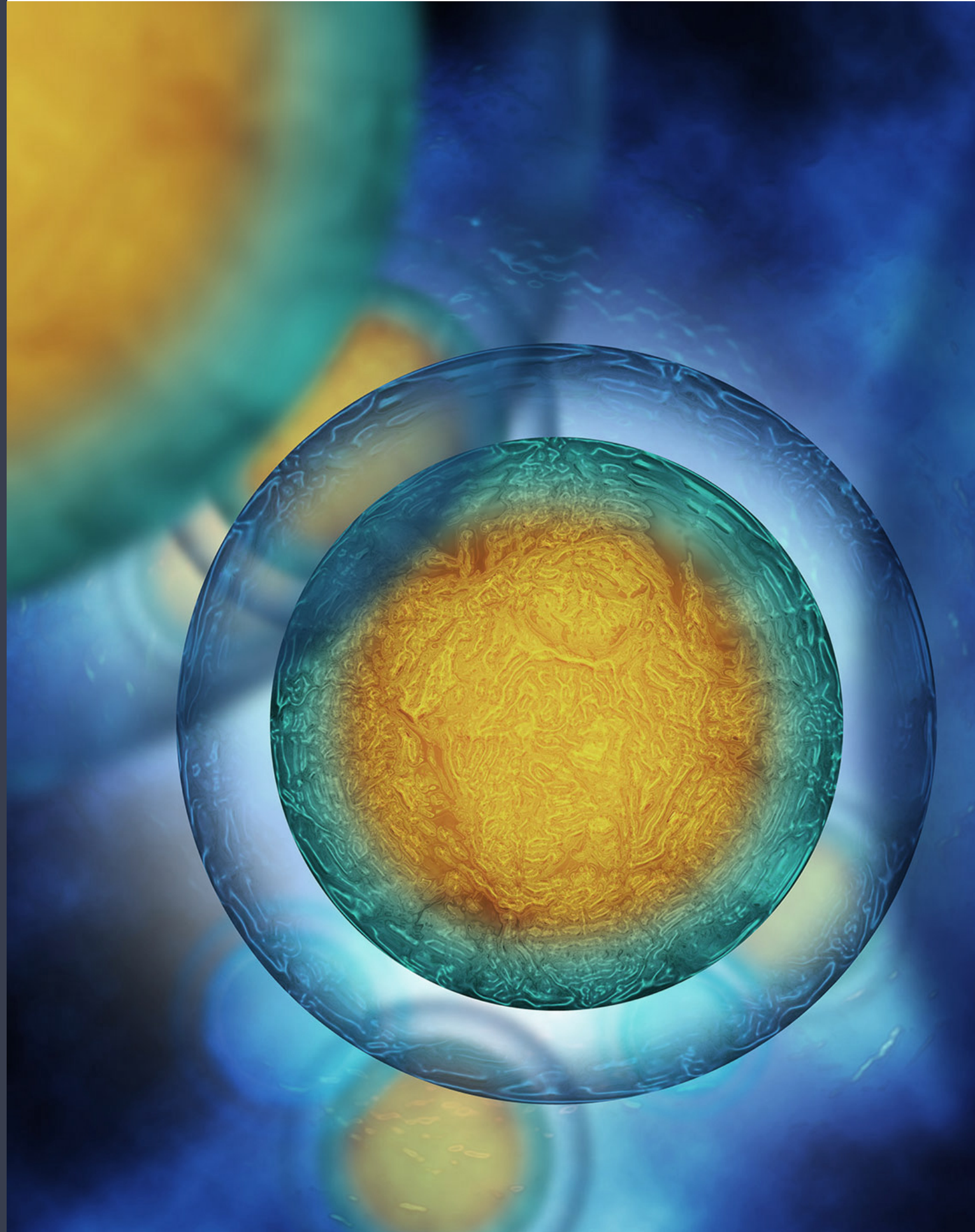
FE3CWS

MATERIÁL KU ŠKOLENIU V AMSTER-DAM-E

Intelektuálny výstup č.2
ERASMUS+ projektu číslo 2017-1-SK01-KA203-035402



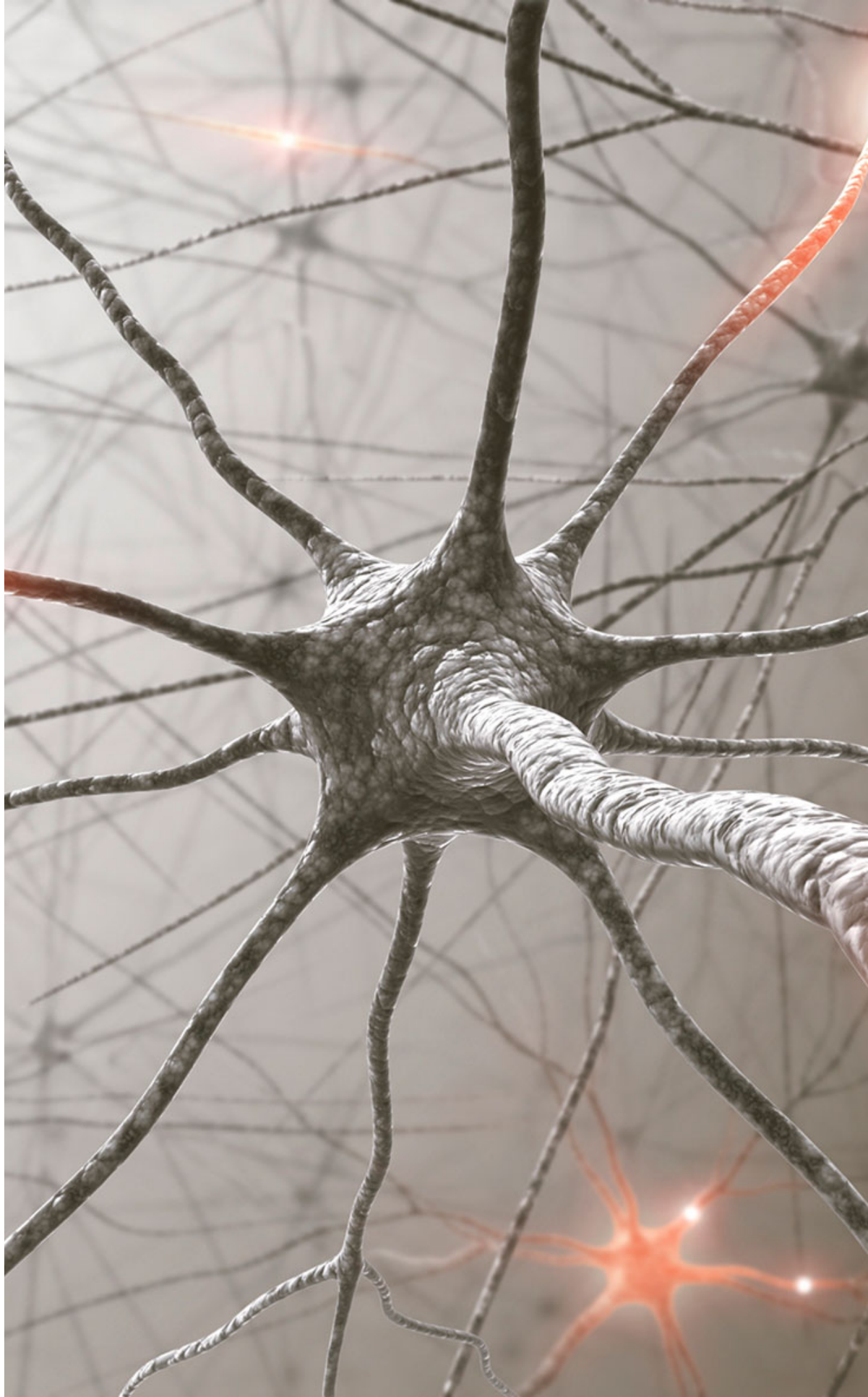
Niekoľko slov

O OBSAHU

- 6 tém o kompozícii, zrozumitelnosti a korektnosti softvéru
- Dostupný v 7 jazykoch: anglický, maďarský, slovenský, chorvátsky, rumunský, bulharský a portugalský

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

OBSAH



CLOUD COMPU-TING ZAMERANÝ NA POUŽÍVATEĽA VO VZDELÁVANÍ

Cloud computing sa stal kľúčovou technológiou, a preto je súčasťou mnohých učebných osnov informatiky. Výskum zameraný na používateľa sa zameriava na stratégie odhadovania doby prevádzky a nákladov na nasadenie aplikácií v reálnom svete v cloude.

V oblasti cloud computingu je dôležitým aspektom pomoc používateľom pri ich rozhodovaní. Takéto rozhodnutia sa týkajú týchto otázok:

- Ako sa správa správa na virtualizovaných zdrojoch?

- Koľko virtuálnych zdrojov toho typu, od ktorého by sa mal poskytovateľ cloudu získať na nasadenie aplikácií?

- Ako dlho? Koľko to bude stáť?

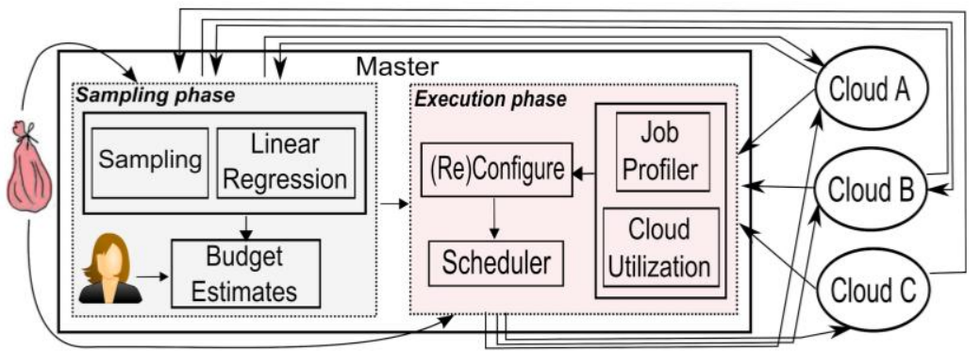
Tieto otázky sú zvyčajne modelované ako problém s plánovaním, pričom sa vychádza z predpokladu, že neexistujú a priori znalosti o aplikácii. Jedným typickým jednoduchým súborom požiadaviek je, že aplikácia je úspešne nasadená a náklady sú minimalizované.

ARCHITEKTÚRA PLÁNOVAČA PRE CLOUDOVÚ APLIKÁCIU

Plánovač BaTS [4] bol vyvinutý s cieľom pomôcť používateľom pri nasadzovaní ich aplikácií do cloudu. Na dosiahnutie tohto cieľa sa vyžaduje samoplánovanie a pravidelne kontroluje postup zavádzania.

Obrázok 1 zobrazuje architektúru BaTS. Počas fázy vzorkovania zhromažďuje spoločnosť BaTS štatistiku o čase vykonávania niektorých úloh aplikácie pomocou vzorkovania s náhradou. Tu je potrebná iba malá vzorka (30 - 50 úloh) na výpočet priemeru a štandardnej odchýlky runtime úloh pri rôznych cloudových ponukách. Lineárna regresia sa používa na optimalizáciu výpočtu rozpočtu a odhadov rozdielov.

Počas fázy vykonávania sa v pravidelných časových intervaloch prehodnocuje aktuálna konfigurácia, aby sa skontrolovalo, či je zvolený plán stále uskutočniteľný.



Ak sa očakáva porušenie rozpočtu, získajú sa ziskovejšie stroje (lepší pomer cena / výkon). Ak sa očakáva narušenie značky, získajú sa rýchlejšie stroje.

POUŽITIE METODIKY BATS NA ZDROJE AWS

Predstavujeme problém, ako pomôcť vlastníkom aplikácií pri výbere najvhodnejších možností z hľadiska virtualizovaných zdrojov pri nasadzovaní ich aplikácií na zdroje Amazon EC2 (AWS) [7].

Optimalizovaná fáza vzorkovania

Hlavnou myšlienkou je použiť priemerný čas vykonávania pre každý typ virtualizovaného zdroja na výpočet rozpočtu a odhadov rozdielov.

Získanie týchto štatistík však môže spôsobiť značné náklady vzhľadom na mnoho druhov ponúk AWS EC2 (v súčasnosti 123 [8]).

Obrázok 2 zobrazuje náhodne vybrané úlohy z aplikácie, v ktorej doby vykonávania úloh nasledujú po určitom rozdelení. Runtime týchto úloh sa používajú na výpočet štatistík pre jednu cloudovú ponuku. Po zhromaždení štatistík pre každú dostupnú cloudovú ponuku môžeme vypočítať rozpočet a urobiť odhady. Za predpokladu, že by sme chceli vyhodnotiť každú súčasnú ponuku AWS EC2, znamenalo by to vykonanie 30 úloh na každom zo 123 typov strojov. Keby sme jednoducho vykonávali rôzne sady náhodne vybraných úloh (spolu 3690), viedlo by to k dlhej (a nákladnej) fáze vzorkovania, čo by mohlo spôsobiť, že by akékoľvek rozhodnutie používateľa bolo irelevantné z dvoch dôvodov: a) zostáva príliš málo úloh na to, aby boli a b) možno už bol prekročený užívateľský rozpočet. Ak by sme vykonali rovnakú množinu náhodne vybraných pre každý typ počítača, viedlo by to stále k dlhej (a nákladnej) fáze vzorkovania.

Túto fázu optimalizujeme pomocou lineárnej regresie, aby sme znížili celkový počet úloh, ktoré je potrebné vykonať pred prípravou štatistík. Na každom type počítača vykonávame rovnakú skupinu siedmich náhodne vybraných úloh a zbierame runtime [9].

Ďalej vykonáme 23 náhodne vybratých úloh na počítačoch, ktoré sú prvýkrát k dispozícii. Obrázok 3 ilustruje náš prístup. Použitím runtime replikovaných 7 úloh vytvoríme lineárny vzťah medzi časmi vykonávania úloh aplikácie vo všetkých typoch počítačov. Tieto lineárne vzťahy potom použijeme na mapovanie 23 runtimeov na všetky ostatné typy strojov. Po dokončení mapovania máme pre každý typ počítača 30 bitov, zatiaľ čo namiesto 3690 vykonávame iba 884 úloh.

