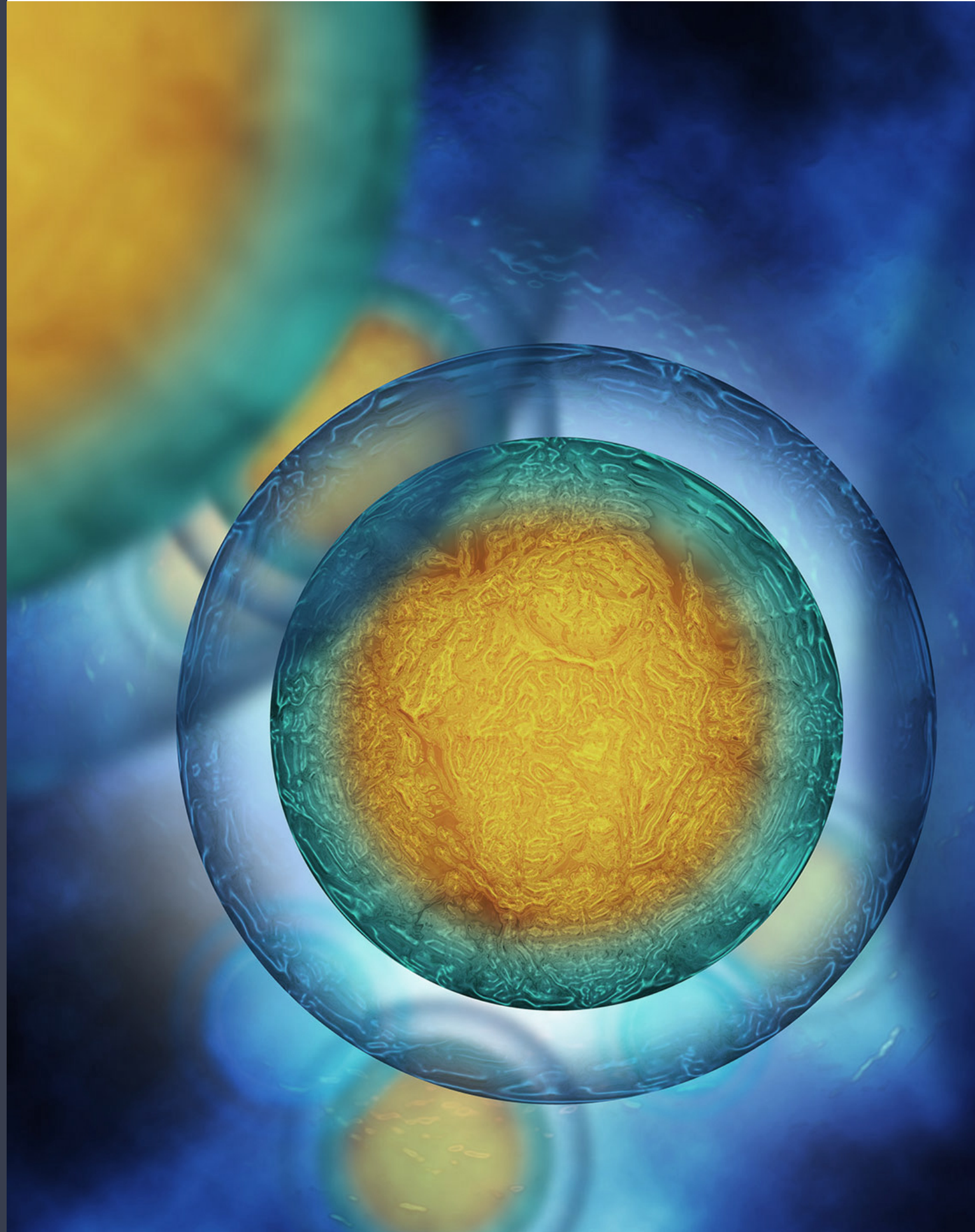
FE3CWS

OBUKA ZA NASTAVNIKE U AMSTERDAMU

Intelektualni doprinos O2 iz ERASMUS+ projekta 2017-1-SK01-KA203-035402



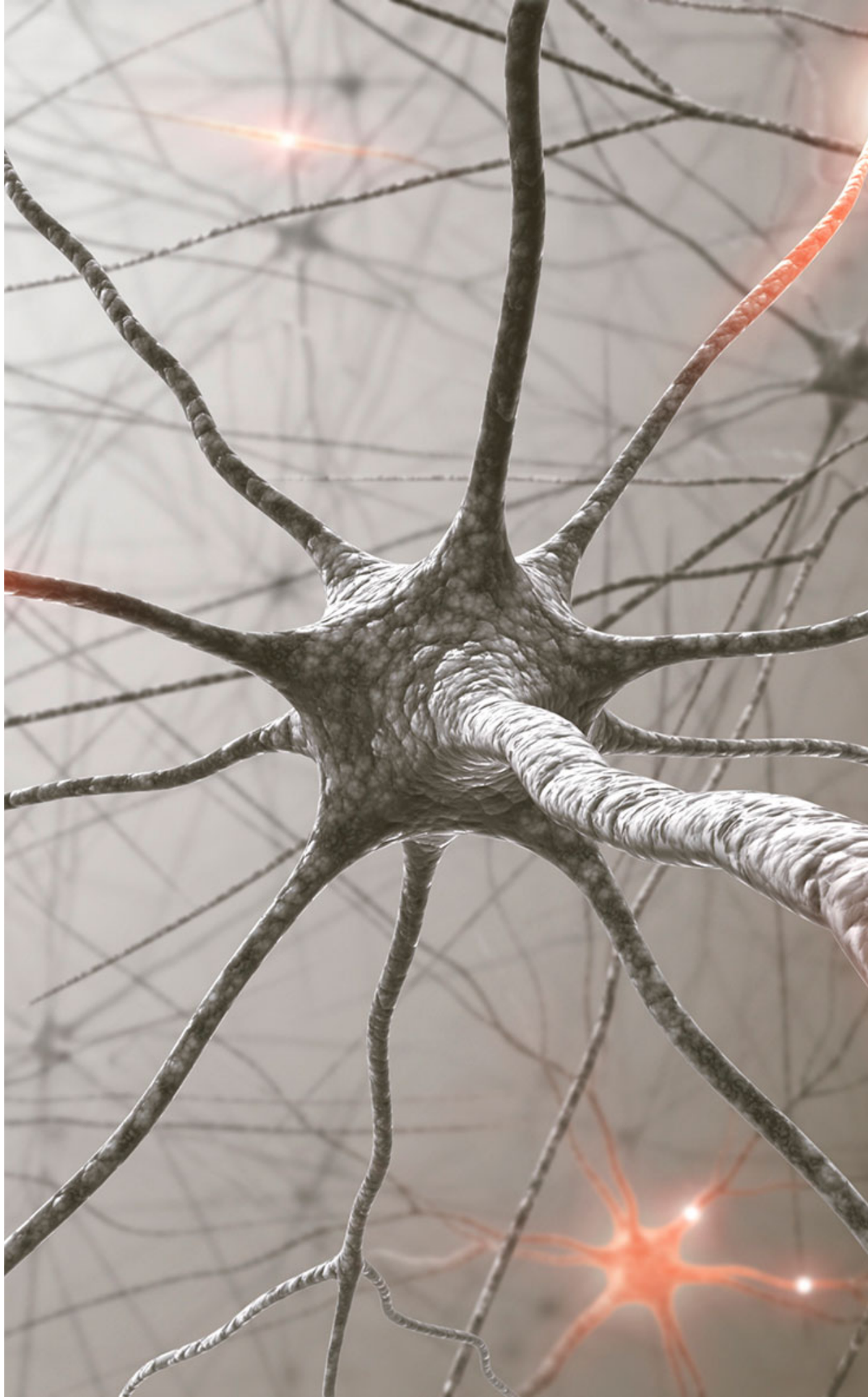
Nekoliko riječi o

SADRŽAJU

- 6 tema o kompoziciji, razumljivosti i ispravnosti softvera
- Dostupan na 7 jezika: engleski, mađarski, slovački, hrvatski, rumunjski, bugarski i portugalski

Co-funded by the Erasmus+ Programme of the European Union

SADRŽAJ



KORISNIČKI ORIJENTIRANO ČUNARSTVO U OBLAKU

Računarstvo u oblaku je postalo ključna tehnologija i stoga dio mnogih kurikuluma iz računarstva. Korisnički orijentirano istraživanje fokusira se na strategije za procjenu vremena izvođenja i troškova za implementaciju u oblaku aplikacija s primjenom u stvarnom svijetu.

Važan aspekt u području računarstva u oblaku je pomoć korisnicima pri donošenju odluka. Pritom, odluke se odnose na sljedeća pitanja:

– Kako se aplikacija ponaša na virtualnim resursima?

– Koliko virtualnih resursa, kakvog tipa i od kojeg dobavljača usluge oblaka treba biti nabavljeno za implementaciju aplikacije?

– Na koliko dugo? Koliko će koštati?

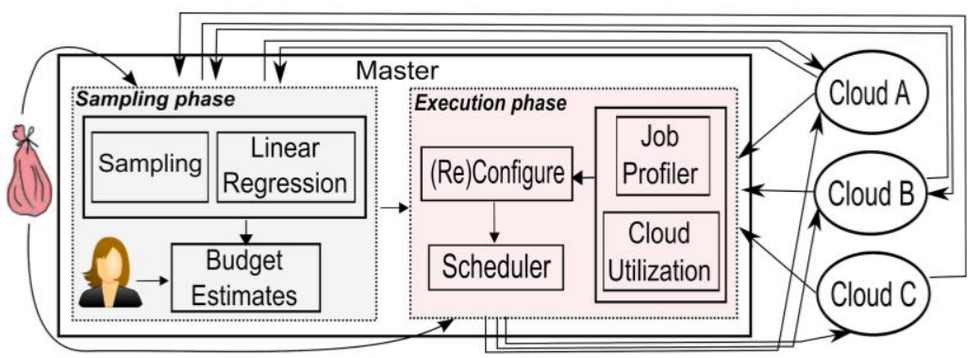
Ova pitanja se obično modeliraju kao problem planiranja, radeći uz pretpostavku da nema a-priori znanja o aplikaciji. Jedan tipičan, jednostavan skup zahtjeva jest da je aplikacija uspješno implementirana u oblaku, a da su troškovi minimizirani.

ARHITEKTURA PLANERA ZA APLIKACIJU U OBLAKU

BaTS planer [4] je razvijen za pomoć korisnicima pri implementaciji njihovih aplikacija u oblaku. Kako bi to postigao, koristi se samoplanirajući pristup i pritom se redovno provjerava napredak implementacije.

Slika 1 prikazuje arhitekturu BaTS-a. Tokom faze uzorkovanja, BaTS prikuplja statističke podatke o vremenu izvođenja nekih zadataka aplikacije, koristeći uzorkovanje sa zamjenom. Ovdje je potreban samo mali uzorak (30-50 zadataka) da se izračuna srednja vrijednost i standardna devijacija vremena izvođenja zadataka na raznim ponudama oblaka. Linearna regresija se koristi za optimizaciju računanja troškova i procjene trajanja izvođenja.

Tokom faze izvođenja, u pravilnim vremenskim intervalima, trenutna konfiguracija se ponovo procjenjuje, kako bi se provjerilo je li odabrani plan još uvijek ostvariv.



Ako se očekuje probijanje očekivanog budžeta, profitabilnije mašine (bolji odnos cijene i performanse) se nabavljaju. Ako se očekuje povreda trajanja izvođenja, nabavljaju se brže mašine.

KORIŠTENJE BATS METODOLOGIJE NA AWS RESURSIMA

Predstavimo sada problem pomaganja vlasnicima aplikacije koji žele napraviti najbolje izbore u smislu virtualnih resursa pri izvođenju njihove aplikacije na Amazon EC2 (AWS) [7] resursima.

Optimizirana faza uzorkovanja

Osnovna ideja je koristiti prosječno vrijeme izvođenja za svaki tip virtualnog resursa za izračun troškova i procjene vremena izvođenja. Međutim, dobivanje tih statistika može značajno utjecati na trošak, obzirom na veliki broj tipova ponuda AWS EC2 (trenutno ih ima 123 [8]).

Slika 2 prikazuje slučajno izabrane zadatke aplikacije, pri čemu vremena izvođenja zadataka prate neku vjerojatnosnu razdiobu. Vremena izvođenja tih zadataka su iskorištena za izračun statistike za jednu ponudu oblaka. Nakon prikupljanja statistike za sve dostupne ponude oblaka, može se izračunati troškove i procijeniti vremena izvođenja. Ako bismo htjeli evaluirati svaku trenutnu ponudu AWS EC2, to bi značilo izvođenje 30 zadataka na svakoj od 123 tipova mašina. Kad bismo jednostavno izvodili različite skupove slučajno izabranih zadataka (ukupno 3690), to bi dovelo do dugotrajne (i skupe) faze uzorkovanja, a vrlo moguće i do pretvaranja bilo kakve korisnikove odluke u nevažnu iz dva razloga: a) preostalo je premalo zadataka za izvođenje, te b) korisnikov budžet bi već mogao biti premašen. Kad bismo na svim tipovima mašina izveli isti skup slučajno izabranih zadataka, to bi još uvijek dovelo do dugotrajne (i skupe) faze uzorkovanja.

Stoga smo ovu fazu optimizirali koristeći linearnu regresiju da bi smanjili ukupni broj zadataka koje je potrebno izvoditi prije pripreme statistike. Izvodimo isti skup od 7 slučajno ozabranih zadataka na svakom tipu mašine i prikupljamo vremena izvođenja [9].

Nakon toga, izvodimo 23 slučajno izabranih zadataka na mašinama koje prve postanu dostupne. Slika 3 ilustrira naš pristup. Koristeći vremena izvođenja repliciranih 7 zadataka, uspostavljamo linearni odnos između vremena izvođenja za zadatke aplikacije u ovisnosti o tipovima mašina. Zatim koristimo taj linearni odnos da pridružimo 23 vremena izvođenja svim ostalim tipovima mašina. Takvim pridruživanjem dobivamo skup od 30 vremena izvođenja za svaki tip mašine, izvodeći samo 884 zadatka umjesto 3690.

Različiti redovi veličine u sustavu naplate

Jednom kad su dobivene procjene vremena izvođenja zadataka, procjene troška i vremena izvođenja se računaju koristeći modificirani Bounded Knapsack algoritam [11]. Ali kad se promatraju AWS EC2 resursi s različitim modelima sustava naplate, kao što su “spot instances”, ovaj pristup se ne skalira dobro zbog različitih redova veličine.

Da bismo riješili taj problem, zamijenili smo determinizam Bounded Knapsack pristupa, za skalabilnost pristupa genetskog algoritma [12].

