sadržaja za veb stranice. Flash se najčešće koristi za kreiranje animacija, reklamnih banerima i raznih veb komponenti koje omogućavaju reprodukciju video sadržaja, a od skora, za kreiranje internet bogatih aplikacija (RIA).

Flash može da manipuliše vektorskom i rasterskom grafikom i podržava obostrani striming audia i videa. Podržava skriptni jezik pod nazivom ActionScript.

Fajlovi su u **SWF** formatu, tradicionalno nazvani „ShockWave Flash“, filmovi i mogu biti objekti neke web stranice ili pušteni u samostalnom Flash playeru. Moguće je sjediniti Flash film i samostalni plejer u Projekat. samo izvršujući Flash film.

Program Flash je osmislio i započeo da razvija Jonathan Gay-a. 1993 - Jonathan Gay, Charlie Jackson, and Michelle Welsh osnovali su kompaniju i nazvali je FutureWave Software i kreirali svoju prvi proizvod, SmartSketch. Aplikacija za „crtanje“ SmartSketch je osmišljena da olakša kreiranje crteža na internetu sa idejom kao da se koristi papir. U početku nije nudio ništa novo na tržištu pa kao takvo rešenje i nije bio zapažen. Ubrzo se prepoznala potreba vektorke veb animacije. 1995 SmartSketch-u je dodata slika-po-slika animacija(frame-by-frame) i tada je dobio naziv **FutureSplash Animator** za MAC i PC računare. Nakon čega je proizvod ponudjen Adobe-u i korišćen je od strane Microsofta za rane radove na internetu (MSN). Decembra 1996. Macromedia je kupila program i nazvala ga Flash koristeći reči „Future“ i „Splash“ od FutureWave naziva. Nakon kupovine Macromedia je ubrzala razvoj.

FutureSplash Animator (April,1996): prva verzija.

Macromedia Flash 1 (Novembar,1996): Macromedia je re-brandirala naziv FutureSplash Animator-a.

Macromedia Flash 2 (Jun 1997): Nova verzija, podrška za Biblioteku objekata(Grafičkih simbola u projektu)

Macromedia Flash 3 (Maj 1998): Nova verzija, Movie clip, JavaScript plug-in integracija

Macromedia Flash 4 (Jun 1999): Interne promenljive, napredniji ActionScript, striming MP3.

Macromedia Flash 5 (Avgust 2000): ActionScript 1.0(Zasnovan na ECMAScript), XML podrška HTML foramtiranje dinamičkog teksta.

Macromedia Flash MX (Mart 2002): Video kodeci, Unicode, UI komponente, kopresija, AS api za crtanje

Macromedia Flash Professional 2004(Septembar 2003): ActionScript 2, Objektno orjentisano programiranje za Flash, timeline Efekti.

Macromedia Flash 8 (Septembar 2005): Brizina, kvalitet i dostupnost na mobilnim uređajima (Flash Lite). Nove mogućnosti su

Adobe Flash CS3 (April 2007): Flash je otkupljen od strane kompanije Adobe, ActionScript 3 i bolja integracija sa Adobe-ovim paketima PhotoShop-om, Illustratorom. Bolja podrška za vektorsku grafiku.

Adobe Flash CS4 – Poslednja verzija Flash-a nazvanog “Diesel” I netreba ga mešati sa Flash Playerom 10 “Astro”, dodata je podrška za inferznu kinematiku (bones), osnovnu 3D objektnu manipulaciju, objektnu animaciju, poboljšan engine za tekst I dalje proširivanje ActionScripta 3.

Dalji razvoj

“Adobe labs” (ranije nazvan Macromedia Labs) nudi za preuzimanje: Flash 9, Flex 3 I ActionScript 9. Novo u razvoju Flash-a je povećavanje upotrebne vrednosti na prezentacionom sloju za mobilne uređaje “handheld devices”.

Od novembra 2007 Adobe aktivno razvija Adobe AIR projekat koje je platfomski nezavisan I omogućava programerima da iskoriste svoje znanje u kreiranju (Flash, Flex, HTML, AJAX) desktop aplikacija. Planirana je Podrška za DRM u Flash-u. kao I podršku za puštanje internet oglasa u potpunosti pre nego što bi bio reprodukovani neki video zapis.

Od Maja 2008 najavljen je “Open Screen Project” koj bi još povećao korisnički osećaj.

Programski jezik

U svojim počecima Flash je bio primarno namenjen animaciji i imao je ogroman ograničenja za programiranje. Od uvođenja ECMAScripta dakle iste sintakse kao i JavaScripte ali u drugom frameworku omogućio je osnovnu interaktivnost ka kreiranju i korišćenju dinamičkih (dugme, teksta, padajućih menija) koji su do sada vidjeni u velikom broju aplikacija.

Flash MX 2004 uvodi ActionScript 2 koj itekako olakšava kreiranje Flash Aplikacija.