Rôzne hodnoty veľkosti v cenách

Po získaní odhadov runtime sa vypočítajú odhady rozpočtu a vyčíslenia pomocou modifikovaného algoritmu Bounded Knapsack [11]. Avšak pri zvažovaní zdrojov AWS EC2 s odlišným cenovým modelom, ako sú napríklad okamžité prípady, tento prístup nie je škálovaný kvôli rôznym veľkostiam.

Na vyriešenie tohto problému sme obchodovali s determinizmom prístupu Bounded Knapsack pre škálovateľnosť prístupu založeného na genetickom algoritme [12].

Pomocou genetického algoritmu by sme mohli aproximovať Paretov front (optimálny súbor) realizovateľných plánov pre danú aplikáciu a danú množinu typov strojov. Obrázok 4 zobrazuje skutočné Pareto front a dva odhady pre aplikáciu, ktorá má multimodálne rozdelenie runtime. Naše riešenie generuje presné Pareto fronty do 1 sekundy.

Kľúčovým nálezom tu bolo, že na zabezpečenie dobrého pokrytia skutočného predného Pareta by funkcia fitnes mala tiež odmeňovať najrýchlejší / najlacnejší make-up.

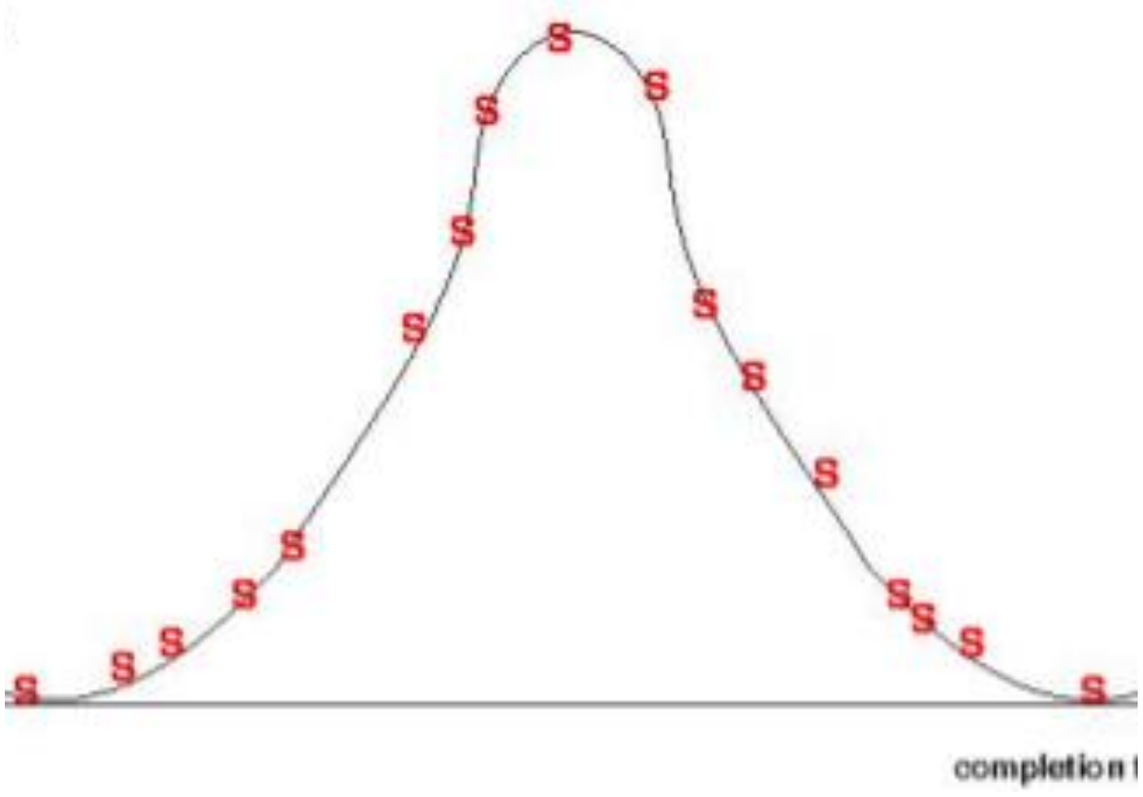
Koncová fáza výpočtu

V záverečnej fáze výpočtu je potrebné sa zaoberať základným predpokladom, že výpočtový čas je „tekutý" [10]. V tejto chvíli aplikácia podľa definície obsahuje príliš málo úloh, aby sa predpoklad mohol dodržať. Analyzovali sme niekoľko prístupov k riešeniu tohto problému so zameraním na lepšie odhadnutie konečných výpočtových potrieb aplikácie tak, aby rozdelenie konečných úloh na stroje bolo optimálne. Naše prístupy sa pohybovali od dokonalých znalostí zostávajúcich runtime až po nulové znalosti (random runtimes). Dopad bol zanedbateľný. Preto je potrebné tento problém riešiť v inej fáze, konkrétne vo fáze odhadu rozpočtu a stanovenia dôsledkov.

Na tento účel spoločnosť BaTS sleduje odhadované nevyužité konečné zlomky zodpovedajúcich časových jednotiek. Spoločnosť BaTS poskytuje vankúš na riešenie odľahlých hodnôt mimo konečnej zodpovedajúcej časovej jednotky a do plánu pridáva virtualizované zdroje a / alebo čas. Mimoriadne hodnoty však môžu viesť k zlyhaniu.

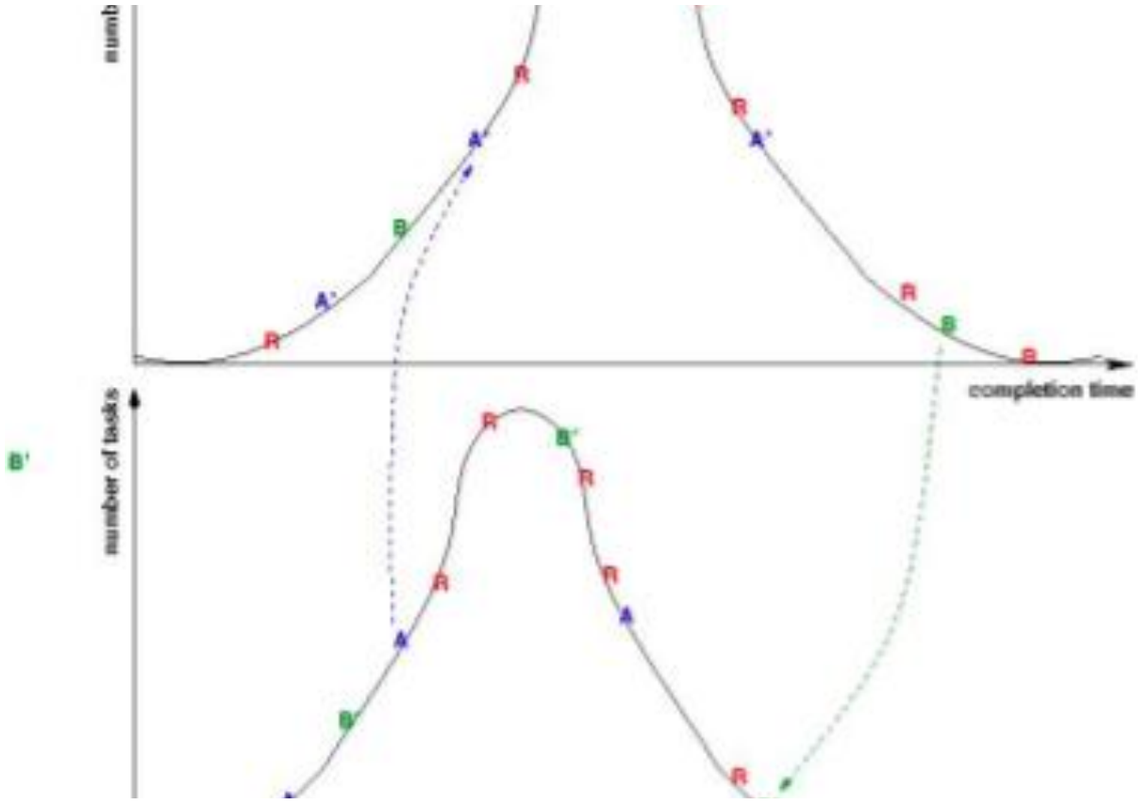
Využitie metodiky BaTS vo vzdelávaní

V študijných programoch Master of Computer Science na univerzite v Amsterdame sa kurz „Webové služby a cloudové systémy" vyučuje už niekoľko rokov. Jedným z dôležitých cieľov tohto kurzu je oboznámiť študentov s problémami cloudových systémov. Praktická práca na tomto kurze si vyžaduje vývoj základného plánovača pre virtualizované zdroje a analýzu jeho správania v internom prostredí v porovnaní s komerčným cloudom. Vlastné nastavenie pozostáva z nasadenia OpenNebula [5] v holandskom národnom klastri pre výpočty DAS [6].

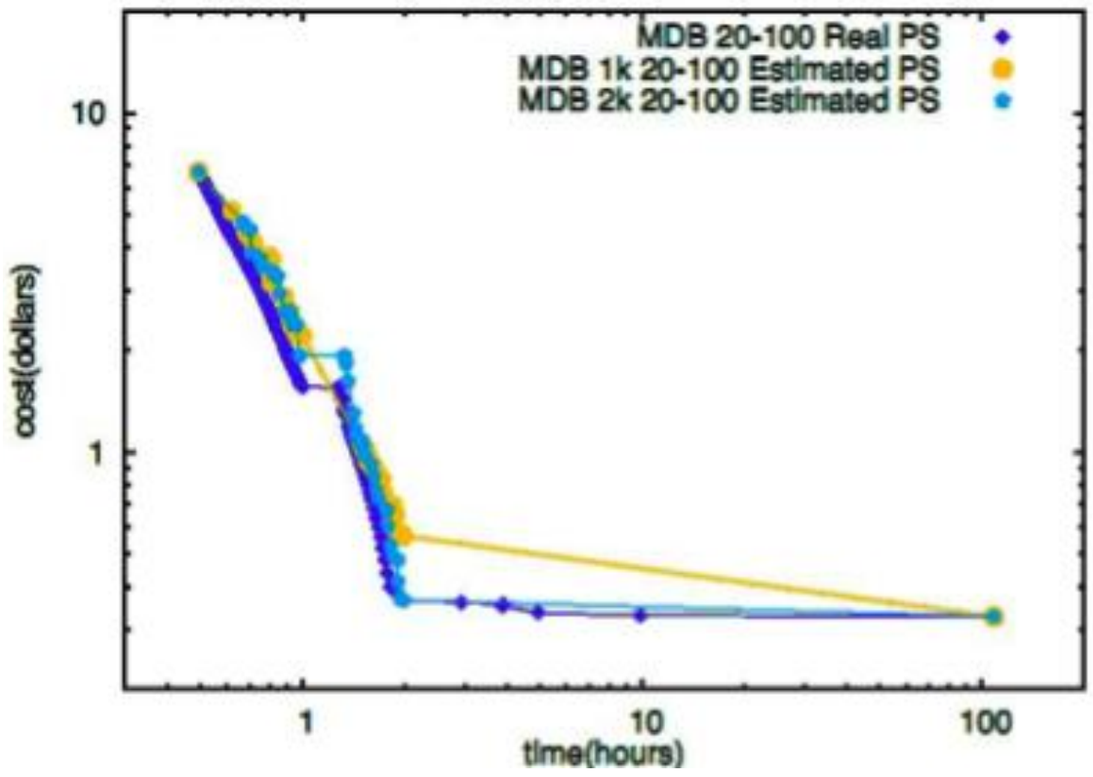


OpenNebula je riešenie s otvoreným zdrojovým kódom pre nasadenie Infraštruktúra ako služba: fyzické zdroje sú spravované a ponúkané ako virtuálne prostriedky. Tu majú študenti hornú hranicu počtu virtuálnych počítačov, ktoré môžu byť spustené súčasne.

Komerčné nastavenie cloudu pozostáva z cloudových ponúk Amazon EC2 [7]. Tu majú študenti rozpočet, ktorý môžu použiť na získanie akýchkoľvek laboratórnych zdrojov. Posúdenie ich laboratórnej práce zohľadňuje každé porušenie rozpočtu.



Výsledok laboratórnej práce vo všeobecnosti ukázal, že študenti boli schopní pochopiť rozdiel medzi najlepším úsilím a získavaním komerčných virtuálnych zdrojov. Zvyčajne vyvinuli plánovače založené na ziskovosti, zatiaľ čo najlepší študenti vyvinuli sofistikovanejšie plánovače, v ktorých si používatelia mohli vybrať z niekoľkých politík: najrýchlejší, najlacnejší a najziskovejší.



ZÁVER A BUDÚCA PRÁCA

Sľubné sú stochastické prístupy pre plánovanie cloudov zameraných na používateľa. Začlenenie výskumu do vzdelávania, len čo dosiahne stabilný stav, je veľmi dôležité.

Ako budúcu prácu by sme chceli podporiť funkcie Haskell AWS Lambda nasadené prostredníctvom implementácie rozhrania Haskell AWS API.

Literatúra

[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.

ZAMERANIE VZDELÁVANIA NA MERANIE ENERGETICKEJ ÚČINNOSTI POČAS TESTOVANIA SOFTVÉRU

Poslaním učiteľov softvérového inžinierstva je pripraviť budúcich softvérových inžinierov, ktorí dokážu zvládnuť každý problém počas celého životného cyklu softvéru. Okrem zručností súvisiacich s porozumením potrieb zúčastnených strán a zložením zodpovedajúceho pracovného softvéru zohráva dôležitú úlohu aj schopnosť kontrolovať správnosť výsledkov.