Koristeći genetski algoritam, možemo aproksimirati Pareto frontu (optimalni skup) izvodivih planova za danu aplikaciju i dani skup tipova mašina. Slika 4 prikazuje realnu Pareto frontu i dvije procjene za aplikaciju s višemodalnom razdiobom vremena izvođenja zadataka. Naše riješenje generira ispravne Pareto fronte unutar 1 sekunde.

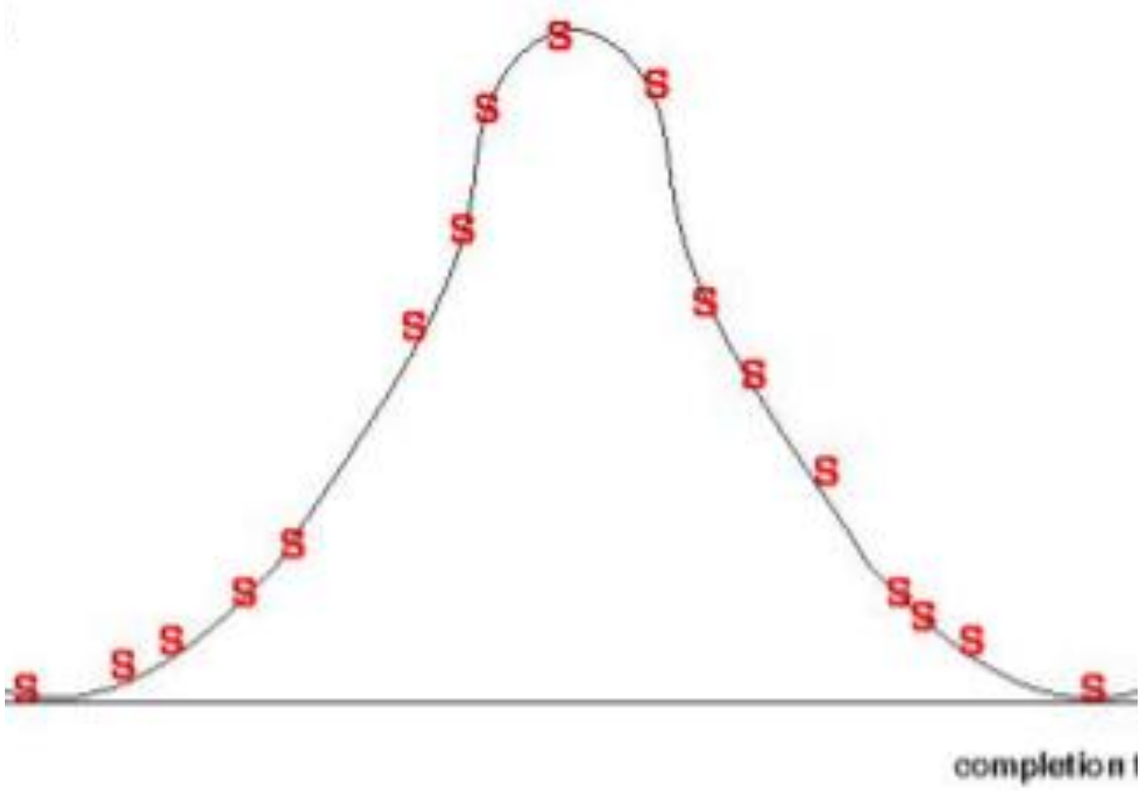
Ovdje je ključno bilo uočiti da za osiguravanje dobre pokrivenosti realne Pareto fronte, fitness funkcija također treba nagrađivati najbrže/najjeftinije vrijeme izvođenja.

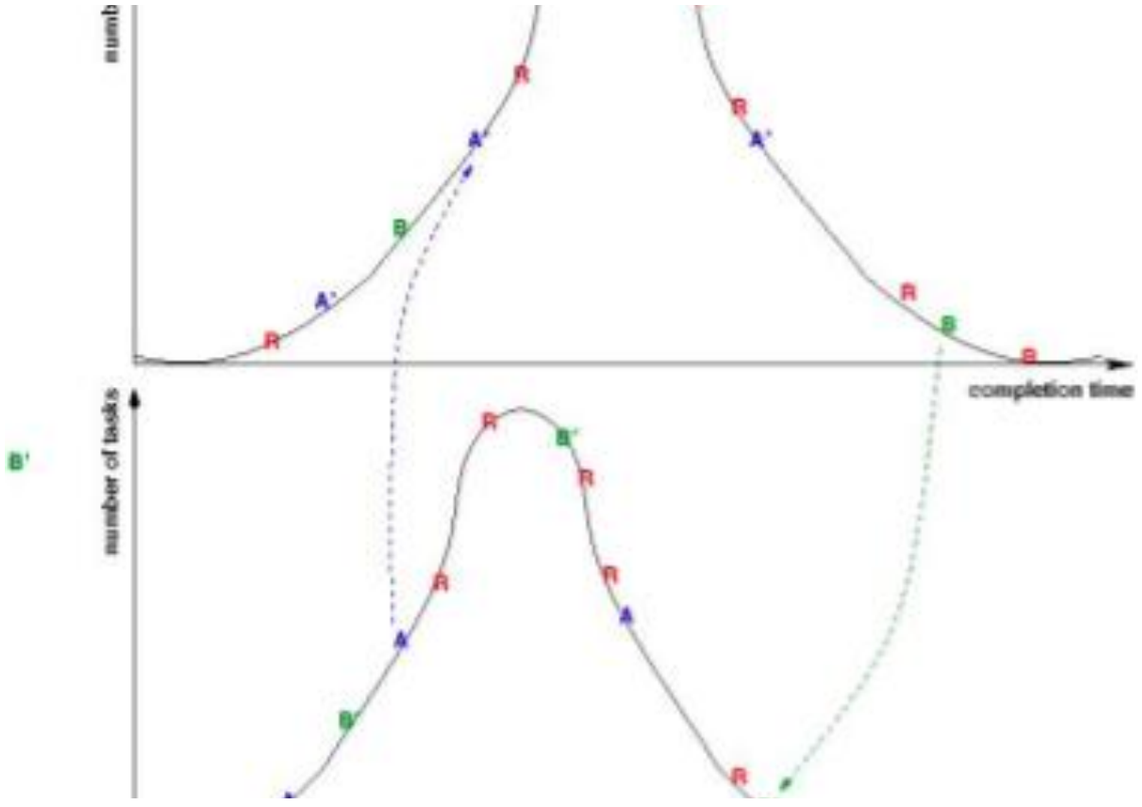
Zadnja faza izračuna

U zadnjoj fazi izračuna, treba istražiti stalno korištenu pretpostavku da je vrijeme izračuna "fluidno" [10]. Na ovom mjestu, po definiciji, promatramo slučaj kad aplikacija sadrži premalo zadataka da bi ta pretpostavka bila ispunjena. Analizirali smo nekoliko pristupa ovom problemu, usredotočivši se na bolju procjenu krajnih potreba aplikacije za računanjem, tako da alokacija krajnjih zadataka na mašine bude optimalna. Naši pristupi su se kretali od potpunog znanja o preostalim vremenima izvođenja do potpunog neznanja (slučajna vremena izvođenja). Razlike među pristupima bile su beznačajne. Stoga je problemu trebalo pristupiti u nekoj drugoj fazi, točnije u fazi procjene troškova i vremena izvođenja. Da bi to postigao, BaTS prati procijenjene neiskorištene krajnje vremenske intervale. BaTS određuje ograde kako bi uzeo u obzir "outliere" čije vrijeme izvođenja je izvan krajnjih predviđenih vremenskih intervala te dodaje virtualne resurse i/ili vrijeme u planiranje. Uprkos tome, "outlieri" još uvijek mogu dovesti do prekršaja.

Korištenje BaTS metodologije u obrazovanju

U kurikulumu studija Master of Computer Science na Sveučilištu u Amsterdamu, kolegij "Web services and cloud-based systems" se predaje već nekoliko godina. Jedan od važnijih ciljeva ovog kolegija je da se studenti upoznaju s izazovima sustava baziranih na oblaku. Praktični dio ovog kolegija sastoji se od razvoja rudimentarnog planera za virtualne resurse i analiziranja njegovog ponašanja na sveučilišnom u usporedbi s komercijalnim oblakom. Sveučilišni sustav sastoji se od OpenNebula [5] implementiran na DAS-u [6], Nizozemski nacionalni računalni klaster.



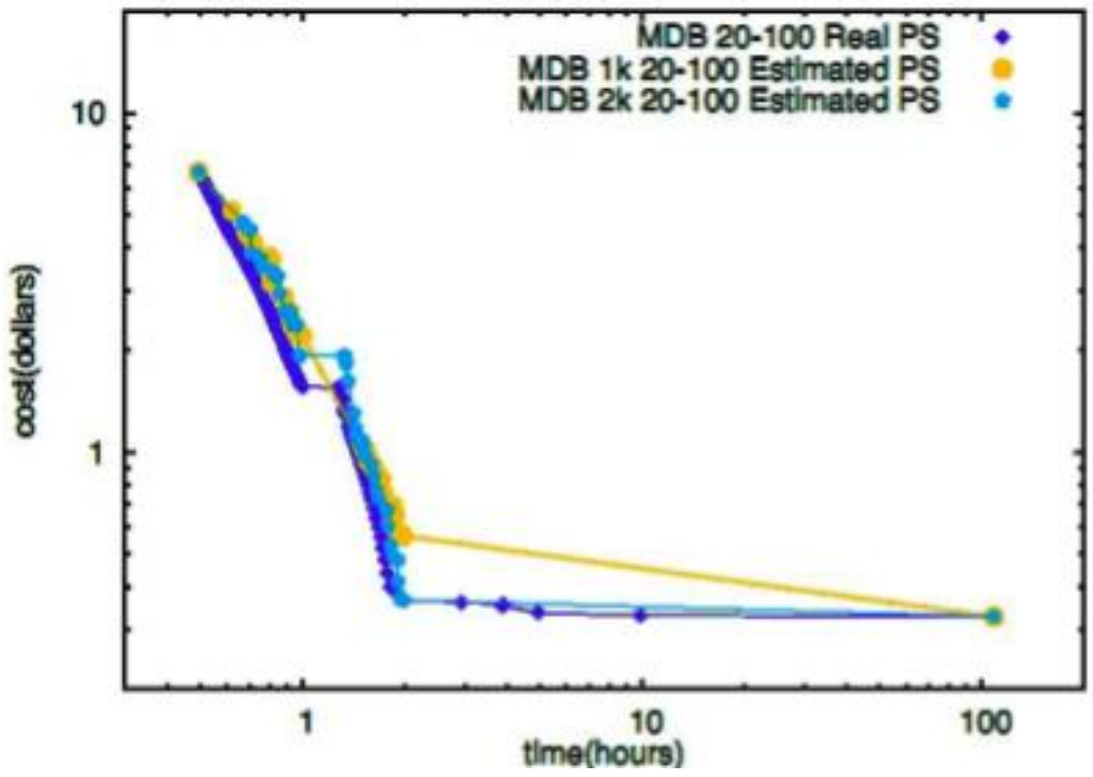


OpenNebula je “open source” rješenje za Infrastructure-as-a-Service implementacije: fizički resursi su upravljani i nude se kao virtualni resursi. Studenti ovdje imaju gornju granicu na broj virtualnih mašina koje se mogu konkurentno koristiti.

Komercijalni oblak se sastoji od Amazon EC2 [7] ponuda oblaka. Ovdje pak studenti dobije budžet koji mogu upotrijebiti za bilo kakvu nabavu resursa vezanu uz laboratorij.

Ocjena njihovog laboratorijskog rada uzima u obzir svako probijanje budžeta. Općenito, rezultati laboratorijskog rada su pokazali da studenti mogu razumijeti razliku između “best-effort” i komercijalne nabave virtualnih resursa.

Obično su razvijali planere bazirane na profitabilnosti, ali najbolji studenti su razvijali složenije planere u kojima su korisnici mogli odabrati od nekoliko politika: najbrži, najjeftiniji, najprofitabilniji.



ZAKLJUČAK I BUDUĆI RAD

Stohastički pristupi korisnički orijentiranom planiranju oblaka su obećavajući. Uklapanje istraživanja u obrazovanje, čim dosegne neko stabilnije stanje, je jako važno.

Kao budući rad htjeli bismo podržati Haskell AWS Lambda funkcije izvedene kroz implementaciju Haskell AWS API-ja.

Literatura

[1] Newman, Sam. Building Microservices. O'Reilly Media, Inc., 2015.

[2] Fowler, M., Lewis, J.: Microservices. http://martinfowler.com/articles/microservices.html (March 2014), Last accessed: 15-08-2018

[3] Balalaie, Armin, Abbas Heydarnoori, and Pooyan Jamshidi. "Migrating to cloud-native architectures using microservices: An experience report." arXiv preprint arXiv: 1507.08217 (2015).

[4] AM Oprescu. Stochastic Approaches to Self-Adaptive Application Execution on Clouds. PhD Thesis, Amsterdam, Vrije Universiteit, 2013.

[5] https://opennebula.org/, Last accessed: 15-11-2018.

[6] https://www.cs.vu.nl/das5/, Last accessed: 15-11-2018.

[7] https://console.aws.amazon.com/ec2/v2/home, Last accessed: 15-11-2018.

[8] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html, Last accessed: 15-11-2018.

[9] A.-M. Oprescu; T. Kielmann; H. Leahu, Budget estimation and control for bag-of-tasks scheduling in clouds, 2011, Parallel Processing Letters, vol. 21.

[10] A.-M. Oprescu; T. Kielmann; H. Leahu, Stochastic tail-phase optimization for bag-of-tasks execution in clouds, 2012, Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing.

[11] A.-M. Oprescu; T. Kielmann; Bag-of-tasks scheduling under budget constraints,2010, IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom).

[12]. A. Vintila; A.-M. Oprescu; T. Kielmann; Fast (re-) configuration of mixed on-demand and spot instance pools for high-throughput computing, 2013, Proceedings of the first ACM workshop on Optimization techniques for resources management in clouds.

OBRAZOVANJE USMJERENO PREMA MJERENJIMA ENERGETSKE EFIKASNOSTI TOKOM TESTIRANJA SOFTVERA

Misija nastavnika softverskog inženjerstva je priprema budućih softverskih inženjera koji mogu ovladati svakim problemom tokom čitavog životnog ciklusa softverskog proizvoda. Osim vještina vezanih uz razumijevanje potreba dionika te uz sastavljanje odgovarajućeg radnog softvera, važnu ulogu također igra sposobnost provjere točnosti rezultata.

U ovom članku, usredotočeni smo na taj posljednji skup vještina. Usredotočeni smo na testiranje, pri čemu je razina automatizacije manje važna. Stavljamo naglasak na mjerenje prirode testiranja, točnije, usredotočeni smo na mjerenje potrošnje energije softvera koje se može dobiti tokom testiranja na različitim razinama. Svi ovi aspekti su prezentirani sa stanovišta nastavnika softverskog inženjerstva, s ciljem da se izloži kako postaviti laboratorijske vježbe iz softverskog inženjerstva koje su

usredotočene na mjerenje energetske efikasnosti tokom testiranja softvera uz autorove komentare tog prijedloga.