Flash 9 uvodi ActionScript 3 je Objektno orjentisan programski jezik koji omogućava više kontrole kao i ponovnu upotrebu koda. Na posletku Flash biblioteke omogućavaju korišćenje XML-a za prikaz podataka u pretraživačima. Tehnologija je poznatija kao „Asynchronous Flash and XML“, poput AJAX-a. Potreba za Asynchronous Flash-om i XML je izazvala potrebu za formalniji pristupu i tehnologija koja to omogućava naziva se Adobe Flex, koj koristi Flash runtime za kreiranje internet Aplikacija.

Koncepti zaštite

Uzevši u obzir da se SWF fajlovi izvršavaju na klijentskoj strani mogući metodi zaštite su korišćenjem SWF obstruktor (obfuscators).

ActionScript

- ActionScript je jezik zasnovan na ECMAScriptu. Primarno se koristi za razvoj websajtova i softvera koristeći Adobe Flash Player platformu (u oliku SWF-a embedovanog u veb stranicu). Također se može sresti u aplikacija za rad sa bazom (Alfa Five), i u osnovnoj robotici (Make Controller Kit).

- ActionScrip je u početku zamišljen da olakša i omogući rad sa 2D vektorskom grafikom. Trenutne nadogranje jezika u verziji 3 su omogućile razvoj igara kao i bogatih internet aplikacija sa podrškom za video i audio striming.
- ActionScript jezik - Biografija
- Paradigma: Multi-paradigm
Kreiran:1998
Kreirao: Gary Grossman
Razvija:Macromedia(sada Adobe System)
Poslednja verzija: 3.0/27 Jun 2006
Način programiranja:Strickt, Static, Safe
Glavna implementacija: Adobe Flash, Adobe Flex
Glavni uticaj: JavaScript, Java
Operativni sistem: Nezavistan
Ekstenzija: AS
Internet media tip: application/actionscript

3.2 Primer programiranja u ActionScript-u

AS2

```
var greet:TextField = new TextField();  
greet.text = "Hello World";
```

```
this.addChild(greet);
```

AS3 Flash IDE

```
package com.example  
{  
    import flash.text.TextField;  
    import flash.display.Sprite;  
  
    public class Greeter extends Sprite  
    {  
        public function Greeter()  
        {  
            var txtHello:TextField = new TextField();  
            txtHello.text = "Hello World";  
            addChild(txtHello);  
        }  
    }  
}
```

AS3 Flex ID

Greeter.as

```
package  
{  
    public class Greeter  
    {  
        public static function sayHello():String  
        {  
            var greet:String = "Hello, world!";  
            return greet;  
        }  
    }  
}
```

Greeter.mxml

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*" layout="vertical"
creationComplete="initApp()">

    <mx:Script>
        <![CDATA[
            public function initApp():void
            {
                // Prints our "Hello, world!" message into "mainTxt".
                mainTxt.text = Greeter.sayHello();
            }
        ]]>
    </mx:Script>

    <mx:Label id="title" fontSize="24" fontStyle="bold" text="Hello, world! Example"/>
    <mx:TextArea id="mainTxt" width="250"/>

