Proširenje modela dinamičkog ažuriranja softvera na promjenu hijerarhije klasa i detekciju fenomena izvođenja

Danijel Mlinarić
mentor: doc. dr. sc. Boris Milašinović
Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva

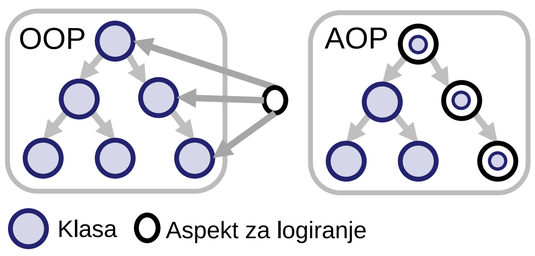
1. Uvod

Dinamičko ažuriranje softvera (*engl. Dynamic Software Updating - DSU*) može se definirati kao zamjena trenutne verzije softvera s novom verzijom tijekom izvođenja programa. U odnosu na standardne postupke ažuriranja uvodi određene izazove: u kojem trenutku ažurirati program, kako očuvati stanje programa i podržati što veći broj promjena bez narušavanja performanci.

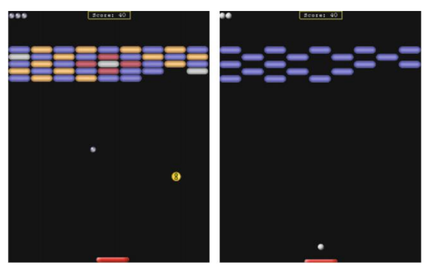
U programskim jezicima koji se kompiliraju za virtualno računalo (npr. JVM – *Java Virtual Machine*) postoji nekoliko koncepata čiji pristupi koriste različite mehanizme kako bi odgovorili izazovima. DAOP (*engl. Dynamic Aspect-Oriented Programming*) je mehanizam koji omogućava dinamičko uplitanje aspekata i time omogućava primjenu promjena u programu tijekom izvođenja, ali se pomoću njih ne mogu definirati promjene u hijerarhiji klasa.

Dinamičko ažuriranje dovoljno je sazrelo kako bi se koristilo u razvojnim okolinama, dok se u produkcijskim okolinama kao posljedica dužeg izvođenja, nakon ažuriranja mogu pojaviti fenomeni izvođenja (engl. *run-time phenomena*).

Fenomeni izvođenja su pojave, u kojima stanje programa nakon dinamičkog ažuriranja nije isto kao što bi bilo nakon standardnog ažuriranja (npr. *engl. lost state*) (slika 2).



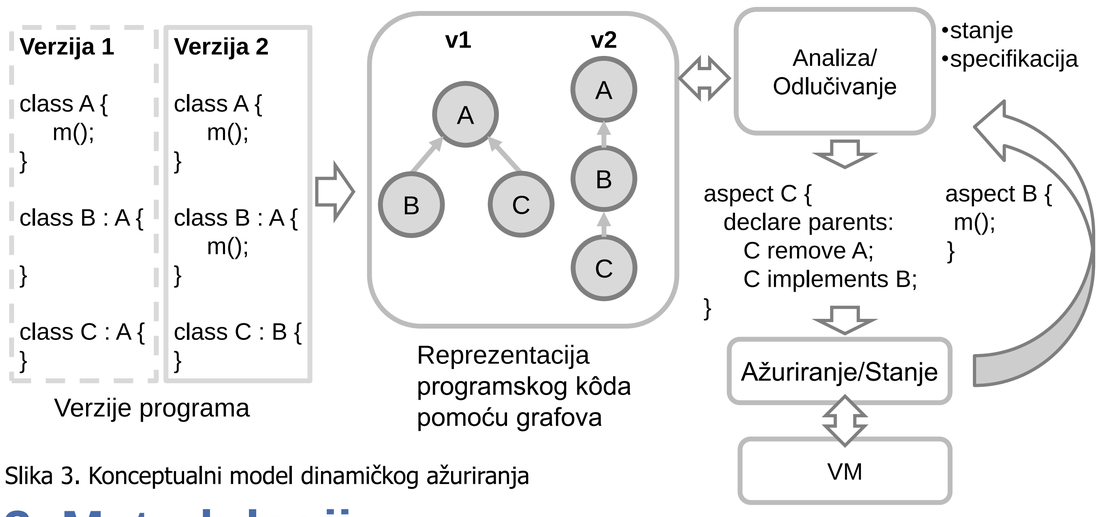
Slika 1. Korištenje aspekta za dnevnik (*engl. log*)