DEFINICIJA PROBLEMA

Jedan od izazova proizvođača baterija je koliko dugo baterija može raditi bez stalnog izvora struje, ali naravno postoje i mnogi drugi izazovi poput veličine koja jako utječe na veličinu uređaja, a još jedan faktor koji čini važno svojstvo baterije je njena težina. Baterija se smatra nešto lakšom u usporedbi s uređajem koji ju koristi da bi radio. Izazov je kako proizvesti bateriju manje veličine i manje težine te svakako veće efikasnosti u terminima vremena koje mobilni uređaj može raditi bez punjenja.

Povrh tih hardverskih izazova, postoji njihov brat – softverski izazov, točnije, da sam softver treba podržavati uštedu energije. Ostvarivanje tog izazova bez smanjenja korisničkog iskustva se u današnje vrijeme smatra tihim, ali važnim, ciljem svakog razvoja softvera koji cilja bilo koju vrstu mobilnih uređaja.

Uzimajući u obzir da je potrošnja energije svakog mobilnog uređaja ovisna počevši od otvorenih aplikacija, preko pristupne razine osnovnih servisa, do korisnikovog raspoloženja, već i razvoj softvera za takve uređaje je izazov sam za sebe [1]. Može se reći da je izazov razvoja softvera uvijek isti, ali moramo istaknuti “mobilnost” kao ključno svojstvo sustava. Status punosti baterije isto određuje performanse sustava kroz razine konfiguracije operativnog sustava – poznatom kao “energy saving preferences”.

Još je teže ako netko (u našem slučaju nastavnik) mora pripremiti studente za takve izazove. Sve poznate “dobre prakse” i “savjeti za štednju energije” moraju biti prezentirani u kontekstu koji je lagano razumljiv studentima.

To se može postići kroz postavljanje koncepata u poznatu okolinu, kao što je testiranje softvera i automatizacija testiranja [2], u našem slučaju. To je naš cilj u ovom članku, to je sadržaj narednih poglavlja, koji započinje s prijedlogom, slijedi evaluaija i završava s dodatnim savjetima za poboljšanja.

PRIJEDLOG RJEŠENJA

Kao što je rečeno ranije, moramo naći najpogodniju okolinu za uvođenje mjerenja potrošnje energije [3] i praksi za evaluaciju energetske efikasnosti.

To može biti inicijalni razvoj softvera, ali i softver u evoluciji, obzirom da obje faze razvoja u općenitom životnom ciklusu softvera nude mogućnosti za mjerenje proizvoda koji se razvija/evoluira [4].

Prednost odabira inicijalnog razvoja softvera je da se sve aktivnosti mogu fokusirati na probleme vezane uz uštedu energije, dok odabir softvera u evoluciji nudi mogućnost evaluacije poboljšanja u implementaciji proizvoda. S druge strane, softver u evoluciji zahtjeva postojanje radnog softvera na početku, dok kod inicijalnog softvera razvoj je proces koji kreira proizvod počevši od prvih zahtjeva na softver.

Stavljajući oba moguća pristupa u nastavnu okolinu, najbolja opcija je koristiti oba. Jedan semestar za inicijalni razvoj softvera, a drugi za evoluciju tog istog softvera. Obično nastavnik nema dva semestra u nizu da prezentira sadržaj na način predložen u prethodnoj rečenici. To je razlog zašto ipak moramo odlučiti koji način razvoja koristiti za uvođenje odabranih praksi. Sa stanovišta arhitekture procesa razvoja, ali i činjenice da inicijalni razvoj isto može biti evolucijski, odlučili smo se za softver u evoluciji. On uključuje mnoge aktivnosti inicijalnog razvoja (osim ranih prikupljanja zahtjeva i analize) te naglašava važnost testiranja i evaluacije.

Ovaj izbor omogućuje nastavniku da:

– Sa studentima prođe kroz njihovu povijest razvoja softvera (prošli projekti) da budu kritični prema sebi.

– Studenti evaluiraju svoje rezultate koristeći kodne metrike i mjerenja (ili procjene) energetske potrošnje.

– Studenti integriraju gornje aktivnosti u standardne verifikacijske i validacijske procese evolucije softvera.

Programski jezik razvoja nije važan, tako da student može izabrati bilo koji od svojih prethodnih projekata za evoluciju – ili sve njih, ako su različitih tipova ili programskog jezika. Međutim, programski jezik obično određuje ili ograničava razvojnu okolinu i alate koji se koriste. Odabir tih alata i njihovih plug-inova također nudi dobru podlogu za evaluaciju kodnih metrika.

Mjerenja potrošnje energije i energetske efikasnosti obično zahtjeva drugačiji alat, obzirom da do danas postoji samo nekoliko razvojnih okolina koje uključuju mjerenja potrošnje energije. Što se tiče testiranja kao osnove za mjerenja, moramo napomenuti da je statička analiza koda isto dio testiranja softvera. Osim toga, odabrani dijelovi aplikacijske kodne baze se izvršavaju tokom “white box” testiranja (uglavnom unit test), koja uz malo proširenje mjerenja potrošnje energije može predstavljati energetsku potrošnju testnog slučaja – posredan pogled na potrošnju energije testiranog koda. Tokom “black box” testiranja, čitava aplikacija se testira koristeći testne scenarije. Takvi testni scenariji su većim dijelom usporedivi s predviđenim svakodnevnim korisničkim slučajevima softvera, dok ostali predstavljaju granične scenarije – uključujući one kod kojih je korisnik loše volje. Mjerenje potrošnje energije izvođenja tih “black box” testova daje približan (ali neposredan) pogled na potrošnju energije proizvoda.

U tome je glavna prednost evolucije (u usporedbi s inicijalnim razvojem softvera). Nastavnik može pripremiti početnu verziju za evoluciju, uključujući kod koji može biti poboljšan, popis poznatih defekata i testnu bazu! Postojanje testa za ponovno testiranje ili regresijsko testiranje je jako važno jer produktivnost evolucije može biti povećana kroz to svojstvo.

Uz pretpostavku da su svi prethodni koraci napravljeni, integracija mjerenja energetske efikasnosti u proces testiranja iz studentske perspektive izgledaju kao sljedeće aktivnosti:

1. Odaberi produkt koji može biti tvoj prošli projekt ili projekt iz repozitorija.

2. Evaluiraj ga koristeći statičku analizu koda, testiranje, mjerenja energije, pregled koristivosti, itd.

3. Poboljšaj ga (različiti načini evolucije kao što su dodavanje/promjena funkcionalnosti, popravak ili prilagodba)

4. Ponovno testiraj da budeš siguran da si eliminirao defekt ili pogrešku

5. Regresijski testiraj (uključujući ponovnu evaluaciju)

6. Zaključi rezultate (daj konačnu presudu obzirom na sve dobivene podatke, uključujući energetsku efikasnost).

DISKUSIJA

Budući da je mjerenje potrošnje energije relativno novo u usporedbi s drugim tehnikama, kao što su statička analiza koda, “black/white box” testiranje i ispravljanje pogrešaka, ono može biti točka za neuspjeh. Ali ako se kombinira s ovim starijim principima i tvori s njima zajedničku ocjenu za svakog studenta, kritičnost je puno manja.

Gledajući na mogućnosti ocjenjivanja, možemo naći razne “stupnjeve slobode”, koji se mogu koristiti odvojeno ili kao dio ukupne ocjene:

1. broj različitih projekata,

2. broj primijenjenih/korištenih programskih jezika,

3. kvaliteta koda konačnog proizvoda,

4. energetska efikasnost konačnog proizvoda,

5. poboljšanje kvalitete koda tokom evolucije softvera,

6. poboljšanje energetske učinkovitosti tokom evolucije softvera.

Promatrajući sve “stupnjeve slobode” izvršavanja zadatka, puno natjecanja se može definirati za natjecateljski orijentirane studente, dok će studenti orijentirani na izvršavanje zadataka sakupiti “male pobjede” u više projekata, cilj perfekcionista će biti optimizirati sve kodne metrike i minimizirati potrošnju energije. Za prosječnog studenta, poboljšanje energetske efikasnosti i razumljivosti koda mogu biti dostižni ciljevi.

Studentsko natjecanje može biti još više pospješeno ako se studenti ne vraćaju svojim starim projektima, nego im se dozvoli da biraju iz određenog repozitorija.

Kao i kod komjuterskih igara, svi nivoi igre su jednako dostupni svima.

Da bi se podržala jednakost, može se još kreirati zajednički javni forum za studente i nastavnike.

Naš budući rad u ovom području će se fokusirati na konfiguriranje prenosive integrirane okoline za razvoj, testiranje i procjenu energetske potrošnje.

Ta okolina će biti korištena u okviru evolucije softvera ili inicijalnog razvoja kao potpora obrazovanju o mjerenjima energetske efikasnosti tokom testiranja softvera. Moguće je da to ograniči studentsku kreativnost nudeći poluzatvoreni alat, budući da hardver igra jako važnu ulogu u trenutnoj arhitekturi procjene potrošnje energije. Postoje istraživanja čiji je cilj uklanjanje ovog ograničenja – ne možemo dočekati rezultate tih istraživanja da i njih integriramo.

Literatura

[1] J. Saraiva, M. Couto, Cs. Szabo, D. Novak: Towards Energy-Aware Coding Practices for Android, Acta Electrotechnica et Informatica, Vol. 18, No. 1, 2018, pp. 19–25. https://doi.org/10.15546/aeei-2018-0003

[2] D. Li, Y. Jin, C. Sahin, J. Clause, W. G. J. Halfond: Integrated energy-directed test suite optimization, in Proc. of the 2014 International Symposium on Software Testing and Analysis, ISSTA 2014, ACM, 2014, pp. 339–350.

[3] M. Santos, J. Saraiva, Z. Porkolab, D. Krupp: Energy Consumption Measurement of C/C++ Programs Using Clang Tooling, in Proceedings of the SQAMIA 2017:6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Z. Budimac, ed., Belgrade, Serbia, 11-13.9.2017, Paper No. 15, 8 pages, also published online by CEUR Workshop Proceedings No. 1938 ISSN 1613-0073.

[4] Cs. Szabo, J. Saraiva: Focusing software engineering education on green application development, in Conference of Information Technology and Development of Education – ITRO 2017, Novi Sad, Serbia, pp. 165–169, ISBN 978-86-7672-302-7.

PREMA INŽENJERSKOJ DISCIPLINI ZA ZELENI SOFTVER

Ovaj tehnički izvještaj opisuje istraživanje u Green Software Laboratory na Coimbra i Minho sveučilištima, koje je prezentirano na prvom sastanku za trening nastavnika u sklopu Erasmus+ projekta “Focusing Education on Composability, Comprehensibility and Correctness of Working Software”. On obuhvaća zeleno rangiranje za programske jezike i strukture podataka, kao i tehnike za lociranje nenormalnih korištenja energije u softveru.

MOTIVACIJA

Današnja široko raširena upotreba bežičnih, ali moćnih, računalnih uređaja, kao što su pametni telefoni, prijenosna računala, itd., mijenja način na koji proizvođači računala i softverski inženjeri razvijaju svoje proizvode. U stvari, vrijeme izvođenja računala/softvera, što je bio primarni cilj u prolom stoljeću, nije više jedina briga. Potrošnja energije postaje sve uže usko grlo za hardverske i softverske sustave. Kao posljedica, istraživanje o zelenom softveru je relevantno i aktivno područje istraživanja. Ovaj izvještaj ukratko opisuje istraživanje o zelenom softveru koje se odvija u Green Software Laboratory (GSL). GSL se sastoji od više portugalskih istraživačkih grupa, uključujući i dvije uključene u projekt “Focusing Education on Composability, Comprehensibility and Correctness of Working Software”. GSL je inicijativa za razvoj tehnika i alata s ciljem smanjenja potrošnje energije za različite računalne sustave (mobilne, programe, baze podataka, itd.).

GSL se posebno usredotočuje na softversku stranu, gdje primjenjuje analizu (izvornog koda) i tehnike transformacije da detektira anomalije u potrošnji energije te da definira optimizacije koje smanjuju tu potrošnju.

U prošlom stoljeću efikasnost softverskog sustava je uglavnom bila fokusirana na vrijeme izvođenja i potrošnju memorije. U današnje vrijeme programeri u razvoju softvera često postavljaju pitanje “je li brži program također i zeleniji program?”. Ima mnogo aspekata softverskog sustava koji utječu na njegove energetske performanse: programski jezik i njegov model izvođenja (komajliran u binarni kod ili na virtualnu mašinu, interpretirani kod, lijena nasuprot stroge evaluacije, korištenje parcijalne evaluacije tokom izvođenja, itd.). Efikasnost memorijskog modela i knjižnica jezika također utječu na performanse. Kompleksnost algoritma, korištenog za implementaciju datog računalnog problema, također utječe na performanse: ako implementirani algoritam mora odraditi više posla nego što je strogo potrebno, onda će više CPU i energije biti upotrebljeno.

U ovom dokumentu ukratko izvještavamo o rezultatima istraživanja ostvarenim u, točnije, u analizi energetske efikasnosti programskih jezika (poglavlje 2), knjižnica struktura podataka (poglavlje 3), i softverskog izvornog koda (poglavlje 4).

ZELENO U PROGRAM-SKIM JEZICIMA

Zanimljivo pitanje koje se javlja pri diskusiji o energiji u programskim jezicima jest je li brži programski jezik ujedno energetski efikasan jezik ili nije. Uspoređivanje programskih jezika je, međutim, izuzetno težak zadatak, zbog toga što su performanse jezika ovisne o kvaliteti njegovog kompajlera, virtualne mašine, sakupljača smeća, itd. U Green Software Laboratory-ju smo istražili, ocijenili i usporedili performanse (ukupno) 27 najraširenijih programskih jezika. Koristili smo dva različita repozitorija problema iz računarstva: Computer Language Benchmark Game (CLBG)3 i Rosetta Code4 repozitorije [1–3]. Oba repozitorija definiraju skup zadataka za računalo i daju implementacije rješenja u velikoj grupi programskih jezika. Dok je CLBG bio napravljen za analizu vremena izvođenja jezika, Rosetta Code je bio definiran iz razloga razumijevanja programa.

Kompajlirali/izveli smo takve programe koristeći najnovije kompajlere, virtualne mašine, interpretere i knjižnice za svaki jezik. Zatim smo promatrali vrijeme izvođenja, najveću i ukupnu potrošnju memorije, te CPU/- DRAM/GPU potrošnju energije. Rezultat je bilo energetsko rangiranje 27 jezika, s time da smo analizirali te rezultate obzirom na korišteni tip izvođenja jezika (kompajliranje, virtualna mašina i interpretiranje) i programsku paradigmu (imperativni, funkcijski, objektno orijentirani, skriptirani). Za svaki tip izvođenja i programsku paradigmu, kombinirali smo rangiranje programskog jezika sa svakim svojstvom posebno (npr., potrošnja vremena ili energije). Naši prvi eksperimenti su pokazali očekivane rezultate, kao što je to da je jezik C istovremeno brži i zeleniji, ali isto tako da neki sporiji jezici su više energetski efikasniji nego drugi [2, 3].