</mx:Application>

```

3.3**Tipovi podataka****3.3.1****ActionScript 2**

- String – Lista karaktera "Zdravo o vliki svete "
- Number – Bilo koja numerička vrednost.
- Boolean – Prost binarni zapis "true" ili "false".
Object – Objekat podržava kompleksne tipove od kojie nasledjuje. Dozvoljava grupisanje metoda, funkcija, parametara, i drugih objekata.
- ActionScript 2 kompleksni tipovi.
- MovieClip – ActionScript kreacija koja dozvoljava laku upotrebu vidljivih objekata
- TextField – Jednostavan dinamički ili tekstualno polje za unos. Nasledeno od MovieClip tipa.
- Button – Dugme sa 4 slike-frejma(stanja):Iznad, preko, dole i pritisnuto. Nasledeno od MovieClip tipa.
- Date – Dozvoljava pristup informacija o nekom specifičnoj tački u vremenu.
- Array – Lienerano skladištenje podataka.
- XML – XML objekat.
XMLNode - XML nod.

- LoadVars – Promenljiva za učitavanje dozvoljava skladištenje i slanje HTTP POST i HTTP GET promenljivih.
- Zvuk
- NetStream
- NetConnection
- MovieClipLoader
- EventListener

3.3.2

ActionScript 3

- Boolean – Tip koj dozvoljava vrednosti true,false, 1, 0. Nijedna druga vrednost nije dozvoljena.
- int – je tip koj je 32 bitni intičer izmedju -2,147,483,648 i 2,147,483,647.
- Null – tip koj sadrži samo jednu vrednost null. Ovo je poletna vrednost za String i sve kompleksne klase koje predstavljaju kompleksne tipove, kao i Object tip.
- Number – Broj je tip koj može da prestavlja intičer, unsigned integers i broj sa pokretnim zarezom.
- Koristi se 64 bitna preciznost kao što je specificirano po IEEE standardu za Binarnu Aritmetiku (IEEE-754).
- String – String je 16 bitni niz karaktera. Interno su smešteni kao Unikod karakteri, koristeći UTF-16 format. U ranijim verzijama korišćen je UTF-8 format.
- uint – Unsigned intidžer 32 bita i ima vrednost od 0 do 4,294,967,295.
void – Void je tip koj sadrži jednu vrednost „undefined“. U ranijim verzijama je bila početna vrednost tipa Objekat. Sada je početna vrednost null.
- ActionScript 3 kompleksni tipovi.
- Object – Objekat tip je definisan Objekat klasom. Objekat klasa služi kao osnovna klasa za definisanje u ActionScriptu.
- Array – Sadrži listu podataka. Iako je AS3 strongly-typed jezik nepodržava imenske nizove. Već sadržaj niza može biti bilo kog tipa.
- MovieClip – Animirani film objekat za prikaz. Nasledjen (sa manjim modifikacijama) i prestavlja glavnu Flashovu vremensku liniju.
- Bitmap – Ne animirana bitmapa za prikaz.
Shape – Ne animiran vektor za prikaz.