Slika 2. Breakout igra: prikaz „izgubljenog stanja” (*engl. lost state*)

2. Opis problema

Za rješenje problema promjene hijerarhije pomoću dinamičkih aspekata i detekciju fenomena izvođenja, cilj istraživanja je proširiti model dinamičkog ažuriranja. Na slici 3 je konceptualni prikaz modela koji kao ulaz prima dvije različite verzije programa. Primljeni programski kôd pretvara u apstraktne strukture, koje koristi za analizu promjena između programa. Izlaz analize su promjene programa u obliku dinamičkih aspekata. Promjene uz trenutno stanje programa koriste se za detekciju fenomena izvođenja te se pri tome donosi odluka o odbacivanju ili primjeni ažuriranja.



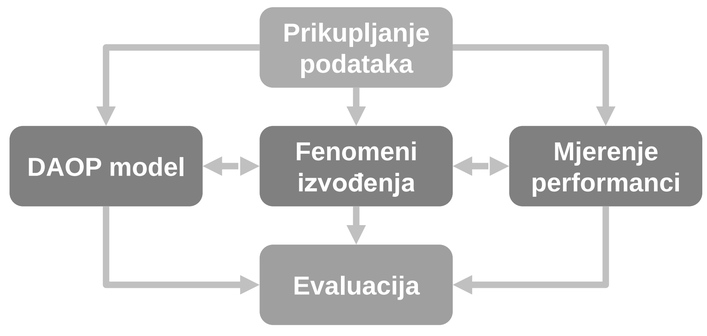
Slika 3. Konceptualni model dinamičkog ažuriranja

3. Metodologija

Plan istraživanja je podijeljen u različite faze koje se mogu odvijati paralelno (slika 4). U prvoj fazi provodi se prikupljanje podataka iz postojećih istraživanja koji će biti korišteni za kasnije faze:

- model dinamičkog ažuriranja pomoću dinamičkih aspekata
- detekcija fenomena izvođenja (analiza/algoritam/odlučivanje)
- metodologija za mjerenje performanci

Rezultati navedenih faza su znanstveni doprinos doktorskog istraživanja. Na kraju se provodi faza s empirijskom evaluacijom nad rezultatima prethodne tri faze.



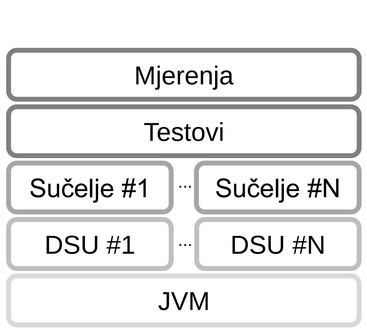
Slika 4. Plan istraživanja

Rezultati prve faze kao što su kategorizacija korištenih mehanizama i tehnika, specifikacija promjena i podržane dinamičke promjene koristit će se za istraživanje modela dinamičkih aspekata, a informacije o podržanima promjenama i metodologija za mjerenje performanci koristi za izradu alata za mjerenje performanci.