ZELENO U STRUKTURAMA PODATAKA

Programski jezik/paradigma, kao i njihove moćne kompajlerske optimizacije, nisu jedini aspekt koji utječe na potrošnju energije softverskog sustava. U stvari, program može postati efikasniji ako se “samo” optimiziraju njegove knjižnice [4,5]. Većina jezika nudi moćne knjižnice za manipulaciju strukturama podataka. U GSL-u smo istražili energetske performanse dviju naprednih struktura podataka široko korištenih u programskim jezicima Java i Haskell.

U Javi smo proveli detaljnu studiju o potrošnji energije knjižnice Java Collections Framework (JCF) 5. Promatrali smo tri uobičajene grupe struktura podataka, točnije, Sets, Lists, i Maps, te za svaku od tih grupa, istražili potrošnju energije svake od njihovoh različitih implementacija i metoda [4]. Ova energetska svjesnost JCF-a može biti korištena ne samo za osvještavanje programera softvera da razvijaju zeleniji Java softver, nego i za optimizaciju postojećeg Java koda. Razvili smo alat za refaktoring struktura podataka u Javi, nazvan jStanley, koji refaktorira Java izvorni kod kada je zelenija kolekcija dostupna [6]. Proveli smo inicijalnu evaluaciju na 7 javno dostupnih Java projekata, kojom smo uspjeli smanjiti potrošnju energije između 2% i 17%.

U programskom jeziku Haskell, istražili smo potrošnju energije knjižnice Edison6, potpuno zrele i dobro dokumentirane knjižnice čisto funkcijskih struktura podataka [7]. Edison pruža različite funkcijske strukture podataka za implementiranje tri tipa apstrakcije: nizove (liste, redove i stogove), kolekcije (skupove i hrpe) i povezane kolekcije (preslikavanja i konačne relacije). Analizirali smo 16 implementacija takvih struktura podataka te mjerili detaljne energetske i vremenstke metrike [5]. Nadalje, istražili smo utjecaj na potrošnju energije različitih kompajlerskih optimizacija. Zaključili smo da je potrošnja energije direktno proporcionalna vremenu izvođenja te da potrošnja energije DRAM-a predstavlja između 15 i 31% ukupne potrošnje energije. Na kraju smo zaključili da optimizacija može imati i pozitivan i negativan utjecaj na potrošnju energije.

ZELENO U IZVORNOM KODU

Ne utječu samo jezici i strukture podataka na potrošnju energije, već i algoritmi i programerske prakse igraju u tome ključnu ulogu. U GSL-u smo prilagodili dobro poznatu tehniku za pronalaženje pogrešaka, kako bismo statički pronašli “energetske rupe” (viđene kao energetske neefikasnosti, te stoga, energetske pogreške) u izvornom kodu aplikacija [8–11]. Definirali smo SPELL - SPectrum-based Energy Leak Localization za određivanje crvenih (energetski neefikasnih) dijelova softvera. Prva eksperimentalna studija pokazuje da su stručnjaci u programiranju, s pristupom otkrivenim energetskim rupama pomoću SPELL-a, mogli bolje optimizirati potrošnju energije svojih programa (između 15% i 74%), nego stručnjaci koji nisu imali te informacije ili su informacije dobivali standardnim programskim profilerom tokom ozvođenja. Istražili smo također energetsko ponašanje C/C++ programa [12].

Široka upotreba bežičnih uređaja te pojava Internet-of-Things mijenja način na koji softverski inženjeri razvijaju softver. Softver se mora izvoditi na raznim mobilnim uređajima i potrošnja energije je glavna briga pri razvoju softvera. Software Product Lines (SPL) su se pojavile kao važna disciplina softverskog inženjerstva, koja omogućuje razvoj softvera koji dijeli zajednički skup svojstava. U GSL-u smo definirali tehnike statičke analize za rezoniranje o potrošnji energije u SPL-ovima na osnovi uvjetnog sastavljanja.

Takve tehnike omogućuju programerima softvera da tokom razvoja identificiraju (ne)zelene proizvode i/ili svojstva u SPL-u [13].

Android je široko korišten ekosustav za bežične uređaje, te je energetska analiza softvera i optimizacija aktivno područje istraživanja. GSL tim je razvio više tehnika [14,15] i alata za analizu i optimizaciju potrošnje energije u izvornom kodu Android aplikacija [16, 17].

U današnje vrijeme, većina podataka spremljenih u našim mobilnim uređajima (datoteke, fotografije, videi) su također spremljeni u oblaku kojeg pruža ekosustav operativnog sustava uređaja. Takvi sustavi u oblaku su podatkovni centri koji dnevno provode veliku količinu procesa za pretragu podataka, nadziranih i kontroliranih pomoću vrlo složenih sustava za upravljanje bazama podataka, koji su odgovorni za uspostavu efikasnih planova pretraga. Sustavi baza podataka obično se pouzdaju u planove koji optimiziraju vrijeme odziva. Mi smo dizajnirali i razvili alternativnu metodu za definiranje planova pretraga baza podataka na osnovi potrošnje energije [18, 19]. Naši prvi eksperimentalni rezultati pokazuju da korištenje optimizacijske heuristike donosi značajne dobitke, u smislu potrošnje energije, ali i vremena potrbnog za izvođenje pretraga.

ZAKLJUČCI

Ovaj tehnički izvještaj opisuje istraživanje u Green Software Laboratory, točnije, zeleno rangiranje progrmaskih jezika i struktura podataka, tehnike pronalaska energetskih neefikasnosti u izvornom kodu sotverskih sustava te energetski svjesan plan pretrage za sustave baza podataka.

Literatura

[1] Couto, M., Pereira, R., Ribeiro, F., Rua, R., Saraiva, J.: Towards a green ranking for programming languages. In: Proceedings of the 21st Brazilian Symposium on Programming Languages. SBLP (2017) 7:1–7:8 (best paper award).

[2] Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P., Saraiva, J.: Energy efficiency across programming languages: How do energy, time, and memory relate? In: Proc. of the 10th ACM SIGPLAN Int. Conference on Software Language Engineering. SLE 2017, New York, NY, USA, ACM (2017) 256–267

[3] Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P., Saraiva, J.: Ranking programming languages by energy efficiency. Science of Computer Programming (2018) Submitted.
[4] Pereira, R., Couto, M., Saraiva, J., Cunha, J., Fernandes, J.P.: The Influence of the Java Collection Framework on Overall Energy Consumption. In: 5th Int. Workshop on Green and Sustainable Software. GREENS '16, ACM (2016) 15–21
[5] Melfe, G., Fonseca, A., Fernandes, J.P.: Helping developers write energy efficient haskell through a data-structure evaluation. In: Proceedings of the 6th International Workshop on Green and Sustainable Software. GREENS '18, New York, NY, USA, ACM (2018) 9–15
[6] Pereira, R., Sima˜o, P., Cunha, J., Saraiva, J.: jStanley: Placing a Green Thumb on Java Collections. In: 33rd ACM/ IEEE International Conference on Automated Software Engineering. ASE 2018, New York, NY, USA, ACM (2018) 856–859
[7] Lima, L.G., Melfe, G., Soares-Neto, F., Lieuthier, P., Fernandes, J.P., Castor, F.: Haskell in Green Land: Analyzing the Energy Behavior of a Purely Functional Language. In: Proc. of the 23rd IEEE Int. Conf. on Software Analysis, Evolution, and Reengineering (SANER'2016), IEEE (2016) 517–528
[8] Pereira, R., Carcao, T., Couto, M., Cunha, J., Fernandes, J.P., Saraiva, J.: Helping programmers improve the energy efficiency of source code. In: Proc. of the 39th Int. Conf. on Soft. Eng. Companion, ACM (2017)
[9] Pereira, R.: Locating energy hotspots in source code. In: Proceedings of the 39th International Conference on Software Engineering Companion. ICSE-C '17, Piscataway, NJ, USA, IEEE Press (2017) 88–90 (ACM SRC silver award).
[10] Pereira, R.: Energyware Engineering: Techniques and Tools for Green Software Development. PhD thesis, Depart. de Informatica, Universidade do Minho (2018)
[11] Pereira, R., Carc¸a˜o, T., Couto, M., Cunha, J., Fernandes, J.P., Saraiva, J.: Spelling out energy leaks: Aiding developers locate energy inefficient code. (2018) (submitted).
[12] Santos, M., Saraiva, J., Porkolab, Z., Krupp, D.: Energy consumption measurement of c/c++ programs using clang tooling. SQAMIA'17 - CEUR Workshop Proceedings 1938 (2017)
[13] Couto, M., Borba, P., Cunha, J., Fernandes, J.P., Pereira, R., Saraiva, J.: Products go green: Worst-case energy consumption in software product lines. In: Proceedings of the 21st International Systems and Software Product Line Conference - Volume A. SPLC '17, ACM (2017) 84–93

[14] Couto, M., Carcao, T., Cunha, J., Fernandes, J.P., Saraiva, J.: Detecting anomalous energy consumption in android applications. In Quinta˜o Pereira, F.M., ed.: Programming Languages: 18th Brazilian Symposium, SBLP 2014, Maceio, Brazil, October 2-3, 2014. Proceedings. (2014) 77–91
[15] Cruz, L., Abreu, R.: Performance-based guidelines for energy efficient mobile ap- plications. In: 4th International Conference on Mobile Software Engineering and Systems. MOBILESoft '17, Piscataway, NJ, USA, IEEE Press (2017) 46–57
[16] Couto, M., Cunha, J., Fernandes, J.P., Pereira, R., Saraiva, J.: Greendroid: A tool for analysing power consumption in the android ecosystem. In: 2015 IEEE 13th International Scientific Conference on Informatics. (Nov 2015) 73–78
[17] Cruz, L., Abreu, R., Rouvignac, J.N.: Leafactor: Improving energy efficiency of android apps via automatic refactoring. In: IEEE/ACM International Conference on Mobile Software Engineering and Systems, MobileSoft 2017. (2017)
[18] Goncalves, R., Saraiva, J., Belo, O.: Defining energy consumption plans for data querying processes. In: 2014 IEEE International Conference on Big Data and Cloud Computing (BdCloud)(BDCLOUD). Volume 00. (Dec. 2015) 641–647
[19] Belo, O., Goncalves, R., Saraiva, J.: Establishing energy consumption plans for green star-queries in data warehousing systems. In: 2015 IEEE International Conference on Data Science and Data Intensive Systems. (Dec 2015) 226–231

PODUČAVANJE PROGRAMIRANJA ORIJENTIRANOG ZADACIMA

Na Composability, Comprehensibility, Correctness zimskoj školi bit će dva predavanja o Task Oriented Programming (Programiranje orijentirano zadacima) i konkretnim sustavima zasnovanim na toj paradigmi. Sustav iTask nudi web-bazirano sučelje za ljude da vide svoje zadatke i dijele svoj napredak u tim zadacima. Sustav mTask primjenjuje iste koncepte za specifikaciju zadataka koje izvode mikroprocesori. U ovom radu, obrazlažemo odluke donesene na treningu nastavnika u Amsterdamu o tome što i kako će ove teme biti prezentirane na zimskoj školi.

Zbog ograničenog vremena i različite pozadine publike, usredetočit ćemo se na praktičnu primjenu te paradigme. Jedva preostaje vremena za opis izazova i ljepote konstrukcije tih sustava.

UVOD

Task Oriented Programming (Programiranje orijentirano zadacima), TOP, je stil programiranja koncentriran oko koncepta zadatka kojeg provode ljudi i mašine. Ti zadaci su specificirani običnim funkcijama u funkcijskom programskom jeziku. U svim ostalim primjerima koristit ćemo Clean [6]. Semantika zadataka je dosta drugačija od obične evaluacije funkcija. Zadatak se evaluira opet i ponovo, sve dok ne postigne stabilnu vrijednost ili njegov rezultat više nije potreban. Rezultati međuzadataka mogu biti vidljivi drugim zadacima. Zadaci se mogu spajati pomoću kombinatora zadataka. Na zimskoj školi Composability, Comprehensibility, Correctness u Košicama, bit će dvije serije predavanja o TOP-u. To su predavanja nazvana Why "Task Oriented Programming" matters i Functional Programming of Devices. Obje serije će se sastojati od predavanja i praktičnog rada polaznika. U ovom članku, motiviramo odluke o sadržaju i organizaciji tih serija nakon diskusije na treningu nastavnika.

AUDIENCE

Zimska škola Composability, Comprehensibility, Correctness je za BSc, MSc, PhD studente kao i njihove nastavnike. Diskusije na treningu nastavnika su razotkrile da iskustvo u funkcijskom programiranju je raznoliko, kako u količini iskustva, tako i u progrmaskom jeziku. Korišteni jezici se kreću od čistih i lijenih jezika kao što su Haskell i Clean do jezika Erlang, Scheme i Scala.

To implicira da ne možemo pretpostaviti dobro zajedničko iskustvo u funkcijskom programiranju. Samo dio publike će biti upoznat s konceptima poput strong typing, higher order functions, type constructor classes, Monads i generics. Iako su ove teme gradivni blokovi TOP-a, ne možemo pretpostaviti da su poznati svim sudionicima.

PROGRAMIRANJE ORIJENTIRANO ZADACIMA

Task Oriented Programming (Programiranje orijentirano zadacima), TOP, se zasniva na malom broju primitivnih zadataka za određenu domenu. Ti primitivni zadaci obično imaju interakciju s okolinom, npr. ljudi izvode dio zadatke ili hardver ima interakciju s fizičkim svijetom. Kombinatori zadataka se koriste za sastavljanje zadatka od manjih zadataka. Zadaci mogu komunicirati kroz svoje rezultate, kao i kroz Shared Data Sources, SDSs. Takav SDS sadrži tipizirane podatke kojima se može pristupiti pomoću primitiva kao get i set. Ti primitivi djeluju na stanje zadatka da bi se osigurala referencijska transparentnost.

Kako bi se iskoristili tipovi podataka i računi postojećeg jezika, TOP sustav je često napravljen kao Domain Specific Language, DSL, uložen u postojeći (funkcijski) programski jezik.

Sustav iTask je bio prva implementacija TOP-a [5]. On je DSL uložen u funkcijski programski jezik Clean. Sustav iTask omogućuje interakciju s ljudskim radnikom pomoću tipski vođenim generiranjem web-stranica. Te stranice se prikazuju u jednom od postojećih preglednika. Stranica pruža informacije o trenutnim zadacima za korisnika. Korisnik može djelovati na sustav iTask popunjavajući formulare ili pritiskom na dugmad.