- ByteArray – Sadrži niz binarnih podataka.

- TextField – Jednostavan dinamički ili tekstualno polje za unos. Nasleđen od text field objekta.
- SimpleButton – Jdugme koja podržava stanja „gore“, „preko“, „dole“ i obaveznom zonom za klik.
- Date – Datum objekat i sadrži trenutno sistemsko vreme.
- Error – Generički objekat za grešku koj se javlja usled izuzetka (exception).
- Function – jezgro klasa za sve flash definicije.
- RegExp – Regularni izrazni objekti za string.
- Video – Objekat za prikazivanje video sadržaja podržava direktno(progressive download) ili striming (RTMP). Od verzije 9.0.115.0, podržan je „open h.264/MP4 high-definition video format“, ako i standardni Flash video(FLV).
XML – prepravljen XML objekat zasnovan na E4C standardu. Pristu je drukaciji od AS2 i zato je ponudjen tip XMLDocument kao kompatibilnost sa predhodnom verzijom.
- XMLList – Niz objekata različitog sadržaja u XML klasi.

Kao što možemo da prisetimo evolucija actionScript-a kao programsko jezika prevazilazi osnovu i prvobitnu namenu koja je bila da olakša manipulaciju grafikom i zvukom preko „interenta“

3.4 Rasprostranjenost Flasha.

3.4.1 Dostupnost flash vm

Globalno u svetu – Septembar 2008

	Flash Player 7	Flash Player 8	Flash Player 9	Flash Player Flash Player 9.0.115*
Razvijena tržišta	98.60%	98.30%	97.70%	89.40%
US/Canada	98.70%	98.30%	98.10%	90.00%
Evropa	99.10%	98.90%	98.00%	88.60%
Japan	97.50%	97.30%	96.30%	88.30%

Globalno u svetu – Jun 2008

	Flash Player 7	Flash Player 8	Flash Player 9	Flash Player Flash Player 9.0.115*
Razvijena tržišta	99.00%	98.70%	97.70%	81.70%
US/Canada	99.10%	98.90%	97.80%	83.30%
Evropa	98.50%	97.90%	96.50%	78.60%
Japan	99.30%	99.30%	98.80%	81.30%
Emerging Markets3	97.30%	97.10%	96.20%	82.40%

Globalno u svetu – Mart 2008

	Flash Player 7	Flash Player 8	Flash Player 9	Flash Player Flash Player 9.0.115*
Razvijena tržišta	98.80%	98.50%	97.20%	61.80%
US/Canada	98.70%	98.50%	97.30%	62.10%
Evropa	98.50%	98.10%	96.50%	61.90%
Japan	99.80%	99.50%	98.00%	61.00%

Globalno u svetu – Decembar 2007

	Flash Player 64	Flash Player 7	Flash Player 8	Flash Player 9
Razvijena tržišta	98.80%	98.80%	98.30%	95.70%
US/Canada	99.00%	99.00%	98.50%	96.80%
Evropa	98.10%	98.10%	97.60%	94.30%
Japan	99.50%	99.50%	99.30%	95.30%
Emerging Markets3	97.40%	97.30%	95.50%	93.30%

http://www.adobe.com/products/player_census/flashplayer/version_penetration.html

3.4.2 Instalirane verzije flash vm

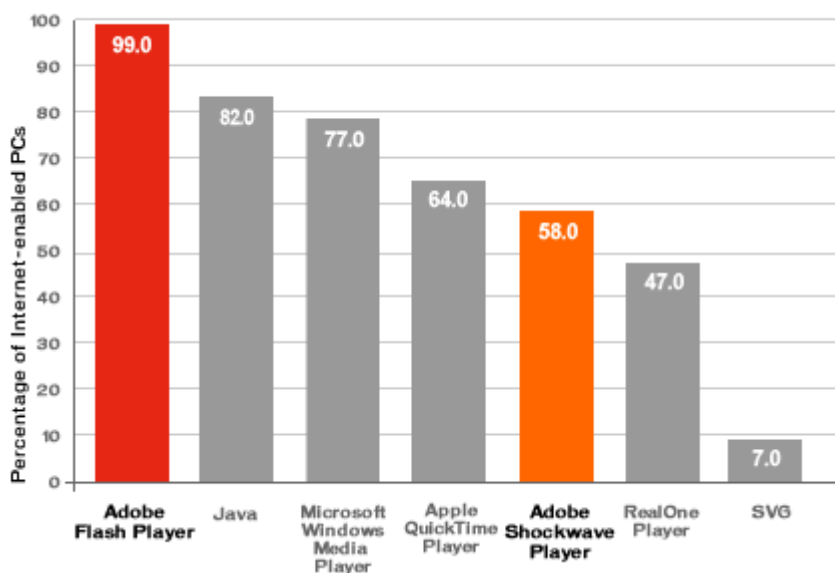