V tomto článku sa zameriavame na druhú skupinu zručností. Zameriavame sa na testovanie, kde úroveň automatizácie je menej dôležitá. Zdôrazňujeme meraciu povahu testovania, presnejšie sa zameriavame na softvérové meranie spotreby energie, ktoré sa môže poskytnúť pri testovaní na rôznych úrovniach. Všetky tieto aspekty sú prezentované z pohľadu pedagóga softvérového inžinierstva s cieľom predstaviť, ako vytvoriť laboratórnu reláciu softvérového inžinierstva zameranú na meranie energetickej účinnosti počas testovania softvéru, as pripomienkami autorov k tomuto návrhu.

DEFINÍCIA PROBLÉMU

Jednou z výziev pre výrobcov batérií je to, ako dlho môže batéria pracovať bez toho, aby bola znovu nabitá, a samozrejme existuje mnoho ďalších výziev, ako je veľkosť, ktorá výrazne ovplyvňuje tvar zariadenia, a ďalší faktor, ktorý je dôležitou vlastnosťou batérie je hmotnosť. Batéria je v porovnaní so zariadením, ktoré potrebuje na prevádzku batériu, o niečo ľahšia. Výzvou je, ako zmenšiť jej veľkosť a hmotnosť a pritom dosiahnuť vysokú účinnosť, pokiaľ ide o čas prevádzky mobilného zariadenia bez nabíjania.

Na vrchole tejto hardvérovej výzvy existuje jej bratská výzva v oblasti softvéru, konkrétne že softvér by mal podporovať úspory energie. Toto sa bez obmedzenia používateľského komfortu v súčasnosti považuje za nemý, ale dôležitý cieľ každého vývoja softvéru, ktorý je zameraný na akýkoľvek druh prenosného zariadenia.

Ak si spomenieme na to, že spotreba energie akéhokoľvek mobilného zariadenia je ovplyvnená počnúc bežiacimi aplikáciami cez úroveň prístupu k základným službám až po skutočnú náladu používateľa, vývoj softvéru pre tieto zariadenia je už výzvou [1]. Dalo by sa povedať, že výzva na vývoj softvéru je vždy rovnaká, ale musíme zdôrazniť „mobilitu“ ako kľúčovú vlastnosť systému. Stav nabitia batérie tiež určuje výkon systému vďaka konfigurácii na úrovni operačného systému - známe ako „preferencie úspory energie“.

Je to ešte ťažšie, ak človek (v našom prípade učiteľ) musí pripraviť študentov na takéto výzvy. Všetky už známe „osvedčené postupy“ a „tipy na úsporu energie“ musia byť prezentované v kontexte, ktorý je pre študentov ľahko zrozumiteľný.

V našom prípade to možno dosiahnuť umiestnením konceptov do známeho prostredia, ako je testovanie softvéru a automatizácia testov [2]. To je to, na čo sa v tomto dokumente zameriame, to je obsah nadchádzajúcich oddielov, počnúc návrhom, po ktorom nasleduje hodnotenie a ukončenie ďalšími tipmi na zlepšenie.

NÁVRH RIEŠENIA

Ako bolo uvedené vyššie, musíme nájsť najlepšie vhodné prostredie na zavedenie postupov merania spotreby energie [3] a postupov hodnotenia energetickej účinnosti.

Mohlo by to byť počiatočný vývoj softvéru aj vývoj softvéru, keďže obe vývojové fázy životného cyklu generického softvéru ponúkajú príležitosti na meranie vyvíjaného / vyvíjaného produktu [4].

Výhodou počiatočného vývoja softvéru je, že všetky činnosti by sa mohli zamerať na problémy spojené s úsporou energie, zatiaľ čo výber alternatívy vývoja softvéru ponúka možnosť vyhodnotiť zlepšenie v implementácii produktu. Na druhej strane evolučný vývoj softvéru vyžaduje existenciu funkčného softvéru na jeho začiatku, zatiaľ čo počiatočný vývoj softvéru je proces, ktorý vytvára produkt počnúc prvou požiadavkou na softvér.

Najlepším riešením by bolo použitie obidvoch prístupov do výučbového prostredia. Jeden semester pre počiatočný vývoj softvéru a druhý pre evolúcia toho istého softvéru. Učiteľ zvyčajne nemá dva semestre v rade, aby mohol prezentovať obsah kurzu spôsobom uvedeným v predchádzajúcom odseku. To je dôvod, prečo sa musíme rozhodnúť, aký vývoj sa má použiť na predstavenie vybraných postupov. Z hľadiska architektúry vývojového procesu a skutočnosti, že počiatočný vývoj by mohol byť tiež evolučný, sme sa rozhodli pre evolúciu softvéru. Zahŕňa veľa činností počiatočného vývoja (okrem včasného zhromažďovania a analýzy požiadaviek) a zdôrazňuje význam testovania a hodnotenia.

Táto voľba umožňuje učiteľovi:

- aby sa študenti obzerali späť vo svojej histórii rozvoja (minulé projekty), aby boli ku sebe kritickí.

- aby boli nútení vyhodnotiť svoje výsledky pomocou metrík kódu a merania spotreby energie (alebo odhadu).

- začleniť vyššie uvedené činnosti do štandardných procesov overovania a validácie vývoja softvéru.

Programovací jazyk vývoja nie je dôležitý, preto si študent môže zvoliť ktorýkoľvek zo svojich predchádzajúcich projektov pre vývoj - alebo všetky z nich, ak súťažia v počte vyvinutých projektov alebo použitých programovacích jazykoch. Programovací jazyk však zvyčajne určuje alebo obmedzuje použité vývojové prostredie a nástroje. Výber týchto nástrojov a ich doplnkov tiež poskytuje dobrú podporu pre hodnotenie metrík kódu. Meranie spotreby energie a hodnotenie energetickej účinnosti si zvyčajne vyžaduje iný nástroj, pretože existuje len málo vývojových prostredí, ktoré doteraz integrujú meranie alebo odhad spotreby energie. Pokiaľ ide o testovanie ako základ merania, musíme poznamenať, že analýza statického kódu sa používa ako súčasť testovania softvéru. Okrem toho sa vybrané časti aplikačnej kódovej základne vykonávajú počas testovania v bielych skriniach (hlavne testovanie jednotiek), ktoré môžu malým rozšírením merania spotreby energie predstavovať spotrebu energie v testovacom prípade - nepriamy pohľad na spotrebu energie testovaný kód. Počas testovania čiernych skriniek sa testuje celá aplikácia pomocou testovacích scenárov. Tieto testovacie scenáre sú porovnateľné najmä s prípadmi zamýšľaného celodenného používania softvéru, iné predstavujú hraničné scenáre - vrátane scenárov, keď je užívateľ v zlej nálade. Meranie spotreby energie pri vykonávaní týchto testov na čiernu skrinku potom poskytuje približný (ale priamy) pohľad na energetickú spotrebu produktu.

Toto je hlavná výhoda evolučného vývoja (v porovnaní s počiatočným vývojom softvéru). Učiteľ môže pripraviť počiatočnú verziu pre vývoj, vrátane vylepšeného kódu, zoznamu známych chýb a testovacej základne! Existencia testu na opakované testovanie a regresné testovanie je tu veľmi dôležitá, pretože produktivita evolúcie sa môže zvýšiť touto vlastnosťou.

Za predpokladu, že boli vykonané všetky vyššie uvedené kroky, vyzerá integrácia meraní energetickej účinnosti do procesu testovania z pohľadu aktivít študenta nasledovne:

1. Výber produktu, ktorý by mohol byť vaším minulým projektom alebo z úložiska.

2. Jeho vyhodnotenie pomocou statickej analýzy kódu, testovania, merania energie, prieskumu použiteľnosti atď.

3. Jeho vylepšenie (rôzne druhy evolúcie, ako napríklad pridanie / zmena funkčnosti, oprava alebo prispôsobenie)

4. Znova-vyskúšanie, či boli odstránené chyby alebo porucha

5. Regresná skúška (vrátane prehodnotenia)

6. Uzatvárať výsledky (urobiť konečné rozhodnutie vo všetkých dostupných údajoch vrátane energetickej účinnosti).

DISKUSIA

Pretože meranie spotreby energie je relatívne nové v porovnaní s inými technikami, ako je analýza statického kódu, testovanie čiernobielych skriniek a ladenie, môže to byť bod zlyhania. Ak sa však skombinuje s týmito staršími princípmi a vytvorí pre každého študenta zložené skóre, kritická vlastnosť je oveľa nižšia.

Pri pohľade na možnosti klasifikácie nájdeme rôzne „úrovne slobody“, ktoré sa môžu používať samostatne alebo ako súčasť kvalitatívnej kompozície:

1. počet rôznych projektov,

2. počet použitých / použitých programovacích jazykov,

3. kódová kvalita konečného produktu,

4. energetická účinnosť konečného výrobku,

5. zlepšenie kvality kódu počas vývoja softvéru,

6. zvýšenie energetickej účinnosti počas vývoja softvéru.

Berúc do úvahy všetky „úrovne slobody“ pri plnení úloh, pre konkurenčných študentov by sa mohlo definovať veľa súťaží, zatiaľ čo realizátori zbierajú „malé víťazstvá“ v množstve projektov, cieľom perfekcionistov bude optimalizovať všetky metriky kódu a minimalizovať energiu spotreba. Pre priemerného študenta by mohlo byť dosiahnuteľným cieľom zlepšenie energetickej účinnosti a zrozumiteľnosť kódu.

Študentskú súťaž možno ešte viac podporiť tým, že sa im nedovolí vrátiť sa k svojim minulým projektom, ale umožní im vybrať si z vybraného úložiska.

Rovnako ako v prípade počítačových hier, všetky úrovne hry sú potom rovnako dostupné pre všetkých.

Na podporu spravodlivosti by sa tiež mohlo vytvoriť spoločné verejné fórum študentov a učiteľov.

Naša budúca práca v tejto oblasti sa zameria na konfiguráciu prenosného integrovaného vývojového, testovacieho a odhadovacieho prostredia spotreby energie.

Toto prostredie sa použije v rámci vývoja softvéru alebo predmetov počiatočného vývoja na podporu vzdelávania v oblasti merania energetickej účinnosti počas testovania softvéru. Môže to obmedziť kreativitu študentov tým, že ponúka polouzavreté pieskovisko, pretože hardvér hrá v súčasnej architektúre odhadu spotreby energie veľmi dôležitú úlohu. Cieľom niektorých výskumov je toto obmedzenie porušiť - tešíme sa na tieto výsledky, aby boli tiež integrované.

Literatúra

[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.

SMEROM K INŽINIERSKEJ DISCIPLÍNE PRE ZELENÝ SOFTVÉR

Táto technická správa opisuje výskum vyvinutý v Zelenom softvérovom laboratóriu na univerzitách v Coimbre a Minho, ktorý bol predstavený na prvom školiacom stretnutí učiteľov projektu Erasmus + „Zameranie vzdelávania na zložiteľnosť, zrozumiteľnosť a správnosť pracovného softvéru“. Predstavuje ekologické hodnotenie programovacích jazykov a dátových štruktúr a techniky na lokalizáciu abnormálneho využitia energie v softvérových systémoch.

MOTIVÁCIA

Súčasné rozšírené používanie bezdrôtových, ale výkonných počítačových zariadení, ako sú smartfóny, notebooky atď., Mení spôsob, akým výrobcovia počítačov a softvéroví inžinieri vyvíjajú svoje výrobky. Čas vykonávania počítačov / softvéru, ktorý bol hlavným cieľom v minulom storočí, už nie je jediným problémom. Spotreba energie sa stáva čoraz ťažším miestom pre hardvérové aj softvérové systémy. V dôsledku toho je výskum zeleného softvéru relevantnou a aktívnou oblasťou výskumu. Táto správa stručne popisuje výskum, ktorý sa vyvíja v zelenom softvéri v Zelenom softvérovom laboratóriu (GSL). GSL pozostáva z rôznych portugalských výskumných skupín vrátane dvoch miest projektu „Zameranie vzdelávania na zložiteľnosť, zrozumiteľnosť a správnosť pracovného softvéru“. GSL je iniciatíva na vývoj techník a nástrojov zameraných na zníženie spotreby energie v rôznych počítačových systémoch (mobilné zariadenia, programy, databázy atď.).

GSL sa špecificky zameriava na softvérovú stránku, kde používa techniky (zdrojový kód) analýzy a transformácie na zisťovanie anomálií v spotrebe energie a na definovanie optimalizácie na zníženie tejto spotreby.

V minulom storočí bola účinnosť softvérového systému zameraná hlavne na čas vykonávania a efektívnosť využitia pamäte. V súčasnosti sa vývojári softvéru často pýtajú „je rýchlejší a ekologickejší program?“. Softvérový systém, ktorý ovplyvňuje jeho energetickú výkonnosť, má mnoho aspektov: programovací jazyk a jeho model vykonávania (zostavené do binárneho kódu alebo do virtuálneho počítača, interpretovaný kód, lenivý verzus prísne hodnotenie, použitie čiastočného vyhodnotenia runtime atď.). Výkonnosť pamäťového modelu a jazykových knižníc tiež ovplyvňuje výkon. Zložitosť algoritmu použitého na implementáciu požadovaného počítačového problému tiež ovplyvňuje výkon: ak implementovaný algoritmus musí urobiť viac práce, ako je nevyhnutne potrebné, použije sa viac CPU a energie.

V tomto dokumente stručne uvádzame výsledky výskumu dosiahnuté v GSL, konkrétne v analýze energetickej účinnosti programovacích jazykov (oddiel 2), knižníc štruktúry údajov (oddiel 3) a zdrojového kódu softvéru (oddiel 4).