KORIŠTENE TEHNIKE

Sustav iTask prosljeđuje stanja vrlo slično monadi stanja. Operatori za return, bind (>>=), i sequence (>>|) su vrlo slični dobro poznatim monadskim verzijama [4,7]. To zahtjeva funkcije višeg reda i korisnički definirane infix operatore. Kako bi se koristio simbol operatora za različite Monade, oni su definirani kao tip konstruktorske klase.

Kombinatori zadataka su svi funkcije višeg reda, najčešće korisnički definirani infix operatori, koje manipuliraju rezultate zadatka i globalno stanje zadatka. Posebnost TOP-a je da zadaci produciraju međurezultate, koje je moguće očitati, dok se zadaci ponavljaju opet i ponovo, sve dok ne produciraju stabilni rezultat ili njihov rezultat više nije potreban. To zahtjeva funkcije višeg reda, lijenost i automatsko prikupljanje smeća.

Sustav iTask generira web-server kojeg koriste ljudski radnici za pronalazak svojih zadataka. Kao svi web-serveri, ovaj treba serijalizaciju i deserijalizaciju stanja da bi spremili i pristupili trenutnom stanju. Ispunjavajući web-forme za proizvoljne algebarske tipove podataka, korisnik prikazuje svoj napredak sa zadacima. Sva ta svojstva su implementirana koristeći generičko programiranje.

Da bi se efikasno implementirao nadzor zadataka vrijednosti SDS-a, postoji skriveni publish-subscribe sustav za svaki SDS koji aktivira zadatke koristeći taj SDS kad je njegova vrijednost promijenjena.

Osim ovim svojstava, sustav iTask koristi mnoge dodatne tehnike. Primjerice, izvođenje dijelova zadataka u interpreteru koji se izvršava unutar preglednika kako bi se osigurao brzi odziv sustava za jako interaktivne zadatke poput crtanja.

PODUČAVANJE NA ZIMSKOJ ŠKOLI

Svako podučavanje programiranja zahtjeva praktično programersko iskustvo koristeći obrazovne tehnike za ovladavanje njima. To stoji i za TOP. Zbog toga smo podijelili četiri sata dostupna za Why "Task Oriented Programming" matters u dva dijela skoro jednake veličine.

U prvom dijelu dajemo prikaz koncepta TOP-a koristeći iTask sustav. U svjetlu predznanja većine studenata, moramo preskočiti skoro sve detalje implementacije sustava i fokusirati se na korištenje knjižnice.

Knjižnica je zapravo plitko uloženi DSL za TOP. U predavanju koristimo skup primitiva, ne trošeći puno vremena na objašnjavanje arhitekture i implementacije.

Za praktičan rad, podijelit ćemo postojeći osnovni primjer projekta u skup malih nezavisnih TOP projekata. Studentski zadaci će se sastojati od malih varijacija tih projekata, kako bi se osjetila draž TOP-a.

Funkcijski programeri s više iskustva mogu preskočiti većino osnovnih zadataka i prijeći direktno na napredniji zadatak. Na taj način moći ćemo se prilagoditi vještinama svakog sudionika posebno.

SUSTAV MTASK

Mikroprocesori su računalni sustavi s vrlo ograničenim mogućnostima računanja. Obično imaju prilično nizak “clock rate” i velike memorijske restrikcije, poput samo nekoliko KB memorije za spremanje podataka programa koji se izvršava. Takvi jeftini procesori su pokretačka snaga mnogih elemenata u Internet of Things, IoT. U takvim mikroprocesorskim sustavima, obično treba motriti nekoliko ulaza, kao i nadzirati neke izlaze ovisno o ulazima.

Zbog hardverskih ograničenja, obično nema operativnog sustava koji bi pružio potporu.

TOP paradigma pruža lagane dretve koje su dobre za motrenje i koordiniranje napretka takvih dobro definiranih jednostavnih zadataka. Takvi zadaci se mogu izvršavati svojom brzinom, dok se kombinatorika i dijeljeni izvori podataka koriste za koordinaciju među njima. Izvršenje sustava iTask na IoT uređajima omogućit će nam konstrukciju programa koji se djelomično izvode na web-serveru, a djelomično na IoT uređajima. Ograničenja mikroprocesora onemogućuju izvođenje full-fledged iTask programa na IoT uređajima.

Kako bi aproksimirali idealno rješenje, razvili smo sustav mTask [3, 2]. To je višepregledni plitko uloženi DSL koji se može koristiti kao dio sustava iTask. Podržava TOP paradigmu uključujući iste rezultate zadataka kao sustav iTask, kombinatore zadataka i dijeljenih izvora podataka.

Po konstrukciji ovaj DSL nema funkcije višeg reda i rekurzivne tipove podataka. Zbog restrikcija ovog DSL-a, mTask programi mogu biti kompajlirani u kod koji se izvršava na mikroprocesorima. Usprkos tim restrikcijama, DSL je vrlo primjenjiv za specifikaciju zadataka koji će se lako i koncizno izvoditi na IoT uređajima.

KORIŠTENE TEHNIKE

Sustav mTask je višepregledni plitko uloženi DSL, zasnovan na tipu konstruktor klase [1]. Svaka instanca tih klasa definira interpretaciju, koju nazivamo pogled (view), programa konstruiranog od tih primitiva. Tipični pogledi implementiraju lijepi ispis, generiranje koda za mikroprocesore te simulaciju mTask programa kao iTask programa.

DSL je proširiv po konstrukciji, kako bi se koristile postojeće knjižnice za krajnje uređaje poput senzora temperature, konzola i servo motora. Jednostavno je dodati takvu knjižnicu kao jezični primitiv u sustav mTask uvodeći novi tip konstruktor klase i potrbne instance.

Da bi implementacija mTask-a bila prenosiva na puno različitih mikroprocesora i da bi se lako koristile postojeće C++ knjižnice, pogled generiranja koda producira C++ kod za Arduino platformu, umjesto mašinskog koda za neki posebni procesor. Kompajler avr-gcc unutar Arduino platforme može prevesti generirani C++ kod i korištene knjižnice u mašinski kod mnogo različitih mikroprocesora.

PODUČAVANJENA ZIMSKOJ ŠKOLI

Postići da se mogu izvesti TOP programi visoke razine na minijaturnom mikroprocesoru u interakciji s krajnjim uređajima je izazovno za predavanje na zimskoj školi. Međutim, izvođenje mTask programa na konkretnom mikroprocesoru zahtjeva puno od studenata; moraju sastaviti mTask program unutar sustava iTask, izvršiti iTask program da dobiju C++ kod, unijeti taj C++ kod u Arduino IDE, spojiti Arduino IDE s mikroprocesorom i odabrati točne opcije, uploadati kompajlirani program na mikroprocesor, i na kraju pokrenuti ga. Svi ti koraci su vrlo jednostavni, ali cijeli proces daje rezultat jedino ako je svaki korak napravljen ispravno.

Budući da će se generirani program izvršiti na mikroprocesoru bez operativnog sustava i s vrlo limitiranim ulaznim/izlaznim krajnjim uređajima, ispravljanje pogrešaka takvog programa je izazov. Nakon duže diskusije zaključeno je da je ovaj niz koraka previše ambiciozan za dato vrijeme i publiku.

Srećom, simulatorski pogled sustava mTask nudi alternativu koja je puno jednostavnija za upotrebu. Ovaj pogled pretvara mTask program u obični iTask program. Simulator nudi izvođenje korak po korak mTask programa. Prikazuje trag zadnjeg koraka posljednjeg izvršenog zadatka i stanje svih krajnjih uređaja i dijeljenih izvora podataka. Sat, vrijednost SDS-ova, kao i stanje krajnjih uređaja se može mijenjati interaktivno kako bi se nadziralo izvođenje i istražili razni scenariji. To čini praktični rad ovog predavanja direktnim sljedbenikom praktičnog rada na prethodnom iTask predavanju.

Odlučeno je staviti dva dijela predavanja u isti dan, tako da je predavanje o sustavu iTask i odgovarajući praktični rad ujutro, a predavanje o mTask sustavu popodne. Na taj način se predavanje o sustavu mTask direktno nastavlja na znanje i vještine dobivene na predavanju o sustavu iTask. Razumijevanje TOP-a formulirano ujutro, bit će stoga produbljeno popodne.

ZAKLJUČAK

Za obje serije predavanja o TOP-u, postoji još puno zanimljivih tema koje bi se mogle proći u danom vremenu za publiku iz zimske škole. Mi ćemo se fokusirati na razumijevanje i korištenje TOP paradigme na relativno jednostavnim primjerima. Malčice napredniji primjeri će biti korišteni da prikažu mogućnosti ovog pristupa. U praktičnom radu fokusirat ćemo se na ilustrativne zadatke koji su najvećim dijelom varijante primjera iz predavanja. Za napredne sudionike bit će izazovnih zadataka, kao i mogućnost dubinske diskusije svih aspekata sustava.

Literatura

[1] Carette, J., Kiselyov, O., Shan, C.c.: Finally tagless, partially evaluated: Tagless staged interpreters for simpler typed languages. J. Funct. Program. 19(5), 509–543 (Sep 2009). https://doi.org/10.1017/S0956796809007205, http://dx.doi.org/10. 1017/S0956796809007205

[2] Koopman, P., Lubbers, M., Plasmeijer, R.: A task-based dsl for micro- computers. In: Proceedings of the Real World Domain Specific Languages Workshop 2018. pp. 4:1–4:11. RWDSL2018, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3183895.3183902, http://doi.acm.org/10.1145/ 3183895.3183902

[3] Koopman, P., Plasmeijer, R.: A shallow embedded type safe extendable DSL for the Arduino. In: Revised Selected Papers of the 16th Interna- tional Symposium on Trends in Functional Programming - Volume 9547. pp. 104–123. TFP 2015, Springer-Verlag New York, Inc., New York, NY, USA (2016). https://doi.org/10.1007/978-3-319-39110-6_6, http://dx.doi.org/ 10.1007/978- 3- 319- 39110- 6_6

[4] Peyton Jones, S.L., Wadler, P.: Imperative functional programming. In: Pro- ceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 71–84. POPL '93, ACM, New York, NY, USA (1993). https://doi.org/10.1145/158511.158524, http://doi.acm.org/10. 1145/158511.158524

[5] Plasmeijer, R., Achten, P., Koopman, P.: iTasks: executable specifications of inter- active work flow systems for the web. In: Hinze, R., Ramsey, N. (eds.) Proceedings of the ICFP'07. pp. 141–152. ACM, Freiburg, Germany (2007)

[6] Plasmeijer, R., van Eekelen, M., van Groningen, J.: Clean language report (version 2.2) (2011), http://clean.cs.ru.nl/Documentation

[7] Wadler, P.: Comprehending monads. In: Proceedings of the 1990 ACM Conference on LISP and Functional Programming. pp. 61–78. LFP '90, ACM, New York, NY, USA (1990). https://doi.org/10.1145/91556.91592, http://doi.acm.org/10.1145/ 91556.91592

INTERAKTIVNI PRISTUP PODUČAVANJU OBOJENIH PETRIJEVIH MREŽA

Formalne metode spadaju u tehnike koje, ako se pravilno koriste, mogu značajno doprinijeti točnosti softverskog ili hardverskog sustava koji se razvija. Jedna od metoda za sustave s konkurentnim ili nedeterminističkim ponašanjem jest jezik za modeliranje obojenih Petrijevih mreža. U ovom članku, podučavanje clja na objašnjenje osnovnih principa jezika te opis nekih svojstava vezanih za funkcijsko programiranje. Trajanje aktivnosti je bilo dva i pol sata i uključivalo je izradu interaktivnog modela s aktivnim sudjelovanjem publike.

UVOD

Uzimajući u obzir rastuću ovisnost suvremenog ljudskog društva o računalnim sustavima, njihova točnost bi trebala biti od najvećeg značaja. Jedan od pristupa koji može značajno doprinijeti točnosti jest korištenje formalnih metoda tokom razvoja softvera i hardvera. Formalna metoda je tehnika, bazirana na matematici, koja daje formalni jezik s strogo definiranom sintaksom i semantikom te aparatom koji omogućuje izvođenje verifikacijskih, razvojnih i simulacijskih zadataka sa specifikacijama sustava, napisanim u tom jeziku. Jedna od značajnih članova familije formalnih metoda jest jezik za modeliranje Coloured Petri Nets (CPN). CPN [4, 3] kombiniraju formalizma Petrijevih mreža [1] s funkcijskim jezikom kako bi se moglo manipulirati podacima i procedurama za odlučivanje.

Funkcijski jezik, koji se zove CPN ML, malo je modificirana verzija jezika Standard ML [2, 5]. Jezik CPN i odgovarajući specifikacijski, verifikacijski i simulacijski zadaci su podržani sa softverom CPN Tools [6].

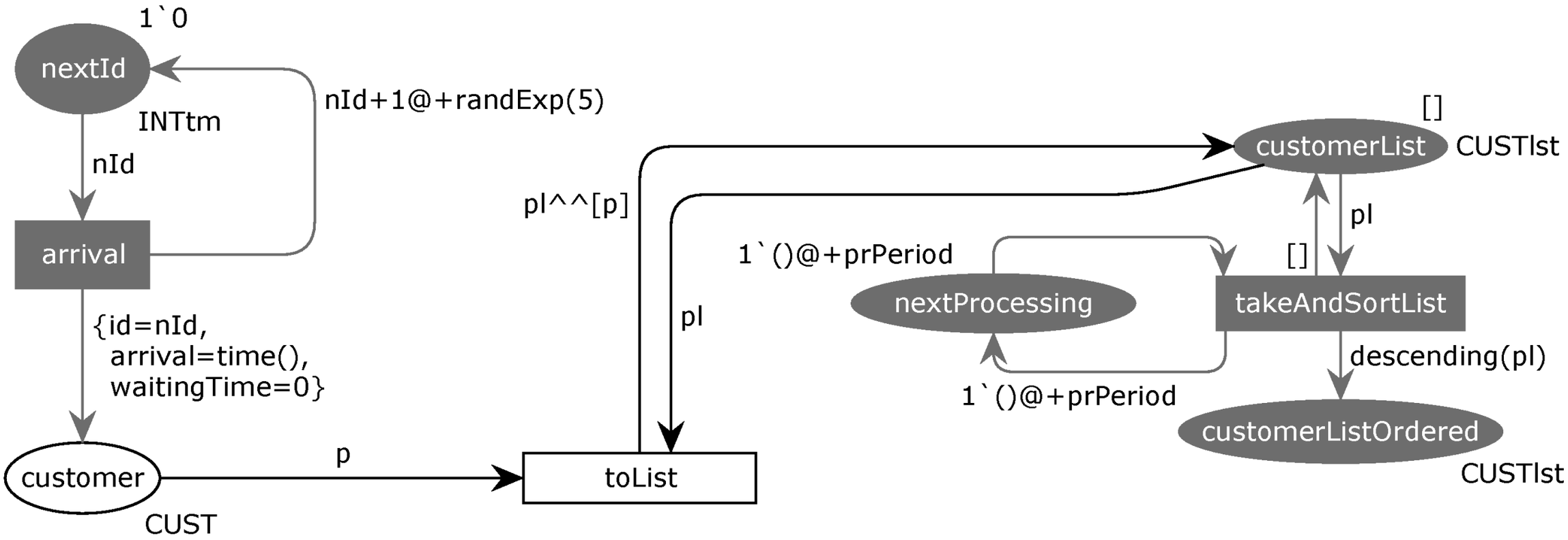
Već dulje od dekade, CPN su dio preddiplomskih kolegija vezanih za formalne metode, modeliranje i simulacije autorove institucije. Jedna od metoda, koju autor primjenjuje kada objašnjava koncepte CPN-a, je interaktivni pristup s aktivnim sudjelovanjem publike. Pritom, publika bira domenu i proces za koje će biti dizajniran CPN model te pomaže izradu njegovih izabranih dijelova. Iskustvo primjene ovog pristupa na treningu za sveučilišne nastavnike je opisano u ostatku ovog članka.