Figure 2 Napomena statistika obuhvata US, Kanadu, Francusku, Nemačku, Japan i predstavlja stanje Septembra 2008.

http://www.adobe.com/products/player_census/flashplayer/

3.4.3 Statistika korišćenja interneta

Regioni	Populacija	Internet korisnici	Internet Usage,	% populacije	korisnici % od sveta	porast u korišćenju 2000-2008
Africa	955206348	4514400	51065630	5.30%	3.50%	10.312
Azija	3776181949	114304000	578538257	15.30%	39.50%	4.061
Evropa	800401065	105096093	384633765	48.10%	26.30%	2.66
Srednji istok	197090443	3284800	41939200	21.30%	2.90%	11.768
Severna amerika	337167248	108096800	248241969	73.60%	17.00%	1.296
Latino amerika	576091673	18068919	139009209	24.10%	9.50%	6.693
Australija	33981562	7620480	20204331	59.50%	1.40%	1.651
Ukupno Svet	6676120288	360985492	1463632361	21.90%	100.00%	3.055

<http://www.internetworldstats.com/stats.htm>

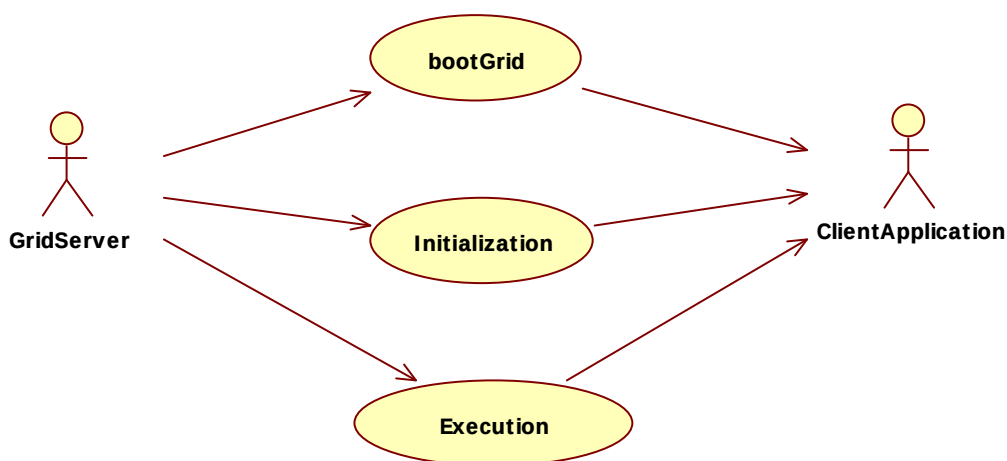
4 Flash kao grid

Dostupnos flash vm mašine predstavlja flash kao idealnog kandidata za formiranje grid-a.

4.1 Slučajevi korišćenja DataGrid sistema

U sistemu *DataGrid* postoje tri osnovna slučajeve korišćenja (Slika 4.1):

- Priprema *Grid* kontrole (*bootGrid*)
- Inicijalizacija (*Initialization*)
- Izvršenje (*Execution*)

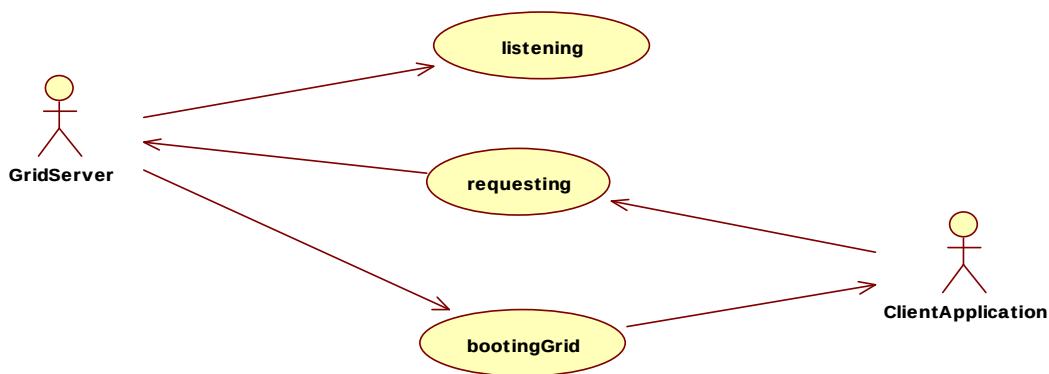


Slika 4.1: Osnovni slučajeve korišćenja

Ova tri SK istovremeno predstavljaju i faze izvršenja aplikacije. Osnovni akteri (uloge) u sistemu su serverska aplikacija (*GridServer*) i klijentska aplikacija (*ClientApplication*). Navedeni SK se odvijaju u komunikacionom kontekstu između ova dva aktera.

4.1.1 Priprema *Grid* kontrole

Serverska aplikacija (*GridServer*) se inicijalno nalazi (Slika 4.2) u režimu neprekidnog osluškivanja (*listening*), očekujući zahtev nekog od klijenata da joj pristupi. U ovim uslovima se odvija SK „Priprema *Grid* kontrole“ (*bootGrid*).



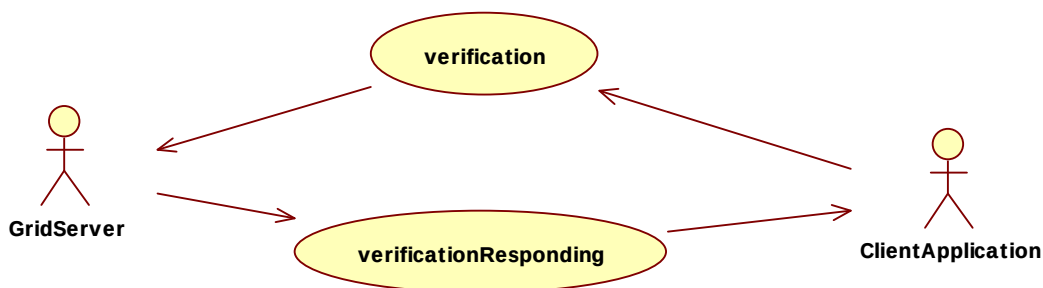
Slika 4.2: SK Priprema Grid kontrole

Klijent predstavlja potpuno nezavisnu klijentsku platformu (u odnosu na *GridServer*), koja je u stanju da uspostavi standardnu Internet komunikaciju. Pri prvom obraćanju serveru (*requesting*), vrši se registracija klijenta, dodeljuje mu se novi identifikator (*UID*), kreira se novi klijentski objekat koji se isporučuje kroz Web resurs (npr. *swf* fajl), i koji predstavlja osnovni entitet posredstvom koga se vrši komunikacija (*bootingGrid*). U slučaju da se klijent već obraćao serveru, on poseduje *UID*, tako da mu se samo isporučuje klijentski objekat.

4.1.2