EKOLOGICKOSŤ V PROGRAMOVACÍCH JAZYKOCH

Zaujímavou otázkou, ktorá vyvstáva pri diskusii o energii v programovacích jazykoch, je, či je rýchlejší jazyk energeticky efektívnym jazykom alebo nie. Porovnávanie jazykov softvéru je však mimoriadne zložitá úloha, pretože výkon jazyka je ovplyvňovaný kvalitou jeho kompilátora, virtuálneho stroja, zberača odpadu atď. V Green Software Laboratory sme študovali, hodnotili a porovnávali výkonnosť (celkom) 27 najčastejšie používaných softvérových jazykov. Použili sme dva rôzne úložiská počítačových problémov: Počítačový jazykový benchmark (CLBG) 3 a úložiská Rosetta Code4 [1–3]. Oba archívy definujú skupinu počítačových úloh a poskytujú implementáciu vo veľkej skupine programovacích jazykov. Zatiaľ čo program CLBG bol prispôsobený na analýzu času vykonávania jazykov v čase vykonávania, kód Rosetta bol definovaný s cieľom viac porozumieť programu.

Tieto programy sme zostavovali / vykonávali pomocou najmodernejších kompilátorov, virtuálnych strojov, tlmočníkov a knižníc pre každý jazyk. Potom sme monitorovali čas vykonávania, špičkovú a celkovú spotrebu pamäte a spotrebu energie CPU / DRAM / GPU. Vytvorili sme energetické hodnotenie 27 jazykov a tieto výsledky sme tiež analyzovali podľa typu vykonávania jazykov (kompilovaný, virtuálny stroj a interpretovaný) a použitej programovej paradigmy (imperatívny, funkčný, objektovo orientovaný, skriptovanie). Pre každý z typov vykonávania a programovacích paradigiem sme zostavili poradie softvérových jazykov podľa každého jednotlivého uvažovaného cieľa (napr. Čas alebo spotreba energie). Naše prvé experimenty ukazujú očakávané výsledky, napríklad, že jazyk C je rýchlejším a zelenším jazykom, ale ukazuje aj pomalšie jazyky, ktoré sú energeticky účinnejšie ako iné jazyky [2, 3].

ZELENOSŤ V DÁTOVÝCH ŠTRUKTÚRACH

Programovací jazyk / paradigma a jeho výkonná optimalizácia kompilátora nie sú jediným aspektom, ktorý ovplyvňuje spotrebu energie softvérového systému. V skutočnosti môže byť program efektívnejší aj „len“ optimalizáciou svojich knižníc [4,5]. Väčšina jazykov ponúka výkonné knižnice na manipuláciu s dátovými štruktúrami. V GSL sme študovali energetickú výkonnosť dvoch pokročilých dátových štruktúr široko používaných v programovacích jazykoch Java a Haskell.

V jazyku Java sme vykonali podrobnú štúdiu, pokiaľ ide o spotrebu energie v knižnici knižnice Java Collections Framework (JCF) 5. Zvážili sme obvyklé tri rôzne skupiny dátových štruktúr, konkrétne sady, zoznamy a mapy, a pre každú z týchto skupín, študovali sme spotrebu energie každej z jej rôznych implementácií a metód [4]. Táto energetická informovanosť JCF sa môže použiť nielen na riadenie vývojárov softvéru pri písaní ekologickejšieho softvéru Java, ale aj na optimalizáciu pôvodného kódu Java. Vyvinuli sme nástroj na refaktoring štruktúry Java dát s názvom jStanley, ktorý refaktoruje zdrojový kód Java, keď je k dispozícii zelenšia kolekcia [6]. Uskutočnili sme tiež počiatočné hodnotenie so 7 verejne dostupnými projektmi Java, kde sme dokázali zvýšiť spotrebu energie medzi 2% a 17%.

V jazyku Haskell sme študovali spotrebu energie Edison6, plne zrelej a dobre zdokumentovanej knižnice čisto funkčných dátových štruktúr [7]. Edison poskytuje rôzne funkčné dátové štruktúry na implementáciu troch typov abstrakcií: sekvencie (zoznamy, fronty a vklady), zbierky (množiny a hromady) a asociatívne zbierky (mapy a konečné vzťahy). Analyzovali sme 16 implementácií takýchto dátových štruktúr pri meraní podrobných energetických a časových metrík [5]. Ďalej sme skúmali dopad spotreby energie pomocou rôznych optimalizácií kompilácie. Dospeli sme k záveru, že spotreba energie je priamo úmerná času vykonávania a že spotreba energie DRAM predstavuje medzi 15 a 31% celkovej spotreby energie. Nakoniec sme tiež dospeli k záveru, že optimalizácia môže mať pozitívny alebo negatívny vplyv na spotrebu energie.

ZELENOSŤ V ZDROJOVOM KÓDE

Spotrebu energie neovplyvňujú iba knižnice jazykov a dátových štruktúr, kľúčovú úlohu pri efektívnosti programov zohrávajú aj algoritmy a postupy programovania. V GSL sme upravili dobre známe techniky lokalizácie porúch, aby sa v zdrojovom kóde aplikácií [8–11] staticky lokalizovali „úniky energie“ (videné ako energetická neefektívnosť, teda energetické chyby). Definovali sme lokalizáciu úniku energie na báze SPELL, aby sme určili červené (energeticky neefektívne) oblasti v softvéri. Prvá experimentálna štúdia ukazuje, že odborní programátori, ktorí mali prístup k detekcii únikov energie prostredníctvom spoločnosti SPELL, boli schopní lepšie optimalizovať spotrebu energie programov (medzi 15% a 74%) ako odborníci bez informácií alebo informácií poskytnutých štandardné programy (runtime) profiler. Študovali sme tiež energetické správanie programov C / C ++ [12].

Rozšírené používanie bezdrôtových zariadení a príchod internetu vecí mení spôsob, akým softvéroví inžinieri vyvíjajú svoj softvér. Softvér musí bežať na rôznych mobilných zariadeniach a spotreba energie je hlavným problémom pri vývoji softvéru. Línie softvérových produktov (SPL) sa objavili ako dôležitá disciplína softvérového inžinierstva, ktorá umožňuje vývoj softvéru, ktorý zdieľa spoločnú skupinu funkcií. V GSL sme definovali techniky statickej analýzy na zdôvodnenie spotreby energie v SPL na základe podmieneného zostavenia.

Takéto techniky umožňujú vývojárom softvéru identifikovať (ne) zelené výrobky a / alebo vlastnosti v SPL [13].

Android je široko používaný ekosystém pre bezdrôtové zariadenia a aktívna oblasť výskumu je analýza a optimalizácia softvérovej energie. Tím GSL vyvinul niekoľko techník [14,15] a nástroje na analýzu a optimalizáciu spotreby energie v zdrojovom kóde aplikácií pre Android [16, 17].

V súčasnosti sa väčšina údajov uložených v našich mobilných zariadeniach (súbory, fotografie, videá) ukladá aj v cloude poskytovanom ekosystémom operačného systému zariadenia. Takéto cloudové systémy sú dátové centrá, ktoré denne prevádzkujú veľké množstvo procesov dotazovania údajov, monitorujú a kontrolujú vysoko sofistikované systémy správy databáz, ktoré sú zodpovedné za vytvorenie efektívnych plánov spracovania dotazov na ich podporu. Databázové systémy sa zvyčajne spoliehajú na plány, ktoré optimalizujú čas odozvy. Navrhli sme a vyvinuli alternatívnu metódu na definovanie plánov spotreby energie pre databázové dotazy [18, 19]. Naše prvé experimentálne výsledky ukazujú, že použitie optimalizačnej heuristiky umožňuje významné zisky, pokiaľ ide o spotrebu energie a čas strávený vykonaním dotazov.

ZÁVERY

Táto technická správa opisuje výskum vyvinutý v Zelenom softvérovom laboratóriu GSL, konkrétne zelené poradie programovacích jazykov a dátových štruktúr, techniky zisťovania energetickej neefektívnosti v zdrojovom kóde softvérového systému a energeticky uvedomelý plán vykonávania dotazov pre databázové systémy.

Literatúra

[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

VÝUČBA PROGRAMOVANIA ZAMERANÉHO NA ÚLOHY

Na zimnej škole 3COWS sa uskutočnia dva návody o programovaní zameranom na úlohy a konkrétnych systémoch založených na tomto paradigme. Systém iTask ponúka webové rozhranie, aby videli svoje úlohy a zdieľali svoj pokrok s týmito úlohami. Systém mTask používa rovnaké koncepty na špecifikovanie úloh vykonávaných mikroprocesormi. Týmto príspevkom odôvodňujeme rozhodnutia učiteľov v Amsterdame o tom, čo a ako budú tieto témy prezentované na zimnej škole. Vzhľadom na obmedzený čas a rozmanité zázemie publika sa zameriame na praktické využitie paradigmy. Sotva je čas zaoberať sa výzvami a krásou výstavby týchto systémov.

ÚVOD

Task Oriented Programming, TOP, je štýl programovania zameraný na koncept úloh vykonávaných ľuďmi a strojmi. Tieto úlohy sú špecifikované bežnými funkciami vo funkcionálnom programovacom jazyku. Vo všetkých našich príkladoch použijeme Clean [6]. Sémantika úloh je úplne odlišná od jednoduchého vyhodnotenia funkcie. Úloha sa vyhodnocuje znovu a znovu, kým nevytvára stabilnú hodnotu alebo jej výsledok už nie je potrebný. Priebežné výsledky úloh môžu pozorovať ďalšie úlohy. Úlohy môžu pozostávať z kombinátorov úloh.

Na zimnej škole pre skladateľnosť, zrozumiteľnosť, zimu korektnosti v Košiciach sa uskutočnia dve prednášky TOP: Prečo je dôležité „programovanie zamerané na úlohy“ a Funkcionálne programovanie zariadení. Obe stretnutia budú pozostávať z prednášky a praktickej práce účastníkov. V tomto článku motivujeme rozhodnutia o obsahu a organizácii týchto stretnutí po diskusiách na príprave učiteľov.

PUBLIKUM

Zimná škola Skladateľnosť, Zrozumiteľnosť, Správnosť je určená pre študentov bakalárskeho, magisterského a doktorandského štúdia, ako aj pre učiteľov. Diskusie pri príprave učiteľov odhalili, že skúsenosti s funkcionálnym programovaním sú rôzne, a to z hľadiska množstva jazykových skúseností. Používané jazyky siahajú od čistých a lenivých jazykov, ako sú Haskell a Clean, po Erlang, Scheme a Scala.

To znamená, že nemôžeme predpokladať solídny spoločný funkcionálny programovací zážitok. Len časť publika bude oboznámená s pojmami ako silné písanie, funkcie vyššieho poriadku, triedy konštruktorov, monády a generiká. Aj keď tieto témy sú stavebnými kameňmi TOP, nemôžeme predpokladať, že ich poznajú všetci účastníci.

PROGRAMOVANIE ZAMERANÉ NA ÚLOHY

Programovanie zamerané na úlohy je založené na malom počte primitívnych úloh špecifických pre jednotlivé domény. Tieto primitívne úlohy typicky interagujú s prostredím, napr. Ľudia vykonávajúci časť úloh alebo hardvér interagujúci s fyzickým svetom. Kombinátory úloh sa používajú na zostavenie úlohy z menších úloh.

Úlohy môžu komunikovať prostredníctvom svojich výsledkov, ako aj prostredníctvom zdieľaných zdrojov údajov (SDS). Takýto KBÚ obsahuje zadané údaje, ku ktorým je možné pristupovať prostredníctvom primitív, ako je get a set. Tieto primitívy pôsobia na stav úlohy, aby zabezpečili referenčnú transparentnosť.

Za účelom opätovného použitia dátových typov a výpočtov existujúceho jazyka je TOP systém často konštruovaný ako doménovo špecifický jazyk, DSL, zabudovaný do existujúceho (funkcionálneho) programovacieho jazyka.

Systém iTask bol prvou implementáciou TOP [5]. Je to DSL zabudované do funkčného programovacieho jazyka Clean. Systém iTask uľahčuje interakciu s ľudským pracovníkom generovaním webových stránok podľa typu. Tieto stránky sa zobrazujú v jednom z existujúcich prehliadačov. Stránka poskytuje informácie o aktuálnych úlohách pre používateľa. Užívateľ môže pracovať so systémom iTask vyplnením formulárov a stlačením tlačidiel.

POUŽITÉ TECHNIKY

Systém iTask prechádza stavom veľmi podobným štátnemu monadu. Operátory na návrat, väzba (>> =) a sekvencia (>> |) sú veľmi podobné známym monadickým verziám [4,7]. Vyžaduje si to funkcie vyššieho poriadku a užívateľom definované operátory infixu. Na opätovné použitie symbolu operátora pre rôzne monády sú definované ako triedy konštruktorov typov.

Kombinátory úloh sú všetky funkcie vyššieho rádu, zvyčajne užívateľom definované operátory infixu, manipulácia s výsledkami úloh a stav globálnej úlohy. Špecifické pre TOP je to, že úlohy vedú k priebežným výsledkom, ktoré je možné pozorovať, zatiaľ čo sa tieto úlohy opakujú znova a znova, až kým neprinášajú stabilný výsledok alebo ak sa ich výsledok už nepoužíva. Vyžaduje si to funkcie vyššieho rádu, lenivosť a automatický zber odpadu.

Systém iTask generuje webový server, ktorý používatelia používajú na nájdenie riešenia svojej úlohy. Rovnako ako všetky webové servery, aj to vyžaduje serializáciu a deserializáciu stavu na uloženie a získanie aktuálneho stavu. Vyplnením webových formulárov pre ľubovoľné algebraické typy údajov používatelia naznačujú, ako postupujú v rámci úloh. Všetky tieto funkcie sú implementované pomocou všeobecného programovania.

Na efektívne vykonávanie úloh monitorujúcich hodnotu SDS je pre každú SDS skrytý systém publikovania a prihlásenia, ktorý aktivuje úlohy pomocou tohto SDS pri aktualizácii jeho hodnoty.

Okrem týchto vlastností systém iTask využíva mnoho ďalších techník. Napríklad vykonávanie častí úloh v prekladači bežiacom v prehliadači, aby sa zabezpečila rýchla reakcia systému na vysoko interaktívne úlohy, ako je napríklad kresba.

VYUČOVANIE NA ZIMNEJ ŠKOLE

Akákoľvek výučba programovania vyžaduje praktické skúsenosti s programovaním pomocou vzdelaných techník na ich zvládnutie. To platí aj pre TOP. V dôsledku toho rozdelíme štyri hodiny, ktoré sú k dispozícii na to, prečo je „Programovanie zamerané na úlohy“ dôležité, do dvoch častí takmer rovnakej veľkosti.