TRENING AKTIVNOST IZRADE INTERAKTIVNOG CPN MODELA

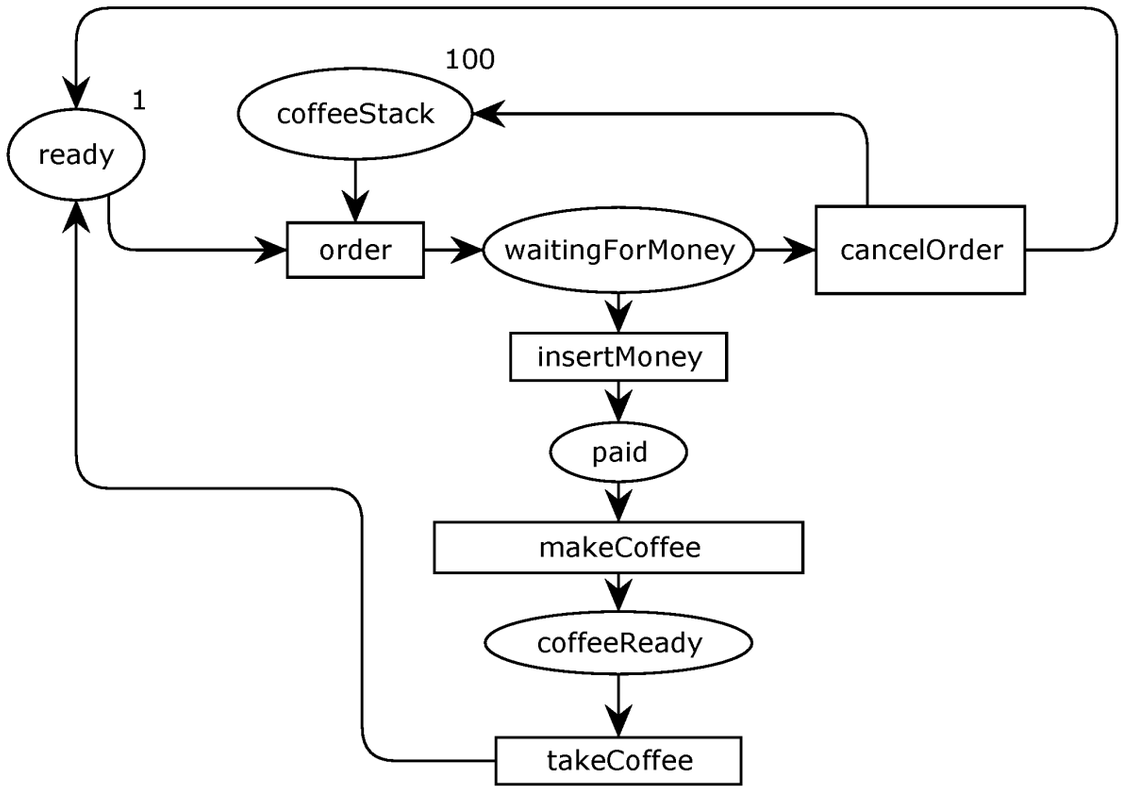
Trening aktivnost je organizirana za oko 10 sudionika, koji su bili sveučilišni nastavnici s izvjesnim poznavanjem funkcijskog programiranja. Sudionici su imali ograničeno ili nikakvo znanje o CPN-u. Ukupno trajanje aktivnosti je oko 2.5 sata, bez pauzi, i bila je podijeljena u tri faze. Prva faza uzela je oko 30 minuta i objasnila je osnovne principe CPN-a. Točnije, CPN ima grafički oblik bipartitnog grafa s dva tipa vrhova: mjesta, nacrtana kao elipse te tranzicije, nacrtane kao pravokutnici. Mjesta nose tokene, koji predstavljaju stanje mreže, a tranzicije mogu biti shvaćene kao događaji, koji mijenjaju stanja uzimajući tokene i kreirajući nove.

Druga i treća faza su posvećene izradi CPN modela. Kako je jedan od ciljeva aktivnosti bio pokazati kako neki napredniji Standard ML koncepti, točnije strukture i funktori, mogu biti korišteni u CPN modelima, sudionicima je dan početni CPN model, koji već koristi te koncepte, prije početka druge faze. Početni model je prikazan na slici 1. Dio koji se sastoji od čvorova nextId, arrival i customer predstavlja dolazak mušterija, koje dolaze jedan po jedan da bi bili posluženi. Samo posluživanje nije prikazano u početnom modelu.

Umjesto toga, imamo tranziciju toList, koja uzima token od customer-a i dodaje njegovu vrijednost na listu, spremljenu na mjestu customerList. Tranzicija takeAndSortList se izvede u pravilnim intervalima, definiranim s vrijednošću prPeriod.



Za dio modela koji poslužuje, sudionici aktivnosti odlučili su modelirati automat za kavu. Tokom druge faze, oni su sudjelovali u izradi CPN modela koji prikazuje osnovni rad automata. Model je prikazan na slici 2. Njegovo početno stanje je da je automat spreman (ready) da posluži mušteriju (jedan token je na mjestu ready) i napunjen sa 100 doza kave (100 tokens na coffeeStack). Posluživanje započinje s mušterijom koji naručuje kave izvođenjem tranzicije order. Tada automat čeka na mušterijin sljedeći korak (token u waitingForMoney). Mušterija može ubaciti novac (izvodi insertMoney) ili poništiti naruđbu (izvodi cancelOrder).



Poništenje vraća automat u stanje “ready”. Ako je novac ubačen, automat priprema kavu (izvodi makeCoffee). Na kraju, izvođenjem takeCoffee, mušterija uzima pripremljenu kavu i automat se vraća u stanje “ready”.

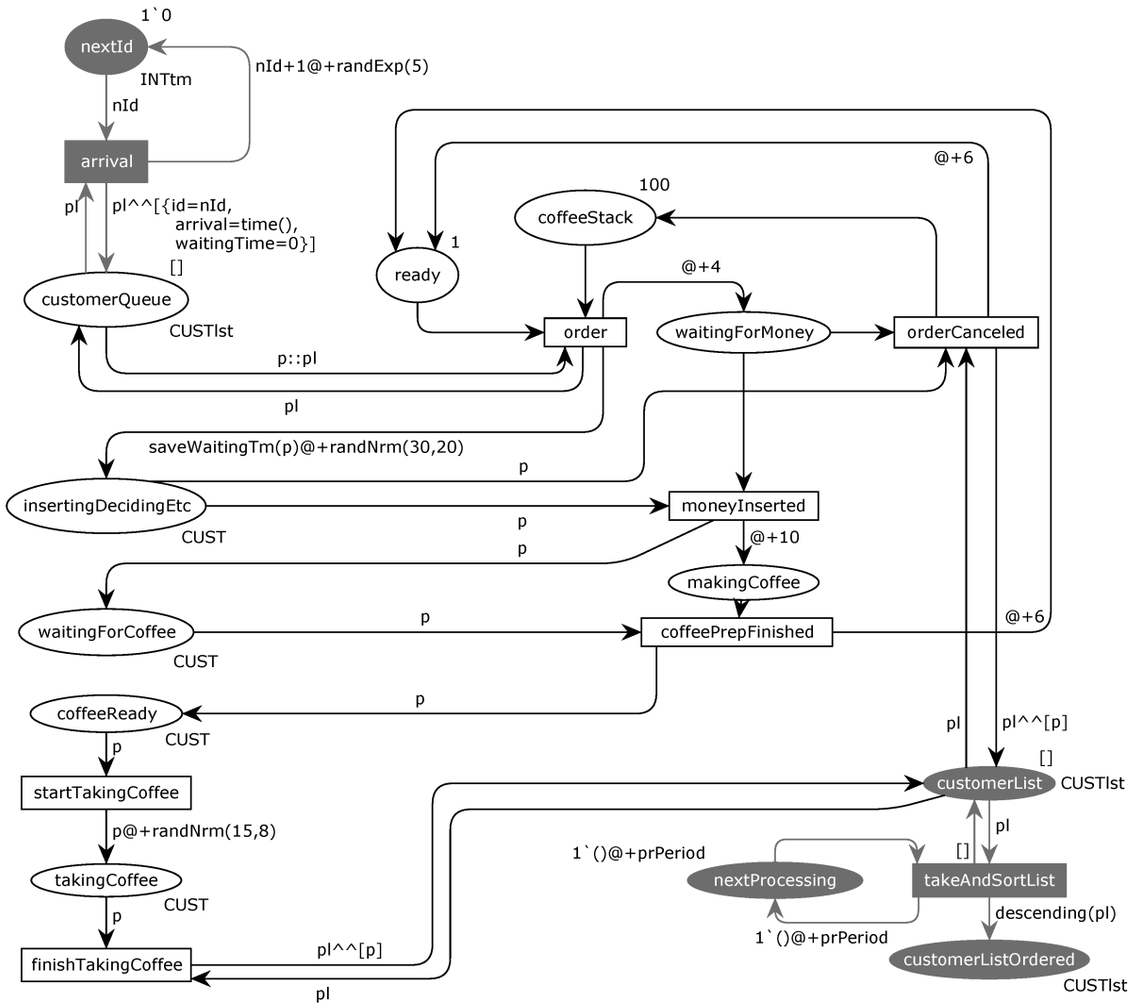
Nakon druge faze napravljena je pauza duga 70 minuta. Tokom pauze predavač je povezao model iz druge faze s dijelovima početnog modela te dodao vrhove i bridove koji opisuju ponašanje mušterije. Popravio je također neke nekonzistentnosti u modelu, koje je uočio jedan od sudionika. Rezultat je konačni CPN model prikazan na slici 3. Vrhovi uzeti iz početnog modela (slika 1) bez promjene su sivi.

Mjesto customer je zamijenjeno s customerQueue, koje nosi token s listom vrijednosti, predstavljajući red mušterija koje čekaju na automat. Umjesto tranzicije toList dodan je dio za posluživanje, izrađen od rezultata druge faze (slika 2). Dio za posluživanje konačnog modela se razlikuje od slike 2 u tri ključna aspekta:

– Tokeni nose informacije o mušteriji koja se poslužuje, a bridovi definiraju trajanje odgovarajućih akcija.

– Nekonzistentnosti u vezi uloge mjesta i tranzicija su ispravljene. Sada sve tranzicije predstavljaju trenutne događaje. Primjerice, tranzicija makeCoffee sa slike 2 je zamijenjena s mjestom makingCoffee, a tranzicija takeCoffee je zamijenjena s vrhovima startTakingCoffee, takingCoffee i finishTakingCoffee.

– Akcije i stanja mušterije i automata su modelirana zasebno. Vrhovi customerQueue, insertingDecidingEtc, waitingForCoffee, coffeeReady, startTakingCoffee, takingCoffee i finishTakingCoffee pripadaju mušterijama, dok ostatak dijela modela za posluživanje predstavlja automat ili oboje.



Treća faza trening aktivnosti je bila posvećena objašnjenju konačnog modela i diskusiji o ulazi takvog modela u razvoj ispravnih računalnih sustava. Uzela je oko 30 minuta.

ZAKLJUČAK

Interaktivna trening aktivnost, koja je prezentirana ovdje, je primjerena za kratak intenzivan kurs koji se često javlja tokom ljetnih škola i sličnih događaja. Opisano testno izvođenje aktivnosti otkrilo je da originalno predviđeno vrijeme od 2 sata nije dovoljno. Stoga je bila potrebna treća faza u kojoj je predavač prezentirao konačni model. Uzevši u obzir vrijeme potrebno predavaču za izradu modela, bilo bi potrebno barem još dva sata za interaktivnu itradu cijelog modela. Svi CPN modeli prezentirani ili spomenuti ovdje mogu biti dobiveni zahtjevom od autora.

Literatura

[1] Desel,J.,Reisig,W.:Place/transitionpetrinets. In:Reisig,W.,Rozenberg,G.(eds.) Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science, vol. 1491, pp. 122–173. Springer Berlin Heidelberg. DOI: 10.1007/3-540-65306-6 (1998)

[2] Harper, R.: Programming in Standard ML. Carnegie Mellon University (2011), http://www.cs.cmu.edu/~rwh/smlbook/book.pdf

[3] Jensen, K.: An introduction to the theoretical aspects of coloured petri nets. In: A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium. pp. 230–272. Springer-Verlag, London, UK. DOI: https://doi.org/10.1007/3-540-58043- 3_21 (1994)

[4] Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer. DOI: 10.1007/b95112 (2009)

[5] Milner, R., Tofte, M., Macqueen, D.: The Definition of Standard ML. MIT Press, Cambridge, MA, USA (1997), http://sml-family.org/sml97-defn.pdf

[6] CPN tools homepage (2018), http://cpntools.org/

CODE COMPASS: PROŠIRIVI OKVIR ZA RAZUMIJEVANJE KODA

CodeCompass je alat otvorenog koda koji pomaže u razumijevanju velikih naslijeđenih softverskih sustava. Na temelju infrastrukture kompajlera LLVM / Clang, CodeCompass daje točne informacije o složenim elementima jezika C / C ++. Širok raspon interaktivnih vizualizacija uključuje dijagrame poziva klase i funkcije; arhitektonski, komponentni i interfejs dijagrami i dijagrami "ukazuje na" i mnoge druge. CodeCompass također koristi informacije o sastavljanju za istraživanje arhitekture sustava kao i informacije o kontroli verzije kad su dostupne. Integrirani su i rezultati statičke analize temeljene na Clangu. Iako se alat uglavnom fokusira na C i C ++, također podržava jezike Java i Python. Imajući internetsku, prilagodljivu, proširivu arhitekturu, CodeCompass okvir može biti otvorena platforma za daljnje razumijevanje koda, statičke analize i napore softverske metrike.