1.2 Inicijalizacija

Priprema *Grid* kontrole, predstavlja permanentnu aktivnost registrovanja klijenta za korišćenje od strane sistema. Inicijalizacija (*Initialization*) je SK kojim započinje eksploatacija klijenata za potrebe *GridServer*-a. Ona specijalizuje u dva SK, preciznije ima dve faze: verifikaciju klijenta i odgovor na verifikaciju.



Slika 4.3: SK Inicijalizacija

Registrovani klijent, koji je na mreži, sadrži živ klijentski objekat, koji šalje signal *GridServer*-u da je klijent spreman za korišćenje (*verification*). Server evidentira aktivnog klijenta i šalje mu odgovarajući odgovor (*verificationResponding*):

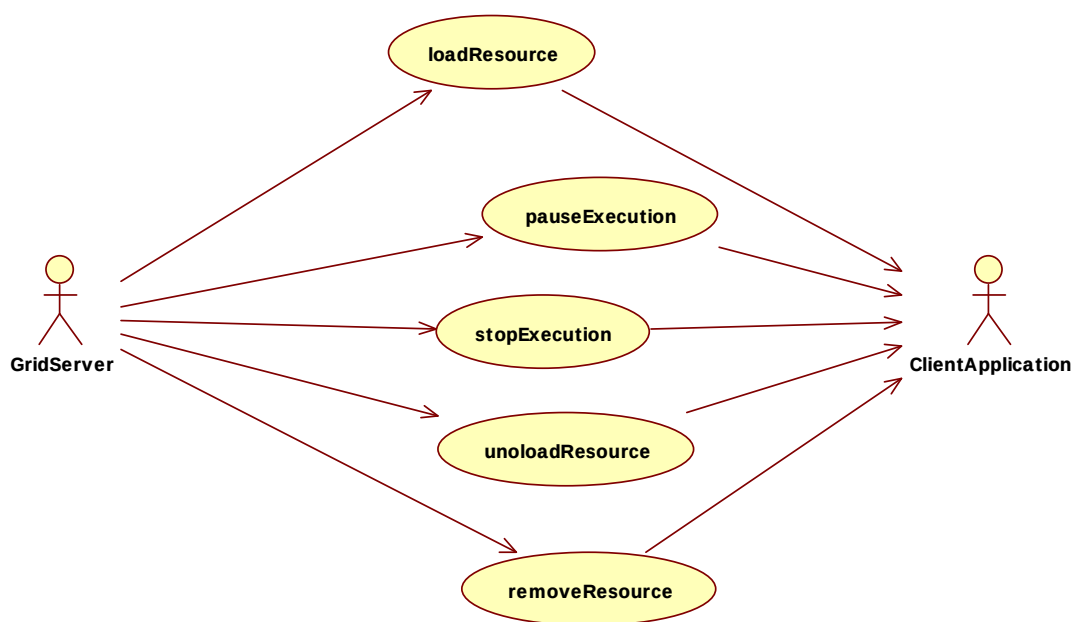
- Odgovor sa resursom – kroz ovaj tip odgovora obavlja isporuku resursa klijentu,
- Samo odgovor – slučaj kada serverska aplikacija nema trenutne potrebe za korišćenje klijenta.

U oba slučaja *GridServer* verifikuje klijenta za potencijalnog egzekutora, smeštajući ga u listu verifikovanih klijenata. Na taj način server vrši *pooling* klijenata, vodeći računa o njihovoj trenutnoj zauzetosti, ili spremnosti za prihvatanje novodobijenih zadataka.

4.1.3

1.3 Izvršenje

SK Izvršenje (*Execution*) predstavlja najsloženiji SK DataGrid sistema. Ovaj slučaj prikazuje da je ključna funkcionalnost koja je rešavana u sistemu komunikaciona, posredstvom koje serverska aplikacija distribuira aplikacione resurse (objekte, klijentsku aplikaciju, podatke), na razne tipove klijentskih platformi. Serverska aplikacija kontroliše isporuku, izvršenje i uklanjanje ovih resursa sa klijentske platforme.



Slika 4: SK Izvršenje

U SK Izvršenje postoji pet slučajeva korišćenja (Slika 4):

- Učitavanje resursa
- Pauziranje izvršenja
- Zaustavljanje izvršenja
- Pražnjenje resursa
- Uklanjanje resursa

Učitavanje resursa

Učitavanje resursa (*loadResource*) je SK ...

Pauziranje izvršenja

Pauziranje izvršenja (*pauseExecution*) je SK ...

Zaustavljanje izvršenja

Zaustavljanje izvršenja (*stopExecution*) je SK ...

Pražnjenje resursa

Pražnjenje resursa (*unloadResource*) je SK ...

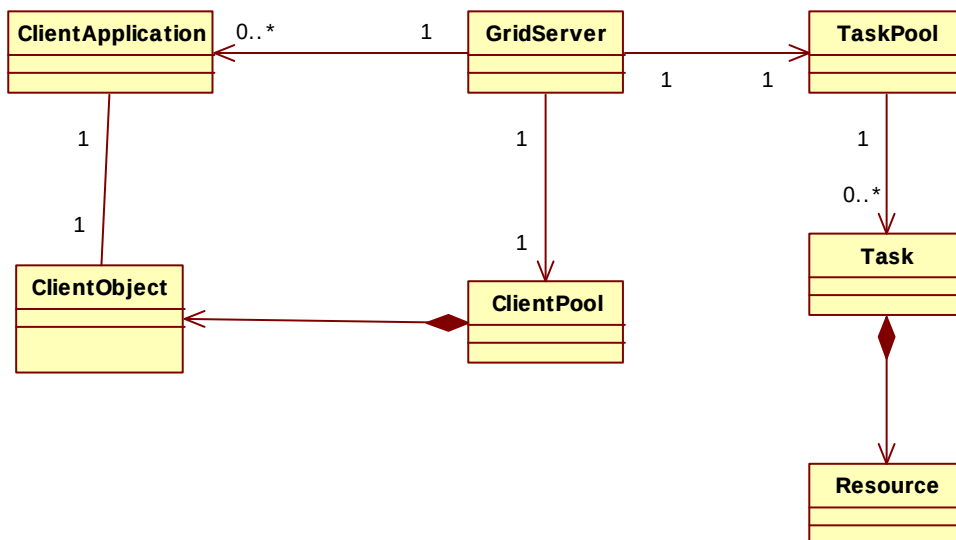
Uklanjanje resursa

Uklanjanje resursa (*removeResource*) je SK ...

4.2

2. Konceptualni model

Konceptualni model (Slika 4.5) predstavlja osnovne statičke koncepte sistema. Ovaj model se dalje razvija u dijagram klasa, dodavajući na koncepte metode koje se implementiraju i podatke koje koncepti sadrže. Osnovna namena konceptualnog modela je da prikaže koncepte i relacije između njih.