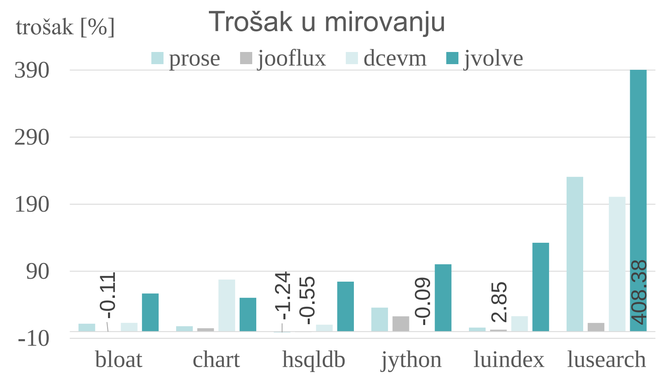
Kategorizacija postojećih istraživanja statičke i dinamičke analize programa, tehnike za strojno odlučivanje i identificirani fenomeni izvođenja koristit će se za istraživanje algoritma koji detektira fenomene izvođenja.

4. Rezultati

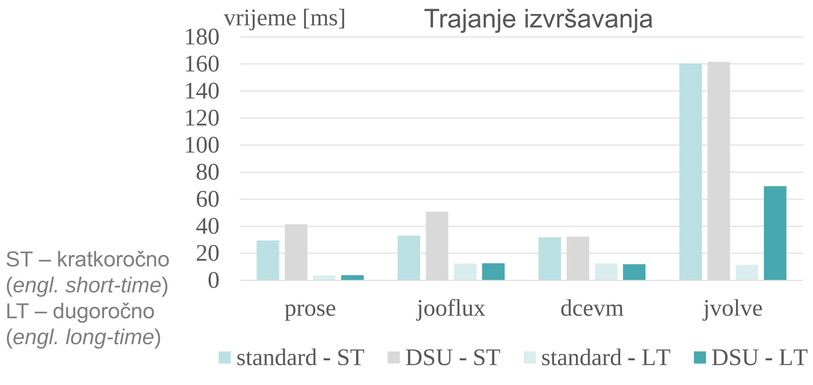
U dosadašnjem istraživanju najviše je razrađena faza mjerenja performanci dinamički ažuriranog programa. Definirana je struktura alata za mjerenje prema slici 5. Sastoji se od različitih vrsta testova i mjerenja koja se obavljaju nad različitim pristupima dinamičkog ažuriranja. Mjerenja uključuju vremensko mjerenje izvršavanja testova ili njihovu razliku te razliku u korištenju memorije. Testovi uključuju vremenski trošak dinamičkog ažuriranja (neovisno jesu li promjene postojale) (slika 6), vremensku razliku u izvršavanju programskih odsječaka nakon ažuriranja (slika 7) te trajanje ažuriranja s obzirom na pojedinu promjenu (npr. dodavanje ili brisanje metode, polja, klase).



Slika 5. Struktura mjerenja performanci



Slika 6. Utjecaj na performance u mirovanju (bez ažuriranja)



Slika 7. Trajanje izvršavanja ažuriranog programskog odsječka

Prikazani rezultati pokazuju kako dinamičko ažuriranje većinom unosi neznatan vremenski trošak, dok pristupi s dinamičkim aspektima (*jooflux*, *prose*) u većini slučajeva unose manji utjecaj na performance od pristupa koji koriste promijenjen Javin virtualni stroj (*dcevm*, *jvolve*). Slučajevi gdje su performance lošije pokazuju kako je metodologija mjerenja performanci dinamičkog ažuriranja korisna kako bi se takvi slučajevi detektirali i poboljšali.

5. Zaključak

Preliminarni rezultati mjerenja performanci pokazali su prednosti pristupa s dinamičkim aspektima čime ohrabruju nastavak rada na modelu dinamičkog ažuriranja pomoću dinamičkih aspekata. Kao dio modela sljedeći korak istraživanja je definiranje prikladnog oblika za usporedbu hijerarhija klasa u programima te izrada algoritma za detekciju promjena hijerarhije.