UVOD

Ispravljanje pogrešaka ili razvoj novih značajki zahtijeva pouzdano razumijevanje svih detalja i posljedica planiranih promjena. Alati za razumijevanje koda mogu vam pomoći u otkrivanju izvornih namjera i detalja implementacije izgradnjom modela iz izvornog koda i ostalih dostupnih informacija. Iako je jedan broj takvih alata dostupan ili kao vlasnički ili besplatni softver, njihov je niz značajki ograničen.

CodeCompass razvijen je za uklanjanje ovih ograničenja.

Projekt CodeCompass zajednički je pokušaj Ericssona doo s otvorenim kodom i Sveučilišta Eötvös Loránd, Budimpešta, kako bi pomogli u razumijevanju velikih softverskih sustava. Da biste pružili točne podatke o složenim C / C ++ jezičnim elementima kao što su preopterećenje, nasljeđivanje, upotreba varijabli i vrsta, moguće uporabe pokazivača funkcija i virtualnih funkcija - značajke koje različite postojeće alate djelomično podržavaju - CodeCompass se temelji na stvarnom prevoditelju, LLVM / Clang infrastruktura. Na taj način se uklanjaju slabosti uobičajenih alata za razumijevanje "male težine", poput OpenGroka.

CodeCompass, međutim, nije ograničen na izvorni kod. Koristi informacije o izgradnji sustava za otkrivanje arhitektonskih veza. Također koristi informacije o kontroli verzije ako su dostupne, pa se tako mogu prepoznati veze između različitih izvornih datoteka „slučajno“ izmijenjenih u istom odredu. Da bi pomogao brzoj i preciznoj percepciji, CodeCompass koristi tekstualni i grafički prikaz softverskog sustava za razumijevanje. Brojni (interaktivni) dijagrami dostupni su s uobičajenih grafikona poziva na jedinstvene arhitektonske dijagrame. Kako bi korisnicima omogućio lak pristup, CodeCompass ima arhitekturu koja se temelji na webu. Klijent može biti standardni web preglednik, dodatak za uređivanje ili bilo koji program treće strane. Komunikacija se temelji na REST API-ju i dobro se mjeri za paralelne zahtjeve klijenta.

U ovom ćemo radu usporediti CodeCompass s postojećim alatima za razumijevanje i opisati njegov skup značajki. U Odjeljku 2 pregledavamo glavne arhetipove postojećih alata za razumijevanje koda. Uvodimo proširivu arhitekturu CodeCompass-a u 3. Glavne značajke alata razmatrane su u odjeljku 4. Rad sažeto prikazujemo u odjeljku 5.

SRODNI RADOVI

Na tržištu softvera postoji nekoliko alata koji ciljaju neku vrstu razumijevanja izvornog koda. Neki od njih koriste statičku analizu, drugi ispituju i dinamičko ponašanje raščlanjenog programa. Ovi se alati mogu podijeliti u različite arhetipove na temelju njihovih arhitektura i njihovih glavnih principa. S jedne strane alati imaju arhitekturu poslužitelja-klijenta. Ovi alati obično raščlanjuju projekt i spremaju sve potrebne podatke u bazu podataka. Klijenti se (obično putem Interneta) poslužuju iz baze podataka. Ovi alati mogu se integrirati u tijek rada dok rade noćni CI.

Na taj način programeri mogu uvijek pregledavati i analizirati čitava, velika, naslijeđena baza podataka kodova. Također postoje teški klijenti aplikacije u kojima je manji dio baze kodova raščlaniti. Ovo je slučaj upotrebe za IDE urednike gdje česta izmjena izvora zahtijeva brzo ažuriranje baze podataka o analiziranim rezultatima. U ovom odjeljku smo predstaviti neke alate koji se koriste u industrijskom okruženju od svake kategorije.

Woboq [3] je internetski preglednik kodova za C i C ++. Ovaj alat ima opsežne značajke koje imaju za cilj brzo pregledavanje softverskog projekta. Korisnik može brzo pronaći datoteke i imenovane entitete pomoću polja za pretraživanje koje omogućava dovršavanje koda radi jednostavne upotrebe. Navigacija u bazi koda omogućena je putem web stranice koja se sastoji od statičkih HTML datoteka. Te se datoteke generiraju tijekom procesa raščlanjivanja. Prednost ovog pristupa je u tome što će web klijent biti brz jer nije potrebno računanje u pokretu na strani poslužitelja tijekom pregledavanja.

Kretanje mišem na određenu funkciju, klasu, varijablu, makro itd. Može pokazati svojstva tog elementa. Na primjer, u slučaju funkcija može se vidjeti njegov potpis, mjesto njegove definicije i mjesto upotrebe. Za klase se može provjeriti veličina objekata, raspored klase i pomak njegovih članova i nasljedni dijagram. Za varijable može se provjeriti njihova vrsta i mjesta na kojima su napisane ili pročitane.

U C i C ++ makronapisi formiraju podjezik koji se procjenjuje u koraku predkompilacije. Ova je procjena tekstualna zamjena makro tokena, što znači da faza kompilacije djeluje s drugačijim kodom od originalnog. U Woboq-u se može provjeriti i konačna vrijednost makronaširivanja.

Vrlo korisna značajka alata je semantičko isticanje. Po ovoj se značajki lako mogu razlikovati različiti jezični elementi: oblikovanje lokalnih, globalnih ili članskih varijabli, virtualne funkcije, vrste, typedefs, klase, makroi, itd. Su različiti.

Woboq može pružiti gore navedene značajke jer se potrebne informacije prikupljaju u stvarnoj fazi sastavljanja. Ispitani projekt najprije mora sastaviti i analizirati Woboq. Analiza se vrši pomoću infrastrukture LLVM / Clang koja čini cjelokupno stablo apstraktnih sintaksa. Na taj se način svi dijelovi semantičkih informacija mogu izvući istom semantikom koju mora imati i konačni program. To također daje nedostatak alata, naime, Woboq se može koristiti samo za pregledavanje projekata C i C ++.

OpenGrok [4] brzi je motor za pretraživanje izvornog koda i cross reference. Odlučen za Woboq, ovaj alat ne provodi dubinsku analizu jezika, stoga ne može pružiti semantičke informacije o pojedinim entitetima. Umjesto toga, koristi Ctags [5] za pariranje izvornog koda samo tekstualno i za određivanje vrste specifičnih elemenata. Jednostavna sintaktička analiza omogućuje razlikovanje imena funkcija, varijabli ili klase itd. Pretraživanje među njima je vrlo optimizirano, a samim tim i vrlo brzo čak i na velikim bazama koda. Pretraživanje se može obaviti složenim izrazima (npr. Defs: target), koji sadrže čak i slojevite kartice, osim toga, rezultati se mogu ograničiti na poddirektoriju. Pored pretraživanja teksta postoji mogućnost zasebnog pronalaska simbola ili definicija. Nedostatak semantičke analize omogućava Ctagovima da podržavaju nekoliko (41) programskih jezika. Također je prednost ovog pristupa to što je moguće postupno ažurirati bazu podataka indeksa. OpenGrok također pruža mogućnost prikupljanja informacija iz sustava za nadzor verzija poput Mercurial, SVN, CSV, itd.

Uz funkcije pretraživanja koda koje smo već spomenuli za prethodne alate, Razumijeva pruža mnoštvo mjernih podataka i izvještaja. Neki od njih su kodni redovi (ukupno / prosječno / maksimalno na globalnoj razini ili po razredu), broj sjedinjenih / baznih / izvedenih klasa, nedostatak kohezije [2], složenost McCabea [1] i mnogi drugi. Treemap je uobičajena metoda reprezentacije za sve mjerne podatke. To je ugniježđeni pravokutni prikaz gdje gniježđenje predstavlja hijerarhiju elemenata, a dimenzije boje i veličine predstavljaju metriku koju odabere korisnik.

Za velike baze kodova potreban je pregled arhitekture. Razumijevanje može prikazati dijagrame ovisnosti na temelju različitih odnosa kao što su hijerarhija poziva funkcija, nasljeđivanje klase, ovisnost datoteke, uključivanje / uvoz datoteka. Korisnici također mogu kreirati svoju prilagođenu vrstu dijagrama putem API-ja kojeg pruža alat.

U programiranju su osnovni pojmovi uobičajeni za sve jezike, ali postoje neki koncepti koji se na određenom jeziku različito tumače. Razumijevanje može podnijeti ~ 15 jezika i može pružiti određene podatke o jeziku o kodu, npr. analiza pokazivača funkcije u C / C ++ ili dijagramima hijerarhije paketa na Adi.

Understand gradi bazu podataka iz baze koda. Sve informacije mogu se prikupiti putem programabilnog API-ja. Na ovaj način korisnik može zatražiti sve potrebne podatke koji nisu obuhvaćeni korisničkim sučeljem.

CodeSurfer [7] je sličan Understandu u smislu da je ujedno i gusti klijent, aplikacija za statičku analizu. Njezin je cilj razumijevanje projekata C / C ++ ili x86 strojnog koda. CodeSurfer provodi dubinsku analizu jezika koja pruža detaljne informacije o softveru ponašanja. Na primjer, ona provodi analizu pokazivača kako bi provjerila koji pokazatelji mogu ukazivati na datu varijablu, navodi izjave koje ovise o odabranom izrazu analizom utjecaja i koristi analizu protoka podataka da utvrdi gdje je varijabli dodijeljena njena vrijednost itd.

ARHITEKTURA CODE COMPASS-A

U prethodnom smo dijelu naveli neke aspekte koji se tiču ciljeva i arhitekture alata za razumijevanje koda. Sada predstavljamo gdje CodeCompass stoji među tim alatima.

CodeCompass ima arhitekturu klijent-poslužitelj u kojoj prikazuje informacije prikupljene u prethodnoj fazi raščlanjivanja. Razlog zašto je izabrana ta arhitektura dolazi iz cilja alata. Za razliku od uređivača koda, Code-Compass je planiran kao alat za razumijevanje koda. Postoje temeljne razlike između ova dva slučaja upotrebe. Tijekom pisanja koda, programeri istovremeno manipuliraju sa samo nekoliko datoteka. Međutim, pri razumijevanju koda potrebno je razmotriti izvore više modula kroz bazu kodova. U uređivačima je dovršavanje koda jedna od najkorisnijih značajki: programer se ne želi sjećati svih metoda i polja klase, ali zahtijeva da ih urednik nabroji. Za razumijevanje koda potreban je širok raspon vizualizacija kako bi se pregledali odnosi dijelova koda. Dok uređuje izvor, programer se usredotočuje samo na relativno mali fragment koda, poput funkcije ili klase. U razumijevanju koda ne samo da se ponašaju funkcije na niskoj razini, već se njihove ovisnosti i učinci razmatraju u kontekstu modularnog sustava visoke razine.

Glavno korisničko sučelje CodeCompass-a temelji se na webu. Sve gore navedene vizualizacije i funkcionalnosti mogu se pretražiti putem javnog API-ja koji je dodijeljen poslužiteljskoj aplikaciji. Web sučelje obrađuje slučajeve upotrebe koji imaju za cilj brzo i praktično zadaće pregledavanja, pregledavanja i razumijevanja. Međutim, CodeCompass je više od samo alata za pretraživanje koda. To je ujedno i okvir, tj. Proširivi sakupljač i predstavljač procesa statičke analize. Zbog toga namjera nije bila stvoriti zahtjevnu aplikaciju koja pohranjuje rezultate analize na strani klijenta, već je u mogućnosti služiti

različitim potrebama korisnika. Na ovaj je način moguće primijeniti skriptu, na primjer, koja prikuplja skup funkcija koje tvore zatvaranje prema odnosu poziva na funkciju, na taj način specificirajući koherentan odsječak softvera.

Drugi je dizajnerski zahtjev CodeCompass-a bio da postupa s velikim brojevima kodova i još uvijek odgovara na zahtjeve korisnika vrlo brzo, tj. U većini sekundi. To se postiže tako da se u bazu podataka pohrani sva najmanja količina podataka koja je dovoljna za odgovor na zahtjeve. Budući da smo željeli dati precizne rezultate za upite, potreban je prethodni postupak raščlanjivanja. U prvom smo pohranili cijelo apstraktno stablo sintakse izvora, ali to je rezultiralo u omjeru 1: 1000 između izvornog koda i veličine baze podataka. No, ispostavilo se da većinu slučajeva korisnici zanimaju samo imenovani entiteti (funkcija, varijable, klase, makroi, itd.), Tako da je bilo nepotrebno pohraniti bilo što drugo, poput upravljačkih struktura ili drugih izjava. Unatoč tome, postoje neki zadaci koji zahtijevaju više od pohranjenih podataka, poput algoritma rezanja. Ako korisnik želi vidjeti učinke promjene vrijednosti varijable, tada moraju uzeti u obzir i izjave koje mijenjaju stanje. Ovo zahtijeva ponovno kôd u letu.

ZNAČAJKE CODE COMPASS-A

U ovom ćemo dijelu dati pregled značajki dostupnih putem standardnog GUI-ja. Kada opisujemo specifične jezične značajke, kao što je nabrajanje pozivanih metoda, uvijek ćemo pretpostaviti da je jezik projekta C ++ jer ima najnapredniju podršku u CodeCompass-u, ali slične su značajke dostupne za Java i Python.

Pretraga

Vjerojatno je najvažniji slučaj korištenja alata za razumijevanje koda pretraživanje. Možete pretražiti ili datoteku ili izvorni kod. Za pronalaženje elemenata izvornog koda alat nudi 3 različite mogućnosti pretraživanja:

U načinu pretraživanja cjelovitog teksta fraza pretraživanja je grupa riječi poput "vraća astnode *". Upitna fraza odgovara tekstualnom bloku, ako su tražene riječi jedna do druge u izvornom kodu navedenim određenim redoslijedom. Ulične znakove, poput *, ili? Može se koristiti, podudarajući bilo koji višestruki ili pojedinačni znak.

Logički operatori poput AND, OR, NOT mogu se koristiti za pridruživanje više fraza upita istovremeno.

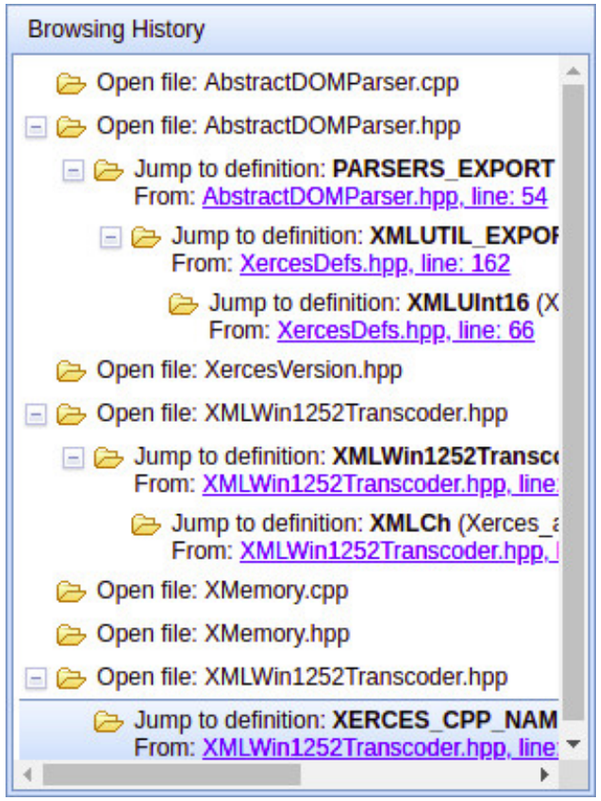
Na višoj razini moguće je pronaći simbole u izvornim kodovima pretraživanjem definicije. Ovdje koristimo CTagove za indeksiranje baze koda kako bismo mogli pronaći varijable, funkcije, klase, makronaredbe itd. Važno je znati da ovo pretraživanje jezičnih entiteta nema nikakve veze s dubokim analizama jezika.