Slika 4.5: Konceptualni model aplikacije

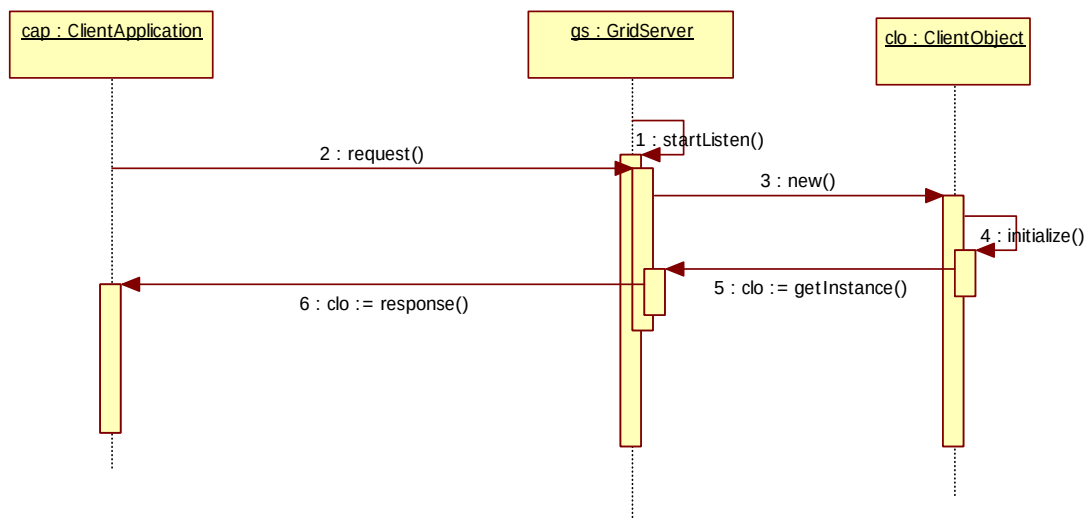
Osnovna klasa aplikacije je *GridServer*. Ona u sebi sadrži dva *pool*-a – koncepti *TaskPool* i *ClientPool*. *GridServer* takođe zna za koncept koji se razvija na klijentskoj strani – *ClientApplication*. Ova klasa enkapsulira aplikaciju koja se distribuira po zahtevu na klijentsku platformu posredstvom mreže. Svaki primerak klijentske aplikacije dobija jedinstveni identifikator – *UID*, koji omogućava njegovu identifikaciju kod konkurentskog pristupa većeg broja klijenata na *GridServer*. Sve akcije koje se izvršavaju na klijentskoj platformi su enkapsulirane u koncept (klasu) *ClientObject*. Klijentska aplikacija, da bi bila upotrebljiva, mora da sadrži jednu instancu *ClientObject* klase. Preko ove klase se vrši i komunikacija između serverske i klijentske platforme. *GridServer* čuva kolekciju aktivnih *ClientObject* objekata u klasi *ClientPool*. Klasa *ClientObject* zapravo servisira zahteve *GridServer*-a za izvršenje određenih zadataka na klijentskoj strani. *GridServer* prima zadatke od entiteta van sistema, i dodaje ih u *TaskPool*. Zadaci su predstavljeni klasom *Task* i sadrže reference na resurse, neophodne za njihovo izvršenje. Resursi su predstavljeni konceptom *Resource*, mogu da budu bilo kog tipa i isporučuju se klijentskoj aplikaciji, kako bi se izvršili na klijentskoj platformi.

4.3 3. Sekvencijalni dijagrami

Sekvencijalni dijagrami su vrsta dijagrama interakcije koji prikazuju hronologiju interakcije između objekata u sistemu. U toku interakcije, između objekata se razmenjuju poruke u vidu funkcionalnih poziva i podaci u vidu objekata ili literala (podataka osnovnih tipova).

4.3.1 3.1 Priprema *Grid* kontrole

Na osnovu istoimenog SK konstruisan je sekvencijalni dijagram (Slika 4.6). Pokretanjem *GridServer*-a, poziva se interna funkcija *startListen*, kojom se *GridServer* dovodi u režim neprekidnog osluškivanja (*listening*).

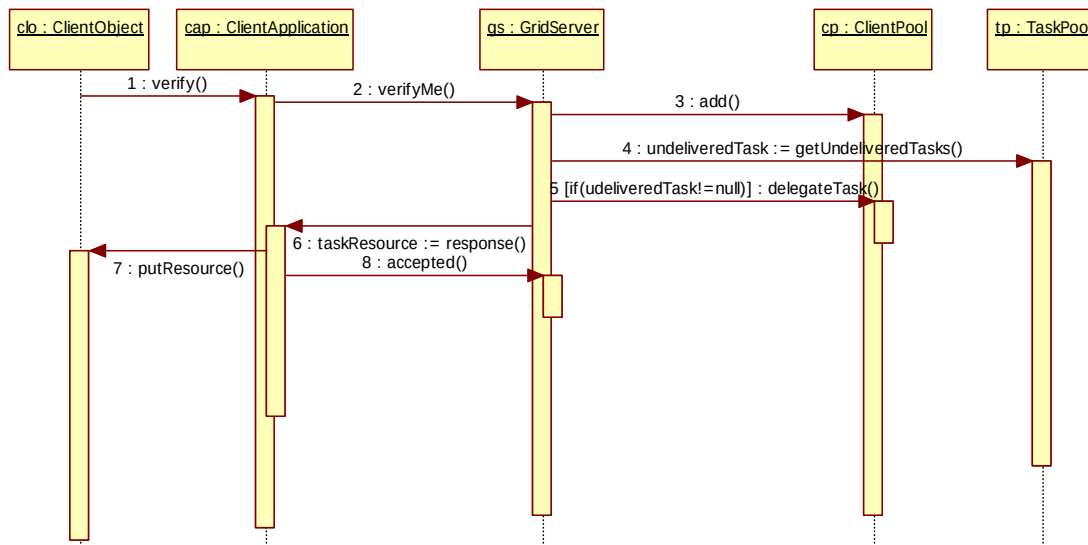


Slika 4.6: Sekvencijalni dijagram za SK Priprema *Grid* kontrole

Klijentska aplikacija (*ClientApplication*) šalje zahtev serveru za nekim resursom (poziv *request*). *GridServer* kreira novi klijentski objekt (*clo*). Pri konstrukciji, ovaj objekt se inicijalizuje potrebnim podacima (*initialize*), nakon čega ga *GridServer* prosleđuje klijentskoj aplikaciji (metode *getInstance* i *response*).

4.3.2 3.2 Inicijalizacija