V prvej časti načrtneme pojem TOP pomocou systému iTask. Vzhľadom na pozadí väčšiny študentov musíme preskočiť takmer všetky podrobnosti o implementácii systému a musíme sa zamerať na využívanie knižnice. Táto knižnica je vlastne plytkým vloženým DSL pre TOP. V prednáške používame tento súbor primitívov bez toho, aby sme trávili veľa času vysvetlením jeho architektúry a implementácie.

Pre praktickú prácu rozdelíme existujúci základný príklad projektu do súboru malých nezávislých TOP projektov. Úlohy budú pozostávať z malých variácií týchto projektov, aby sa zakúsila chuť programovania zameraného na úlohy.

Skúsenejší funkcionálni programátori môžu preskočiť väčšinu základných cvičení a skočiť priamo na pokročilejšie zadanie. Týmto spôsobom sa budeme môcť prispôsobiť individuálnym schopnostiam každého účastníka.

SYSTÉM MTASK

Mikroprocesory sú počítačové systémy s veľmi obmedzenými výpočtovými schopnosťami. Zvyčajne majú pomerne nízku taktovaciu frekvenciu a závažné obmedzenia pamäte, ako napríklad niekoľko kB pamäte na ukladanie údajov spusteného programu. Tieto lacné procesory sú hnacou silou mnohých prvkov internetu vecí, internetu vecí. V takýchto mikroprocesorových systémoch je obvykle potrebné monitorovať niekoľko vstupných portov, ako aj riadiť niektoré výstupy na základe týchto pozorovaní. Kvôli obmedzeniam hardvéru zvyčajne neexistuje žiadny operačný systém ponúkajúci podporu.

Model paradigmy TOP poskytuje ľahké vlákna, ktoré sú veľmi vhodné na monitorovanie a koordináciu postupu takýchto dobre definovaných jednoduchých úloh. Tieto úlohy môžu bežať vlastnou rýchlosťou, zatiaľ čo kombinátory a zdieľané zdroje údajov sa používajú na ich koordináciu. Spustenie systému iTask na zariadeniach IoT by nám umožnilo zostaviť programy, ktoré sa čiastočne vykonávajú na webovom serveri aj na zariadeniach IoT. Obmedzenia mikroprocesorov znemožňujú spustenie plnohodnotného programu iTask na zariadeniach IoT.

Na priblíženie ideálneho riešenia sme vyvinuli systém mTask [3, 2]. Jedná sa o povrchovo vloženú DSL, ktorá môže byť použitá ako súčasť systému iTask. Podporuje TOP paradigmu vrátane rovnakých výsledkov úloh ako systém iTask, kombinátory úloh a zdieľané zdroje údajov.

Z hľadiska konštrukcie nemá tento DSL funkcie vyššieho poriadku a žiadne rekurzívne dátové typy. Kvôli obmedzeniam uloženým v tomto DSL môžu byť programy mTask kompilované do kódu, ktorý sa spúšťa na mikroprocesoroch. Napriek týmto obmedzeniam je DSL veľmi vhodné ľahko a veľmi stručne špecifikovať úlohy, ktoré sa majú vykonať na zariadeniach IoT.

POUŽITÉ TECHNIKY

Systém mTask je plytkým vloženým DSL s viacerými pohľadmi založeným na triedach konštruktorov typu [1]. Každá inštancia týchto tried definuje interpretáciu programu nazývaného pohľad, skonštruovaného z týchto primitívov. Typické pohľady implementujú peknú tlač, generovanie kódu pre mikroprocesory a simuláciu programov mTask ako program iTask. Konštrukcia DSL je rozšíriteľná tak, aby bolo možné znova použiť existujúce knižnice pre periférie, ako sú snímače teploty, displeje a servomotory. Je jednoduché pridať takú knižnicu ako jazykový primitív do systému mTask zavedením novej triedy konštruktorov a požadovaných inštancií.

Aby sa implementácia mTask stala prenosnou pre mnoho rôznych mikroprocesorov a aby sa uľahčilo opätovné použitie existujúcich knižníc C ++, pohľad generovania kódu vytvorí kód C ++ pre platformu Arduino namiesto natívneho strojového kódu pre niektoré špecifické procesory. Kompilátor avr-gcc vnútri platformy Arduino môže prekladať vygenerovaný kód C ++ a knižnice použité na natívny kód pre mnoho rôznych mikroprocesorov.

VYUČOVANIE NA ZIMNEJ ŠKOLE

Schopnosť vykonávať vysoko kvalitné programy na malom mikroprocesore, ktorý spolupracuje s periférnymi zariadeniami, je príťažlivý pre tutorial v zimnej škole. Vykonanie programu mTask na skutočnom mikroprocesore si však vyžaduje veľa študentov; musia zostaviť program mTask vo vnútri systému iTask, spustiť program iTask na získanie kódu C ++, vložiť tento kód C ++ do Arduino IDE, pripojiť Arduino IDE k mikroprocesoru a vybrať správne možnosti, naložiť kompilovaný program do mikroprocesor a nakoniec spustiť. Všetky tieto kroky sú pomerne ľahké, ale celý proces vedie iba k výsledku, keď je každý krok vykonaný správne.

Pretože vygenerovaný program bude bežať na mikroprocesore bez operačného systému a ladenie veľmi obmedzených vstupných / výstupných periférií je taký program náročný. Po rozsiahlej diskusii sa táto postupnosť krokov považovala za príliš ambicióznu pre daný čas a publikum.

Našťastie, simulátorový pohľad na systém mTask ponúka alternatívu, ktorá sa omnoho ľahšie používa. Toto zobrazenie transformuje program mTask na bežný program iTask. Simulátor ponúka postupné vykonávanie programu mTask. Zobrazuje stopu posledného kroku poslednej vykonanej úlohy a stav všetkých periférnych zariadení a zdieľaných zdrojov údajov. Hodiny, hodnota SDS, ako aj stav periférií sa môžu interaktívne meniť, aby sa dalo riadiť vykonávanie a skúmať rôzne scenáre. Vďaka tomu je praktická práca tohto tutoriálu priamym nástupcom praktickej práce predchádzajúceho tutoriálu iTask.

Bolo rozhodnuté naplánovať tieto príručky na jeden deň s prednáškou iTask as ňou spojenou praktickou prácou ráno a tutoriálom mTask popoludní. Týmto spôsobom môže relácia mTask priamo stavať na znalostiach a zručnostiach získaných v relácii iTask. Pochopenie TOP, ktoré bolo uvedené ráno, sa prehĺbi popoludní.

ZÁVER

Pre obidva návody na tému TOP existuje omnoho viac zaujímavých tém, ako je možné pokryť v danom čase pre divákov zimnej školy. Na stretnutiach sa zameriame na porozumenie a používanie TOP paradigmy relatívne jednoduchými príkladmi. Na ilustráciu schopností tohto prístupu sa použijú mierne pokročilé príklady. V praktickej práci sa zameriame na ilustračné cvičenia, ktoré sú väčšinou variantmi príkladov použitých v návode. Pre pokročilých účastníkov bude existovať niekoľko náročných úloh, ako aj príležitosť dôkladne prediskutovať aspekty systémov.

Literatúra

[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

INTERAKTÍVNY PRÍSTUP K VÝUČBE FARBENÝCH PETRIHO SIETÍ

Formálne metódy patria k technikám, ktoré pri správnom použití môžu významne prispieť ku korektnosti vyvíjaného softvérového či hardvérového systému. Jednou z vhodných formálnych metód pre systémy so súbežným alebo nedeterministickým správaním je modelovací jazyk Farbené Petriho siete. Tento príspevok opisuje výučbovú aktivitu, zameranú na vysvetlenie základných princípov tohto jazyka a niektorých jeho vlastností, súvisiacich s funkcionálnym programovaním. Aktivita trvala dve a pol hodiny a zahŕňala interaktívnu tvorbu modelu s aktívnym príspevkom jej účastníkov.

ÚVOD

Vzhľadom na rastúcu závislosť súčasnej ľudskej spoločnosti od počítačových systémov by mala byť ich korektnosť nanajvýš dôležitá. Jedným z prístupov, ktoré môžu významne prispieť k správnosti, je využitie formálnych metód počas vývoja softvéru a hardvéru. Formálna metóda je matematicky založená technika, ktorá poskytuje formálny jazyk s jednoznačne definovanou syntaxou a sémantikou a prístroj, ktorý umožňuje vykonávať overovacie, vývojové a simulačné úlohy so systémovými špecifikáciami napísané v jazyku. Jedným z významných členov rodiny formálnych metód je modelovací jazyk Colored Petri Nets (CPN). CPN [4, 3] kombinujú formalizmus Petriho sietí [1] s funkčným jazykom na manipuláciu s údajmi a na rozhodovacie postupy. Funkčný jazyk sa nazýva CPN ML a je to mierne upravená verzia štandardu ML [2, 5]. Softvér CPN Tools [6] podporuje jazyk CPN a príslušné špecifikácie, overovacie a simulačné úlohy.

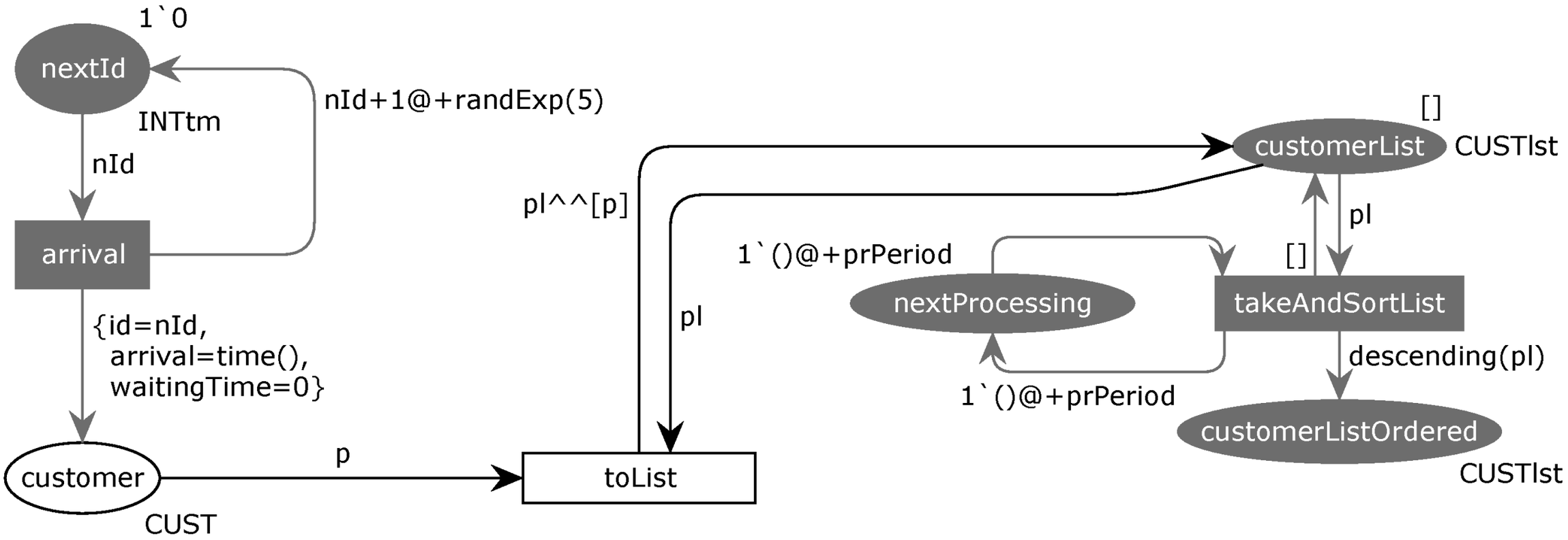
Viac ako desať rokov sú CPN súčasťou vysokoškolských kurzov týkajúcich sa formálnych metód, modelovania a simulácie v domácej inštitúcii autora. Jednou z metód, ktorú autor používa pri vysvetľovaní koncepcií CPN, je interaktívny prístup s aktívnou účasťou publika. Publikum tu vyberie doménu a proces, pre ktorý bude navrhnutý model CPN, a pomáha vytvárať jeho vybrané časti. Skúsenosti z konkrétneho zavádzania tohto prístupu do vzdelávacích aktivít pre vysokoškolských učiteľov sú opísané vo zvyšku tohto dokumentu.