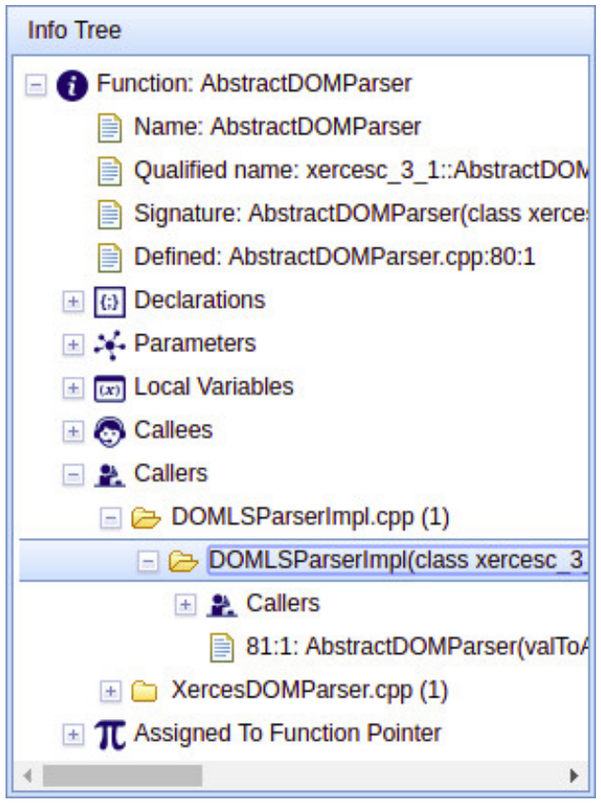
Dok uklanjate program za uklanjanje pogrešaka, ponekad je jedina informacija koju treba započeti s izlaznom porukom u dnevniku konzole koju emitira naš softver. To je jedini trag od kojeg se može započeti, npr. "DEBUG INFO: TSTHan: sys_offset = -0.019821, drift_comp = -90.4996, sys_poll = 5". Imajte na umu da takva poruka može sadržavati vremenske oznake ili druge dinamički generirane fragmente, tako da je nemoguće pronaći ovu poruku kao izravni niz. Međutim, u CodeCompass-u neizrazita pretraga može se izvršiti pretraživanjem dnevnika.

Informacije o jezičnim simbolima

Kad je element pronađen, sljedeći korak je prikupljanje podataka o njemu. Korisnik može odabrati "Info stablo" iz skočnog izbornika nakon što odabere imenovani entitet. Ovo stablo sadrži sve informacije koje pruža jezični analizator. U slučaju C / C ++ koristimo LLVM / Clang prevodilac kako bismo dohvatili informacije o simbolima.

Za funkcije možemo provjeriti njihove parametre, lokalne varijable, pozivatelje i pozive. Zanimljiva značajka stabla je da su pozivači predstavljeni recesivno, tj. Djeca čvora su pozivači funkcije. Čvorovi njihove djece su pozivači tih funkcija, a to se ponavlja, teoretski, na glavnu funkciju.





Međutim, pozivi funkcija nisu uvijek izravni, već se mogu dogoditi preko pokazivača funkcija. Iako se radi o dinamičnom ponašanju, CodeCompass saziva sve pojave u kojima je funkcija dodijeljena pokazivaču funkcije i poziva se putem ovog pokazivača.

U slučaju klasa, prikupljeni podaci su pseudonimi (prema vrstidef klasa može imati sinonim), nasljedni odnosi (grupirani po vidljivosti), prijatelji, metode / polja (izravna ili naslijeđena) i upotrebe (kao lokalna / globalna varijabla, funkcijski parametar / povratni tip ili polje druge klase).

Za varijable je korisno znati mjesta u kodu na kojima je napisana i pročitana. Za vrste nabrajanja konstante popisivanja su navedene s njihovim cijelim vrijednostima.

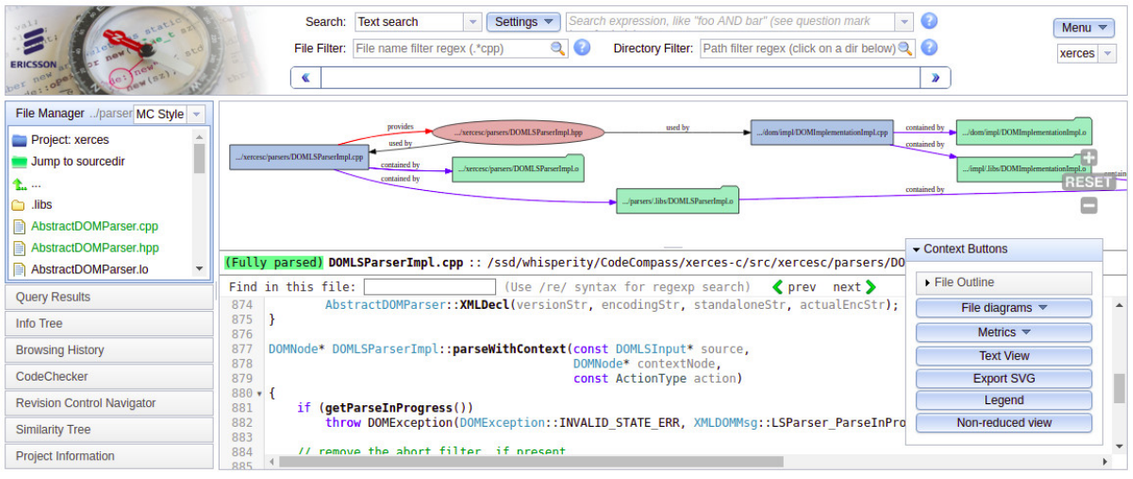
Dijagrami

Vizualizacije su jedan od najvažnijih prikaza za pregled ljudi u sustavu. CodeCompass predstavlja nekoliko dijagrama na temelju simbola i datoteka. Ti su dijagrami temeljeni na grafovima, tj. Predstavljaju subjekte i njihove veze. Ovo su ujedno i interaktivni dijagrami: pokazivačem miša preko čvorova prikazani entitet prikazuje se u prikazu teksta.

Klikom na njih odabrani entitet postaje središnji čvor koji prikazuje svoje odnose prema vrsti dijagrama.

Dijagram poziva funkcije prikazuje sve pozivatelje i pozive funkcija u grafikonu. Dijagram nasljeđivanja UML klase prikazuje cijeli lanac nasljeđivanja sve do osnovne baze klase i rekurzivno za sve izvedene klase. Također smo implementirali dijagram analize pokazivača koji pokazuje dodijeljene objekte i pokazivače koji ih možda upućuju. Naravno, to je dinamična informacija koja se samo dijelom može prikupiti statičkom analizom.

Dijagram sučelja koji se poziva na izvornu datoteku C / C + + pokazuje koje su zaglavlje dane datoteke „korištene samo“ ili „implementirane“. Upotreba znači da izvorna datoteka koristi drugu datoteku ako u njoj postoji upotreba simbola koja je deklarirana u drugoj datoteci. Odnos implementacije znači da se simbol deklarira u datoteci (tako formira sučelje) i definira u drugoj. Ti su odnosi primjenjivi i za direktorije koje imaju u vidu sadržane datoteke. U slučaju sastavljenog jezika, postoje i izlazne datoteke poput objekata i izvršnih datoteka. Na temelju podataka o vezama možemo predstaviti iz kojih izvora se sačinjava binarna datoteka.



CodeBites pruža drugačiju vizualizaciju pregledanog izvornog koda. U ovom su pogledu čvorovi grafikona definicije određenih imenovanih simbola, poput klasa, funkcija itd. Ideja je da programer želi otkriti ovaj entitet razumijevanjem njegovog ponašanja, ali bez gubitka fokusa. Dakle, dijelove teksta koda u čvoru je moguće kliknuti, što pokreće dodavanje definicije odabranog elementa.

Vizualizacija kontrole verzija

Vizualizacija podataka o kontroli verzija važno je pomagalo za razumijevanje evolucije softvera. Git blame view line-by-line prikazuje promjene (prinose) datoj datoteci. Promjene koje su se nedavno dogodile obojene su svijetlo zelenom bojom, dok su starije promjene tamnije crvene. Ovaj je pogled izvrstan za pregled zašto su određene linije dodane u izvornu datoteku. CodeCompass također može prikazati Git obveze u filtrabilnom popisu poredanom u vrijeme počinjenja. Ova pretraga može se koristiti za popis promjena koje je osoba napravila ili za filtriranje obveza po relevantnim riječima u poruci za počinjenje.

Metrike

CodeCompass može prikazati McCabe Cyclomatic Complexity [1], retke koda i broj pogrešaka pronađenih u Clang-ovim Static Analyzer metricama za pojedinačne datoteke i sažeti u hijerarhiji direktorija. Ove se metrike mogu vizualizirati na drvenoj karti, gdje su direktori označeni okvirima. Veličina okvira i njegova nijansa boja proporcionalna je odabranom metričkom stanju.

Povijest pretraživanja

De Alwis and Murphy istraživali su zašto su programeri disorijentirani kada koriste Eclipse Java integriranu razvojnu okolinu (IDE) [8]. Koristi se tehnika vizualnog momenta [9] kako bi se identificirala tri faktora koja mogu utjecati na disorijentaciju. I) nedostatak navigacije koja povezuje kontekst tokom istraživanja koda, ii) prijelaz između zaslona koji prikazujurazličite djelove koda, I iii)slijed od ponekad nepovezanih podzadataka.

Prvi faktor znači da je programer, tijekom istraživanja problema obilazi nekoliko datoteke kako slijedi u lancu poziva ili istražuju upotrebu varijable. Na kraju dugog sesija istraživanja, teško je to učiniti sjetite se zašto je istraga završila u određenoj datoteci.

Drugi razlog dezorijentacije su česti promjena različitih pogleda u Eclipseu. Treći suradnik problem je u tome što programer prilikom rješavanja programa promijeniti zadatak, procijeni nekoliko hipoteza, koje su sve pojedinačne podvrste razumijevanja. Programeri imaju tendenciju da obustavite podsklad (prije nego što ga dovršite) i prebacite se na još. Na primjer, programer istražuje kako koristi se povratna vrijednost funkcije, ali zatim se mijenja u a potpoziv razumijevanja provedbe funkcije sebe. Primijećeno je da je za programera to teško podsjećaju se na suspendirani podvrsta [10].

CodeCompass implementira pogled na povijest pretraživanja koji zapisuje (u stablastoj formi) putanju navigacije u izvornom kodu. Novi podzadatak je predstavljen sa novom granom u stablu, a čvorovi stabla su navigacijski skokovi u kodu koji su nazvani prema kontekstu na koji se spajaju (primjerice ‘skok na definiciju init). Tako da su problemi i) i ii) adresirani, sa čvorovima sa imenom iz povijesti pretraživanja a problem iii) se upravlja iz grana koje su pridodjeljene podzadacima.

CodeChecker - C/C++ Izvješćivanje o pogreškama

Clang Static Analyzer implementira naprednu simboličku izvedbenu mašinu za izvješćivanje o programskim pogreškama. CodeCompass može vizualizirati pogreške koje su identificirane sa Clang Static Analyzer alatom I Clang Tidy i povezivanjem sa CodeChecker serverom [11]. CodeCompass prikazuje poziciju pogreške I simboličnu izvedbenu putanju koja vodi ka pogrešci.

Adresni prostor I katalog tipova

CodeCompass procesuira Doxygen dokumentaciju i pohranjuje je za potrebe definicije funkcije, tipova i varijable. Isto osigurava pogled na katalog tipova koji navodi tipove koji su deklerirani u radnom prostoru I organizira ih prema hijerarhijskoj stablastoj strukturi u pogledu adresnog prostora.

SAŽETAK

Predstavili smo CodeCompass, alat za statičku analizu za razumijevanje velikih softvera. Dizajniran je na način da se izbjegnu brojni nedostaci postojećih alata za razumijevanje koji su ili prejednostavni, jednostavni za korištenje ali bez dubinskog znanja realnih kompjutera; ili su zahtjevni, neskalabilno instalirani na klijentskoj mašini. Web-bazirani, plugin, i alati sa proširivom arhitekturom mogu za okvir preuzeti otvorene platforme za buduće alate za razumijevanje koda, statičku analizu i rad sa softverskim metrikama. Inicijalni odaziv korisnika I statistike o korištenju alata pokazuju o korisnosti alata za softverske inžinjere u aktivnostima razumijevanja softvera I koristi se pored tradicionalnih IDE I drugih alata.

Literatura

[1] Thomas J. McCabe, A Complexity Measure, IEEE Transactions on Software Engineering: 308–320, December 1976

[2] Henderson-Sellers, Object-Oriented Metrics: Measures of Complexity Prentice-Hall, 1996, Upper Saddle River, NJ, ISBN-13: 978-0132398725

[3] Woboq, https://woboq.com/codebrowser.html, 18. 03. 2018

[4] OpenGrok, https://opengrok.github.io/OpenGrok, 18. 03. 2018

[5] CTAGS, http://ctags.sourceforge.net, 18. 03. 2018

[6] Understand, https://scitools.com, 18. 03. 2018

[7] CodeSurfer, https://www.grammatech.com/products/codesurfer, 18. 03. 2018

[8] B. De Alwis and G.C. Murphy, Using Visual Momentum to Explain Disorientation in the Eclipse IDE, Proc. IEEE Symp. Visual Languages and Human Centric Computing, pp. 51-54, 2006.

[9] D. D. Woods., Visual momentum, A concept to improve the cognitive coupling of person and computer. Int. J. Man-Mach. St., 21:229–244, 1984.

[10] D. Herrmann, B. Brubaker, C. Yoder, V. Sheets, and A. Tio. Devices that remind, In F. T. Durso et al., editors, Handbook of Applied Cognition, pages 377–407. Wiley, 1999.

[11] Daniel Krupp, Gyorgy Orban, Gabor Horvath and Bence Babati, Industrial Experiences with the Clang Static Analysis Toolset, EuroLLVM 2015 Confernece, April 2015

[12] E. Baniassad and G. Murphy, “Conceptual Module Querying for Software Engineering,” Proc. Int’l Conf. Software Eng., pp. 64-73, 1998.