Inicijalizacija (*Initialization*) je SK kojim započinje eksploatacija klijenata za potrebe *GridServer*-a (Slika 4.6). Ona podrazumeva da se na klijentskoj platformi već nalazi klijentski objekat, koji započinje inicijalizaciju pozivom metode *verify* klijentske aplikacije. Ovaj poziv se prosleđuje *GridServer*-u, koji evidentira - dodaje aktivnog klijenta u listu, enkapsuliranu u klasi *ClientPool*. Pošto se radi o sinhronoj komunikaciji, *GridServer* najpe proveriti da li postoji neraspoređen zadatak u listi zadataka koja se održava u klasi *TaskPool*. Ako takav zadatak postoji, *GridServer* ga dobija preko metode *getUndeliveredTask* i dodeljuje ga novom klijentskom objektu, smeštenom u *ClientPool*-u.



Slika 7: Sekvencijalni dijagram za SK Inicijalizacija

U odgovoru *GridServer* –a klijentu (*response*), šalje se resurs(i), neophodan za izvršenje zadatka. Ovaj resurs se prosleđuje klijentskom objektu, nakon čega klijentska aplikacija šalje potvrdu preuzimanja istog. Klijentski objekat i aplikacija su spremni za izvršenje zadatka.

4.3.3

3.3 Izvršenje

Izvršenje (*Execution*) je najkompleksniji SK sistema, prvenstveno zbog velikog broja stanja u kojima može da se nađe resurs pri izvršavanju na klijentskoj strani (Poglavlje 4.3).

Za sada je predviđena osnovna funkcionalnost izvršavanja zadataka. Zadatak je moguće pokrenuti na bilo kom nodu i primiti njegov rezultat. Dalji razvoj podrazumeva kreiranje mehanizama koji bi omogućili efikasno rešavanje konkurentnih zadataka.

5

Dalji razvoj

Postignuto

- Kreiranje flash grid mreže i distribucije zadataka.

Dalji razvoj i mogući smerovi

- p2p.
- Implementacija Erlanga (Paralelizam).
- Ublique computing.
- API ka klasičnom gridu.
- Grid neuronska mreža.
- Veb operativni sistemi.

Literatura:

- [1] **Wikipedia Distributed computing** http://en.wikipedia.org/wiki/Distributed_computing
- [2] **Grid computing** http://en.wikipedia.org/wiki/Grid_computing
- [3] **Cloud computing** http://en.wikipedia.org/wiki/Cloud_computing
- [4] **Cluster** (computing) http://en.wikipedia.org/wiki/Computer_cluster
- [5] **Erlang** <http://erlang.org/doc.html>
- [6] **Globus Alliance** <http://www.globus.org/alliance/publications/papers.php>
- [7] **Adobe** <http://www.adobe.com/>
- [8] Milosavljević, M., **Naziv knjige**, Izdavač, Beograd, 2008.
- [9] Milosavljević, M., „Naziv članka“, Univerzitet Singidunum, Beograd, 2009.
- [10] Njeguš, A., „Projektovanje informacionih sistema“, prezentacije sa predavanja, Univerzitet Singidunum, Beograd, 2008.