VÝCVIKOVÁ AKTIVITA S INTERAKTÍVNYM VYTVORENÍM MODELU CPN

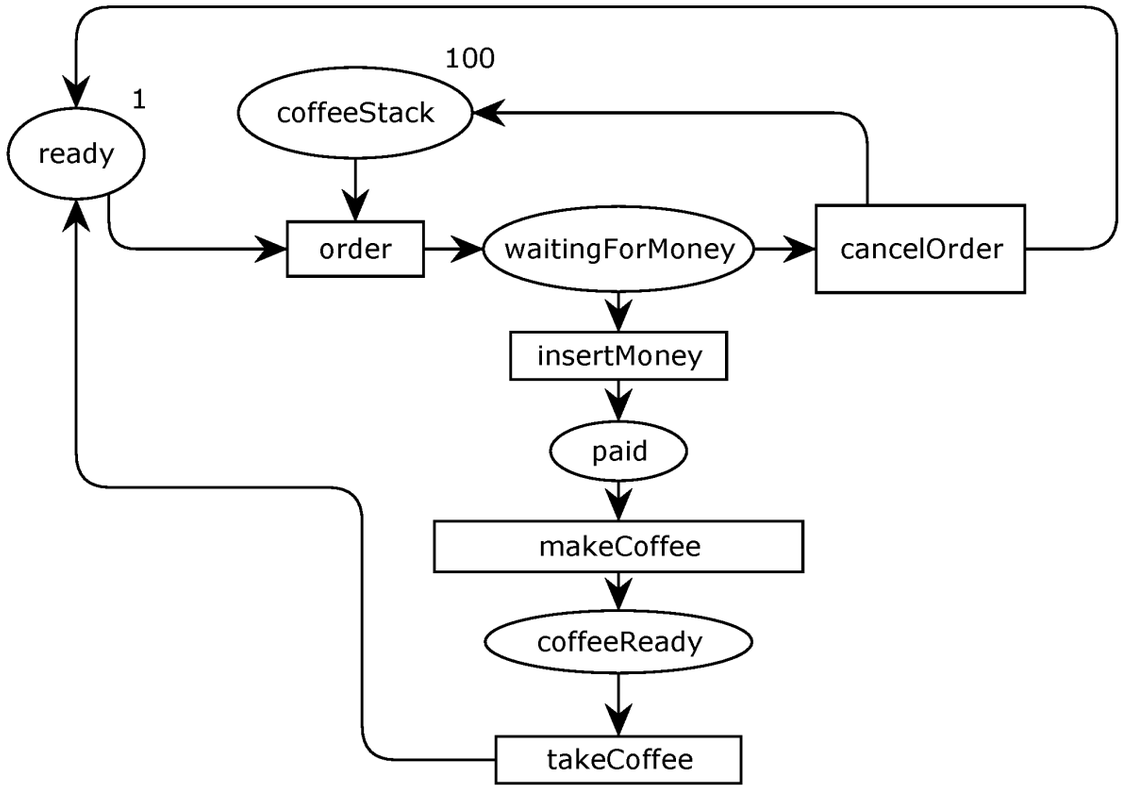
Školiaca aktivita sa zorganizovala pre približne 10 účastníkov, ktorí boli univerzitnými učiteľmi s určitými funkčnými jazykovými znalosťami. Účastníci sa neobmedzili na žiadne predchádzajúce znalosti CPN. Celková doba aktivity bola asi 2,5 hodiny, bez prestávok, a bola rozdelená do troch fáz.

Prvá fáza trvala asi 30 minút a vysvetlila základné princípy CPN. Konkrétne, táto CPN má grafický tvar, bipartitný graf s dvoma typmi vrcholov: miesta, nakreslené ako elipsy a prechody, nakreslené ako obdĺžniky. Miesta obsahujú tokeny, ktoré predstavujú stav siete a prechody možno chápať ako udalosti, ktoré menia stav spotrebovaním existujúcich tokenov a vytváraním nových tokenov.

Druhá a tretia fáza boli venované vytvoreniu modelu CPN. Keďže jedným z cieľov aktivity bolo ukázať, ako sa v modeloch CPN dajú použiť niektoré vyspelejšie štandardy ML, konkrétne štruktúry a funktory, dostali účastníci predtým, ako sa tieto pojmy používali, štartovací model CPN, ktorý už tieto pojmy používal. druhá fáza začala. Štartovací model je znázornený na obr. 1. Časť pozostávajúca z uzlov nextId, Arr a Customer predstavuje prichádzanie zákazníkov, ktorí prichádzajú jeden po druhom, aby ich obsluhovali. Samotná obsluha nie je v modeli štartéra uvedená. Namiesto toho existuje prechod na zoznam, ktorý vezme token od zákazníka a pridá jeho hodnotu do zoznamu, ktorý sa nachádza na mieste customerList.



Prechod takeAndSortList je spustený v pravidelných intervaloch, definovaných hodnotou prPeriod. Každé spustenie takeAndSortList vyprázdni zoznam v customerList, triedi jeho obsah a uloží objednanú verziu do customerListOrdered. Miesto nextProcessing je pomocné a zaisťuje, aby sa funkcia takeAndSortList spúšťala iba v pravidelných intervaloch. Triedenie je zabezpečené funkciou zvanou zostupne, ktorá implementuje algoritmus Quicksort. Funkcia využíva štandardné štruktúry ML a funktory.



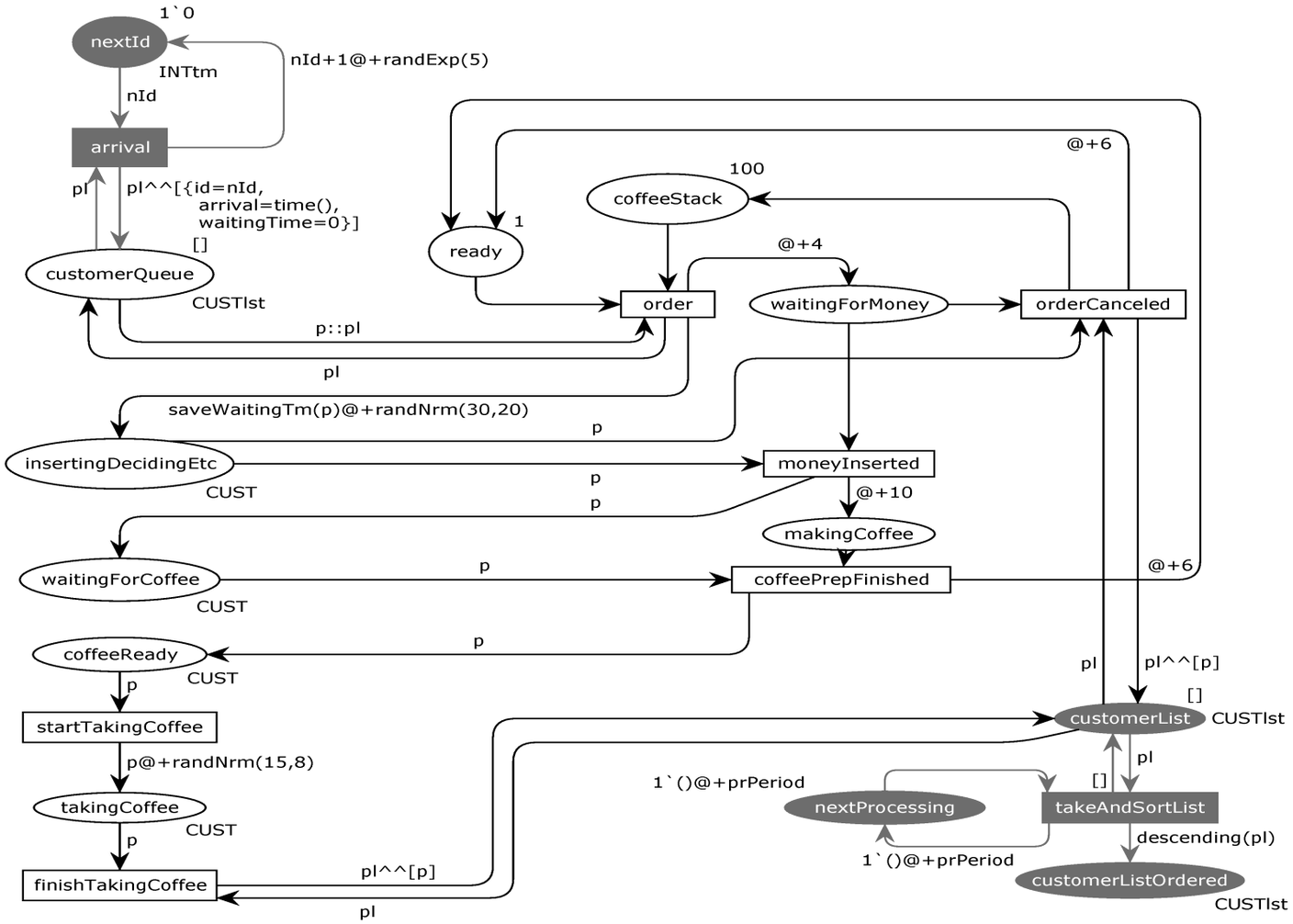
Pokiaľ ide o obsluhujúcu časť, účastníci aktivity sa rozhodli modelovať zariadenie na predaj kávy. V druhej fáze sa podieľali na vytvorení modelu CPN ktorý zachytáva základnú činnosť stroja. Model je znázornený na obrázku 2. V počiatočnom stave je stroj pripravený obsluhovať zákazníka (jeden token na mieste pripravený) a je naplnený 100 dávkami kávy (100 žetónov v coffeeStack). Podávanie začína zákazníkom, ktorý si objedná kávu vypálením prevodnej objednávky. Potom stroj čaká na ďalší krok zákazníka (token v čaká na ForMoney). Zákazník môže vložiť peniaze (spustením insertMoney) alebo zrušiť objednávku (spustením cancelOrder). Zrušenie vráti stroj do stavu „pripravený“.

Ak sú peniaze vložené, zariadenie pripraví kávu (vypálením makeCoffee). Nakoniec po odpálení takeCoffee zákazník vezme pripravenú kávu a stroj sa vráti do stavu „ready“.

Po druhej fáze došlo k prestávke asi 70 minút. Počas prestávky lektor prepojil model z fázy 2 s časťami štartovacieho modelu a pridal vrcholy a oblúky popisujúce správanie zákazníka. Opravil tiež niektoré nezrovnalosti v modeli, na ktoré poukázal jeden z účastníkov. Výsledný konečný model CPN je znázornený na obr. 3. Vrcholy prevzaté zo štartovacieho modelu (obr. 1) bez akýchkoľvek zmien sú zobrazené šedou farbou.

Miesto zákazníka je nahradené zákazníkomQueue, ktorý drží token so zoznamom hodnôt, ktorý predstavuje rad zákazníkov čakajúcich na stroj. Namiesto prechodu na zoznam je obslužná časť vytvorená z výsledku druhej fázy (obr. 2). Poskytujúca časť konečného modelu sa líši od obrázku 2 v troch kľúčových aspektoch:

- Žetóny nesú informáciu o obsluhovanom zákazníkovi a výrazy oblúka definujú trvanie zodpovedajúcich akcií.



- Opravujú sa nezrovnalosti týkajúce sa úlohy miest a prechodov. Teraz všetky prechody predstavujú okamžité udalosti. Napríklad prechodová značka Káva z obr. 2 je nahradená miestom výroby Káva a prechodová cesta Káva je nahradená vrcholmi startTakingCoffee, takingComffee a finishTakingCoffee.

- Úkony a stavy zákazníkov a stroja sa modelovajú osobitne. Vrcholy zákazníkaQueue, vloženieDecidingEtc, waiting for For Coffee, coffeeReady, startTakingCoffee, taking Coffee and finishTakingCoffee patria zákazníkom, zatiaľ čo zvyšok obsluhujúcej časti predstavuje stroj alebo obe strany. Tretia fáza vzdelávacej činnosti bola venovaná vysvetleniu konečného modelu a diskusii o mieste týchto modelov vo vývoji správnych počítačových systémov. Trvalo to asi 30 minút.

ZÁVER

Tu prezentovaná interaktívna vzdelávacia aktivita je vhodná pre krátke, intenzívne kurzy, ktoré sa často konajú počas letných škôl alebo iných podobných vyučovacích akcií. Opísaný test činnosti ukázal, že pôvodný časový dar, ktorý bol 2 hodiny, nebol dostatočný. Preto je potrebná tretia fáza, kde prednášajúci predstavil konečný model. Berúc do úvahy čas potrebný na zostavenie konečného modelu lektorom, bude vyžadovať ďalšie najmenej dve hodiny na to, aby sa celý proces tvorby modelu interaktívne vykonal so sluchou. Všetky tu uvedené alebo uvedené modely CPN je možné získať na žiadosť autora.

Literatúra

[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/

CODECOMPASS: ROZŠÍRITEĽNÝ RÁMEC NA POROZUMENIE KÓDU

CodeCompass je nástroj s otvoreným zdrojovým kódom, ktorý pomáha porozumieť veľkým starým softvérovým systémom. Na základe infraštruktúry kompilátora LLVM / Clang poskytuje CodeCompass presné informácie o zložitých jazykových prvkoch C / C ++. Široká škála interaktívnych vizualizácií zahŕňa diagramy volaní tried a funkcií; architektonické diagramy, diagramy komponentov a rozhraní a diagramy „odkazuje na“ a mnoho ďalších. CodeCompass tiež využíva informácie o zostavení na preskúmanie architektúry systému a informácie o riadení verzií, ak sú k dispozícii. Integrované sú aj výsledky statickej analýzy založené na Clang. Aj keď sa tento nástroj zameriava hlavne na C a C ++, podporuje aj jazyky Java a Python. Rámec CodeCompass, ktorý má webovú, zásuvnú a rozšíriteľnú architektúru, môže byť otvorenou platformou na ďalšie porozumenie kódu, statickú analýzu a úsilie v oblasti softvérových metrík.

ÚVOD

Oprava chýb alebo vývoj nových funkcií si vyžaduje sebavedomé porozumenie všetkým detailom a dôsledkom plánovaných zmien. Nástroje na porozumenie kódu môžu pomôcť odhaliť pôvodné úmysly a podrobnosti o implementácii vytvorením modelu zo zdrojového kódu a ďalších dostupných informácií. Aj keď je niekoľko takýchto nástrojov k dispozícii ako proprietárny alebo slobodný softvér, ich sada funkcií je obmedzená.

Na odstránenie týchto obmedzení bol vyvinutý program CodeCompass.

Projekt CodeCompass je spoločným úsilím spoločnosti Ericsson Ltd. a Eötvös Loránd University v Budapešti, ktoré pomáhajú porozumieť veľkým softvérovým systémom. Na poskytnutie presných informácií o zložitých jazykových prvkoch C / C ++, ako je preťaženie, dedičnosť, použitie premenných a typov, možné použitie ukazovateľov funkcií a virtuálnych funkcií - funkcií, ktoré rôzne existujúce nástroje podporujú iba čiastočne - je CodeCompass založený na skutočnom kompilátore, infraštruktúra LLVM / Clang. Eliminuje tak slabé stránky obvyklých nástrojov na porozumenie ľahkých váh, ako je napríklad OpenGrok.

CodeCompass sa však neobmedzuje iba na zdrojový kód. Používa informácie o zostavení systému na odhalenie architektonických spojení. Používa tiež informácie o riadení verzií, ak sú dostupné, takže je možné identifikovať spojenia medzi rôznymi zdrojovými súbormi „náhodne“ modifikovanými v rovnakom potvrdení. Na uľahčenie rýchleho a presného vnímania používa CodeCompass na porozumenie textové a grafické zobrazenie softvérového systému. Z bežných grafov volania funkcií je k jedinečným architektonickým diagramom prístupných množstvo (interaktívnych) diagramov. Pre ľahký prístup pre používateľov má CodeCompass webovú architektúru. Klientom môže byť štandardný webový prehľadávač, doplnok editora alebo akákoľvek aplikácia tretej strany. Komunikácia je založená na rozhraní REST API a dobre sa prispôsobuje požiadavkám paralelných klientov.

V tomto článku porovnáme CodeCompass s existujúcimi nástrojmi na porozumenie a opíšeme jeho sadu funkcií. V časti 2 uvádzame prehľad hlavných archetypov existujúcich nástrojov na porozumenie kódu. Predstavujeme rozšíriteľnú architektúru CodeCompass v 3. Hlavné rysy tohto nástroja sú uvedené v časti 4. Zhrnutie článku v časti 5.

SÚVISIACA PRÁCA

Na softvérovom trhu existuje niekoľko nástrojov, ktorých cieľom je porozumenie zdrojovým kódom. Niektoré používajú statickú analýzu, iné skúmajú aj dynamické správanie analyzovaného programu. Tieto nástroje možno rozdeliť do rôznych archetypov na základe ich architektúry a hlavných princípov. Na jednej strane majú nástroje architektúru server-klient. Vo všeobecnosti tieto nástroje analyzujú projekt a ukladajú všetky potrebné informácie do databázy. Klienti (zvyčajne on-line) sú obsluhovaní z databázy. Tieto nástroje sa môžu integrovať do pracovného postupu pri nočnom spustení CI.

Týmto spôsobom môžu vývojári vždy prehľadávať a analyzovať celú, veľkú, starú kódovú základňu. Existujú aj aplikácie náročné na klienta, v ktorých sa analyzuje menšia časť kódovej základne. Toto je prípad použitia pre editory IDE, kde častá zmena zdroja vyžaduje rýchlu aktualizáciu databázy o analyzovaných výsledkoch. V tejto časti uvádzame niektoré nástroje používané v priemyselnom prostredí z jednotlivých kategórií.

Woboq [3] je webový prehliadač kódov pre C a C ++. Tento nástroj má rozsiahle funkcie, ktorých cieľom je rýchle prezeranie softvérového projektu. Užívateľ môže rýchlo nájsť súbory a pomenované entity pomocou vyhľadávacieho poľa, ktoré poskytuje dokončenie kódu pre ľahkú použiteľnosť. Navigácia v kódovej základni je povolená prostredníctvom webovej stránky pozostávajúcej zo statických súborov HTML. Tieto súbory sa generujú počas procesu analýzy. Výhoda tohto prístupu spočíva v tom, že webový klient bude rýchly, pretože počas prehliadania nie je na serveri potrebný žiadny výpočet.

Ak umiestnite kurzor myši na konkrétnu funkciu, triedu, premennú, makro atď., Môžete zobraziť vlastnosti tohto prvku. Napríklad v prípade funkcií je možné vidieť jeho podpis, miesto jeho definície a miesto použitia. Pre triedy je možné skontrolovať veľkosť ich objektov, rozloženie triedy a odsadenie jej členov a dedičský diagram. Pre premenné je možné skontrolovať ich typ a umiestnenie, kde sú napísané alebo prečítané.

V makrách C a C ++ tvoria podskupinu, ktorá sa hodnotí v predkompilačnom kroku. Toto vyhodnotenie je nahradenie textových tokenov textom, čo znamená, že fáza kompilácie pracuje s iným kódom ako pôvodným. Vo Woboq je možné skontrolovať aj konečnú hodnotu rozšírení makra.

Veľmi užitočnou črtou tohto nástroja je sémantické zvýraznenie. Touto funkciou sa dajú ľahko rozlíšiť rôzne jazykové prvky: formátovanie lokálnych, globálnych alebo členských premenných, virtuálnych funkcií, typov, typedefov, tried, makier atď. Sú rôzne.

Woboq môže poskytnúť vyššie uvedené vlastnosti, pretože potrebné informácie sa zhromažďujú v skutočnej fáze kompilácie. Skúmaný projekt musí najprv zostaviť a analyzovať Woboq. Analýza sa vykonáva pomocou infraštruktúry LLVM / Clang, ktorá sprístupňuje celý abstraktný strom syntaxe. Týmto spôsobom je možné extrahovať všetky sémantické informácie s rovnakou sémantikou, akú má konečný program. Nevýhodou tohto nástroja je to, že Woboq sa dá použiť iba na prezeranie projektov C a C ++.

OpenGrok [4] je nástroj na rýchle vyhľadávanie zdrojového kódu a krížového odkazu. Na rozdiel od Woboq tento nástroj nevykonáva hĺbkovú jazykovú analýzu, preto nie je schopný poskytnúť sémantické informácie o konkrétnych entitách. Namiesto toho používa Ctags [5] na analýzu zdrojového kódu iba textovo a na určenie typu konkrétnych prvkov. Jednoduchá syntaktická analýza umožňuje rozlíšiť názvy funkcií, premenných alebo tried atď. Hľadanie medzi nimi je vysoko optimalizované, a preto veľmi rýchle aj na veľkých kódových základniach. Vyhľadávanie sa môže uskutočniť pomocou zložených výrazov (napr. Defs: target), obsahujúcich dokonca aj divoké karty, a výsledky môžu byť ďalej obmedzené na podadresáre. Okrem textového vyhľadávania existuje možnosť nájsť symboly alebo definície osobitne. Nedostatok sémantickej analýzy umožňuje Ctagom podporovať niekoľko (41) programovacích jazykov. Výhodou tohto prístupu je aj to, že je možné postupne aktualizovať databázu indexov. OpenGrok tiež poskytuje možnosť zhromažďovať informácie zo systémov na správu verzií, ako sú Mercurial, SVN, CSV atď.

Understand [6] nie je iba nástrojom na prehliadanie kódu, ale aj úplným IDE. Jeho veľkou výhodou je, že zdrojový kód je možné editovať a zmeny analýzy je možné okamžite vidieť.

Okrem funkcií prehľadávania kódov, ktoré už boli spomenuté pre predchádzajúce nástroje, poskytuje Understand množstvo metrík a správ. Niektoré z nich sú riadky kódu (celkom / priemer / maximum globálne alebo na triedu), počet združených / základných / odvodených tried, nedostatok kohézie [2], zložitosť McCabe [1] a mnoho ďalších. Treemap je bežná metóda reprezentácie všetkých metrík. Je to vnorený obdĺžnikový pohľad, kde vnorenie predstavuje hierarchiu prvkov a farebné a rozmerové rozmery predstavujú metriku zvolenú používateľom.

Pre veľké kódové základne je potrebná kontrola architektúry. Pochopenie môže ukázať diagramy závislosti založené na rôznych vzťahoch, ako je hierarchia volania funkcií, dedičnosť triedy, závislosť súboru, zahrnutie / import súboru. Používatelia môžu tiež vytvoriť svoj vlastný typ diagramu pomocou rozhrania API poskytnutého nástrojom.

V programovaní sú základné pojmy bežné vo všetkých jazykoch, ale existujú určité pojmy, ktoré sa v konkrétnom jazyku interpretujú odlišne.

Understand ovláda ~ 15 jazykov a môže poskytnúť informácie o jazyku špecifické pre daný jazyk, napr. analýza ukazovateľov funkcií v C / C ++ alebo diagramy hierarchie balíkov v Ada. Vytvára databázu zo základne kódu. Všetky informácie možno získať prostredníctvom programovateľného rozhrania API. Týmto spôsobom môže užívateľ vyhľadávať všetky potrebné informácie, ktoré nie sú obsiahnuté v používateľskom rozhraní.

CodeSurfer [7] je podobný ako Understand v tom zmysle, že ide o silnú klientskú aplikáciu statickej analýzy. Jeho cieľom je porozumieť projektom strojového kódu C / C ++ alebo x86. CodeSurfer vykonáva hĺbkovú jazykovú analýzu, ktorá poskytuje podrobné informácie o správaní softvéru. Napríklad implementuje analýzu ukazovateľov, aby skontrolovala, ktoré ukazovatele môžu ukazovať na danú premennú, uvádza výkazy, ktoré závisia na vybranom výkaze, pomocou analýzy dopadu, a pomocou analýzy toku údajov určuje, kde bola premennej priradená jej hodnota, atď.

ARCHITEKTÚRA CODECOMPASS

V predchádzajúcej časti sme uviedli niektoré aspekty týkajúce sa cieľov a architektúry nástrojov na porozumenie kódu. Teraz uvádzame, kde medzi týmito nástrojmi stojí CodeCompass.

CodeCompass má architektúru klient-server, v ktorej predstavuje informácie zhromaždené v predchádzajúcej fáze analýzy. Dôvod, prečo bola táto architektúra vybraná, pochádza z cieľa nástroja. Na rozdiel od editorov kódu sa program Compass Compass plánuje ako nástroj na porozumenie kódu. Medzi týmito dvoma prípadmi použitia sú zásadné rozdiely. Počas písania kódu manipulujú programátori iba s niekoľkými súbormi súčasne. Pri porozumení kódu je však potrebné brať do úvahy zdroje viacerých modulov prostredníctvom kódovej základne. V editore je dokončenie kódu jednou z najužitočnejších funkcií: programátor si nepamätá všetky metódy a polia triedy, ale vyžaduje od editora, aby ich uviedol. Na pochopenie kódu je potrebná široká škála vizualizácií, aby bolo možné prehľadávať vzťahy medzi časťami kódu.

Pri úprave zdroja sa programátor zameriava iba na relatívne malý fragment kódu, napríklad na funkciu alebo triedu. Pri porozumení kódu to nie je iba správanie funkcií na nízkej úrovni, ale ich závislosti a účinky sa posudzujú v kontexte modulového systému na vysokej úrovni.

Hlavné užívateľské rozhranie CodeCompass je webové. Všetky vyššie uvedené vizualizácie a funkcie sa môžu spýtať prostredníctvom verejného rozhrania API, ktoré je priradené serverovej aplikácii. Webové rozhranie sa zaoberá prípadmi použitia, ktoré sa zameriavajú na rýchle a šikovné úlohy prehliadania, kontroly a porozumenia. CodeCompass je však viac ako len nástroj na prehliadanie kódu. Je to tiež rámec, t.j. rozšíriteľný zberač a moderátor statických analytických procesov. Z tohto dôvodu nebolo zámerom vytvoriť aplikáciu náročnú pre klienta, ktorá ukladá výsledky analýzy na strane klienta, ale bola schopná uspokojiť rôzne potreby používateľov. Týmto spôsobom je možné implementovať skript, napríklad, ktorý zbiera množinu funkcií, ktoré tvoria spojenie volania funkcie, čím špecifikuje koherentnú časť softvéru.

Ďalšou požiadavkou na návrh programu CodeCompass bolo zvládnuť rozsiahle bázy kódov a stále odpovedať na požiadavky používateľov veľmi rýchlo, t. J. Najviac v sekundách. To sa dosiahne uložením všetkého množstva informácií do databázy, ktoré je dostatočné na zodpovedanie požiadaviek. Pretože sme chceli poskytnúť presné výsledky dotazov, je potrebný predchádzajúci proces analýzy. V prvom sme uložili celý abstraktný syntaxový strom zdroja, výsledkom čoho bol pomer zdrojového kódu k veľkosti databázy v pomere 1: 1000. Ukázalo sa však, že väčšina používateľov sa zaujíma iba o pomenované entity (funkcia, premenné, triedy, makrá atď.), Takže nebolo potrebné ukladať nič iné, ako napríklad kontrolné štruktúry alebo iné príkazy. Napriek tomu existujú niektoré úlohy, ktoré vyžadujú viac ako uložené informácie, napríklad algoritmus segmentovania. Ak chce užívateľ vidieť účinky zmeny hodnoty premennej, musia sa vziať do úvahy aj príkazy na zmenu stavu. Vyžaduje si to opravu kódu za chodu.

CODECOMPASS FEATURES

V tejto časti uvádzame prehľad funkcií dostupných prostredníctvom štandardného používateľského rozhrania. Pri popisovaní funkcií špecifických pre daný jazyk, ako je napríklad zoznam volajúcich metódy, vždy predpokladáme

že jazyk projektu bude C ++, pretože má najpokročilejšiu podporu v CodeCompass, ale podobné funkcie sú dostupné aj pre Java a Python.

Vyhľadávanie

Pravdepodobne najdôležitejším prípadom použitia nástroja na porozumenie kódu je vyhľadávanie. Dá sa hľadať buď súbor, alebo zdrojový kód. Na nájdenie prvkov zdrojového kódu poskytuje nástroj 3 rôzne možnosti vyhľadávania:

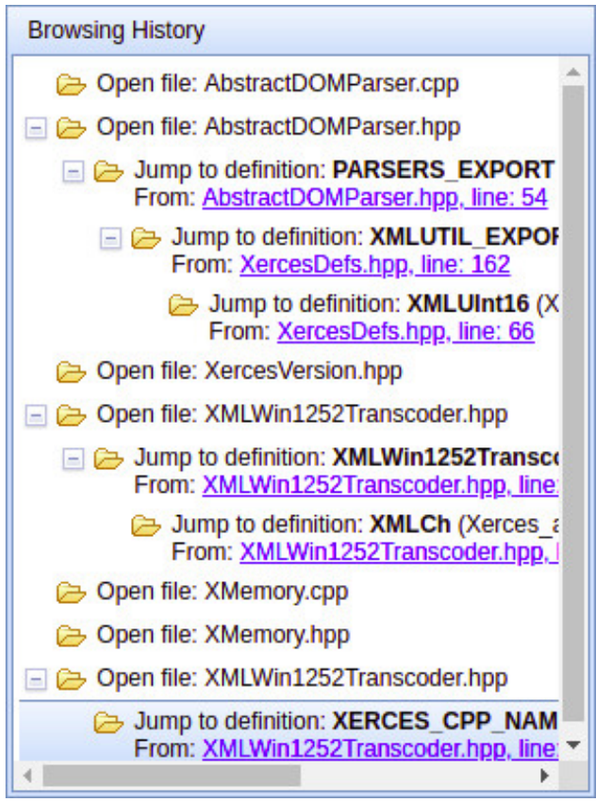
V režime fulltextového vyhľadávania je hľadaná fráza skupina slov, ako napríklad „vracia astnode *“. Fráza dopytu sa zhoduje s textovým blokom, ak sú hľadané slová vedľa seba v zdrojovom kóde v danom poradí. Zástupné znaky, napríklad * alebo? môžu byť použité pri porovnávaní ľubovoľného viacnásobného alebo jedného znaku. Logické operátory ako AND, OR, NOT sa dajú použiť na spojenie viacerých fráz dotazu súčasne.

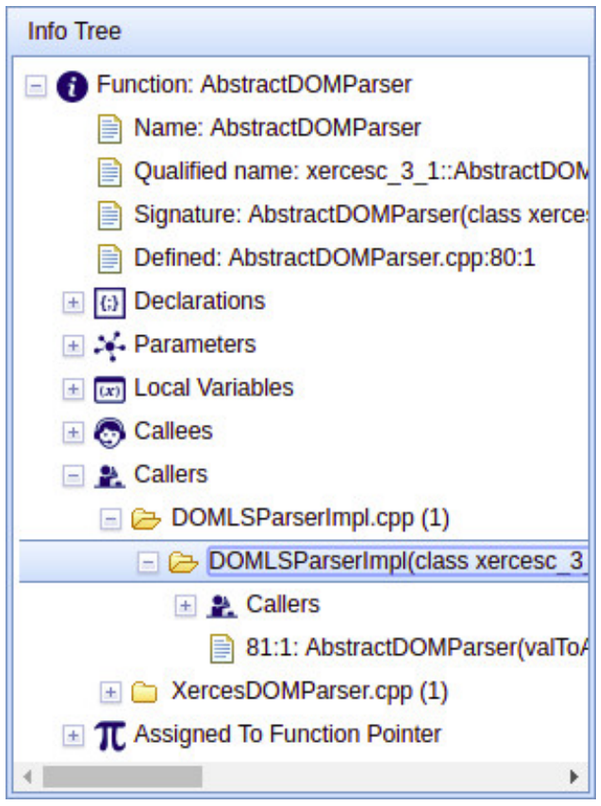
Na vyššej úrovni je možné nájsť symboly v zdrojových kódoch podľa definície definície. Tu používame CTag na indexovanie kódovej základne, aby sme mohli nájsť premenné, funkcie, triedy, makrá atď. Je dôležité vedieť, že toto vyhľadávanie jazykových entít nemá nič spoločné s hlbokým analyzovaním jazyka.

Pri ladení programu je niekedy jedinou informáciou, ktorá sa môže začať, výstupná správa v protokole konzoly vydanom naším softvérom. Toto je jediná stopa, ktorá môže začať, napr. "DEBUG INFO: TSTHan: sys_offset = -0,019821, drift_comp = -90,4996, sys_poll = 5". Upozorňujeme, že takáto správa môže obsahovať časové značky alebo iné dynamicky generované fragmenty, takže nie je možné nájsť túto správu ako priamy reťazec. Avšak v CodeCompass je možné fuzzy vyhľadávanie vykonať protokolovaním.

Informácie o jazykových symboloch

Po nájdení prvku je ďalším krokom zhromažďovanie informácií o ňom. Po výbere pomenovanej entity si môže používateľ vybrať z rozbaľovacej ponuky „Informačný strom“. Tento strom obsahuje všetky informácie poskytované jazykovým analyzátorom. V prípade C / C ++ používame kompilátor LLVM / Clang, aby sme získali informácie o symboloch. U funkcií môžeme skontrolovať ich parametre, lokálne premenné, volajúcich a volaných.





Zaujímavou črtou stromu je to, že volajúci sú uvedení striedavo, t. J. Deti uzla sú volajúci funkcie. Ich detské uzly volajú po týchto funkciách, a to pokračuje rekurzívne, teoreticky späť k hlavnej funkcii. Volania funkcií však nie sú vždy priame, ale môžu sa uskutočňovať prostredníctvom funkčných ukazovateľov. Aj keď sa jedná o dynamické správanie, CodeCompass privolá všetky výskyty, v ktorých bola funkcia priradená funkčnému ukazovateľovu a vyvolávanie sa uskutoční prostredníctvom tohto ukazovateľa.

V prípade tried sú zozbierané informácie aliasy (podľa typu môže mať trieda synonymum), dedičské vzťahy (zoskupené podľa viditeľnosti), priatelia, metódy / polia (priame alebo zdedené) a zvyklosti (ako lokálna / globálna premenná, parameter funkcie / návratový typ alebo pole inej triedy). Pre premenné je užitočné poznať miesta v kóde, kde boli napísané a prečítané. Pre typy výčtu sú konštanty výčtu uvedené s ich celočíselnými hodnotami.

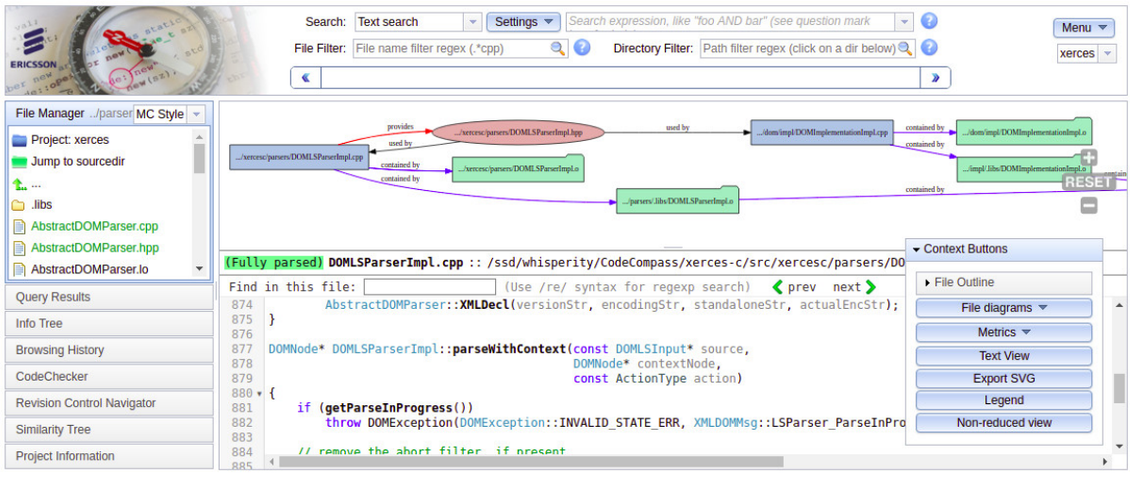
Diagramy

Vizualizácie sú jednou z najužitočnejších reprezentácií pre prehľad systému. CodeCompass predstavuje niekoľko diagramov založených na symboloch a súboroch. Tieto diagramy sú založené na grafoch, to znamená, že predstavujú entity a ich spojenia. Toto sú tiež interaktívne diagramy: vznášaním myši nad uzly reprezentovaná entita sa zobrazí v textovom zobrazení.

Kliknutím na ne sa vybraná entita stane stredovým uzlom ukazujúcim jej vzťahy podľa typu diagramu.

Diagram volania funkcií zobrazuje všetkých volajúcich a volaní funkcie v grafe. Schéma dedičnosti triedy UML zobrazuje celý reťaz dedičnosti až do koreňovej základnej triedy a rekurzívne pre všetky odvodené triedy. Implementovali sme tiež diagram analýzy ukazovateľov, ktorý ukazuje pridelené objekty a ukazovatele, ktoré na ne prípadne odkazujú. Toto je, samozrejme, dynamická informácia, ktorá sa dá získať len čiastočne v statickej analýze.

Schéma rozhrania požadovaná pre zdrojový súbor C / C + + ukazuje, ktoré hlavičky sú „použité iba“ alebo „implementované“ daným súborom. Použitie znamená, že zdrojový súbor používa iný súbor, ak sa v ňom nachádza použitie symbolu, ktoré je deklarované v inom súbore. Implementačný vzťah znamená, že symbol je deklarovaný v súbore (čím vytvára rozhranie) a je definovaný v inom. Tieto vzťahy sú tiež použiteľné pre adresáre zvažujúce obsiahnuté súbory. V prípade kompilovaného jazyka existujú aj výstupné súbory ako objekty a spustiteľné súbory. Na základe informácií o prepojení môžeme uviesť, ktoré zdroje vytvárajú binárny súbor.



CodeBites poskytuje odlišnú vizualizáciu kontrolovaného zdrojového kódu. V tomto pohľade sú uzly v grafe definície konkrétnych pomenovaných symbolov, ako sú triedy, funkcie atď. Ide o to, že programátor by chcel objaviť túto entitu pochopením jej správania, ale bez straty zamerania. Časti textového kódu v uzle sa dajú kliknúť, čím sa spustí pridanie definície vybraného prvku.

Vizualizácie riadenia verzií

Vizualizácia informácií o verzii je dôležitou pomôckou na pochopenie vývoja softvéru. Git vina pohľad ukazuje line-by-line zmeny (zaväzuje sa) k danému súboru. Zmeny, ku ktorým došlo nedávno, sú farebne svetlejšie zelené, zatiaľ čo staršie zmeny sú tmavšie červené. Toto zobrazenie je vynikajúce na posúdenie, prečo boli do zdrojového súboru pridané niektoré riadky. CodeCompass môže tiež zobraziť potvrdenia Git vo filtrovateľnom zozname zoradenom podľa času potvrdenia. Toto vyhľadávacie zariadenie sa môže použiť na zoznam zmien vykonaných osobou alebo na filtrovanie potvrdení podľa relevantných slov v správe o potvrdení.

Metriky

CodeCompass môže ukázať McCabe Cyclomatic Complexity [1], riadky kódu a počet chýb nájdených metrikami Clang Static Analyzer pre jednotlivé súbory a zhrnuté cez hierarchie adresárov. Tieto metriky je možné vizualizovať na stromovej mape, kde sú adresáre označené rámčekmi. Veľkosť škatule a jej farebný odtieň sú úmerné zvolenej metrike.

História prehľadávania

De Alwis a Murphy študovali, prečo programátori pri používaní integrovaného vývojového prostredia Eclipse Java (IDE) [8] zažívajú dezorientáciu. Používajú techniku vizuálnej hybnosti [9] na identifikáciu troch faktorov, ktoré môžu viesť k dezorientácii: i) absencia prepojenia navigačného kontextu počas prieskumu programu, ii) rušenie medzi displejmi na zobrazenie potrebných častí kódu a iii) sledovanie niekedy nesúvisiacich vzťahov. čiastkové úlohy. Prvý faktor znamená, že programátor počas vyšetrovania problému navštívi niekoľko súborov nasledujúcim spôsobom v reťazci volaní alebo skúma použitie premennej. Na konci dlhej prieskumnej relácie je ťažké si spomenúť, prečo sa vyšetrovanie skončilo v konkrétnom súbore. Druhým dôvodom dezorientácie je častá zmena rôznych názorov v Eclipse. Tretím prispievateľom k problému je, že vývojár pri riešení úlohy zmeny programu vyhodnocuje niekoľko hypotéz, ktoré sú samostatnými čiastkovými porozumeniami. Programátori majú tendenciu pozastaviť podúlohy (pred dokončením) a prejsť na ďalšie.

Napríklad programátor skúma, ako sa používa návratová hodnota funkcie, ale potom sa zmení na podúlohy, ktoré pochopia implementáciu samotnej funkcie. Zistilo sa, že pre vývojárov je ťažké pripomenúť si pozastavenú podúlohy [10]. CodeCompass implementuje zobrazenie histórie prehľadávania, ktoré zaznamenáva (v stromovej forme) cestu navigácie v zdrojovom kóde. Nová podúloha predstavuje novú vetvu stromu, zatiaľ čo uzly sú navigačné skoky v kóde označenom spojovacím kontextom (napríklad „skok na definíciu init“). Problém i) a ii) je teda riešený označenými uzlami v histórii prehľadávania, zatiaľ čo problém iii) je riešený vetvami priradenými k čiastkovým úlohám.

CodeChecker - C/C++ Bug Reporting

Statický analyzátor Clang implementuje pokročilý symbolický vykonávací motor na hlásenie chýb programovania. CodeCompass môže vizualizovať chyby identifikované ako Clang Static Analyzer a Clang Tidy pripojením k serveru CodeChecker [11]. CodeCompass zobrazuje polohu chyby a symbolickú cestu vykonávania, ktorá vedie k chybe.

Katalóg namespace a typov

CodeCompass spracováva dokumentáciu kxygénu a ukladá ich pre definície funkcií, typov a premenných. Poskytuje tiež zobrazenie katalógov typov, v ktorom sú uvedené typy deklarované v pracovnom priestore usporiadané hierarchickým stromovým zobrazením názvových priestorov.

ZHRNUTIE

Predstavili sme CodeCompass, statický analytický nástroj na porozumenie rozsiahleho softvéru. Bol navrhnutý tak, aby sa vyhlo rôznym nedostatkom existujúcich nástrojov na porozumenie, ktoré sú buď ľahké, ľahko použiteľné, ale bez hlbokých znalostí skutočného prekladača; alebo s veľkou váhou, ktoré nie je možné škálovať, nainštalované na klientskom počítači. S webovou, zásuvnou a rozšíriteľnou architektúrou môže byť rámec otvorenou platformou na ďalšie porozumenie kódu, statickú analýzu a úsilie v oblasti softvérových metrík. Pôvodná štatistika spätnej väzby od používateľov a použitia naznačuje, že tento nástroj je užitočný pre vývojárov v činnostiach zameraných na porozumenie a používa sa okrem tradičných IDE a iných nástrojov krížového odkazu.

Literatúra

[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.