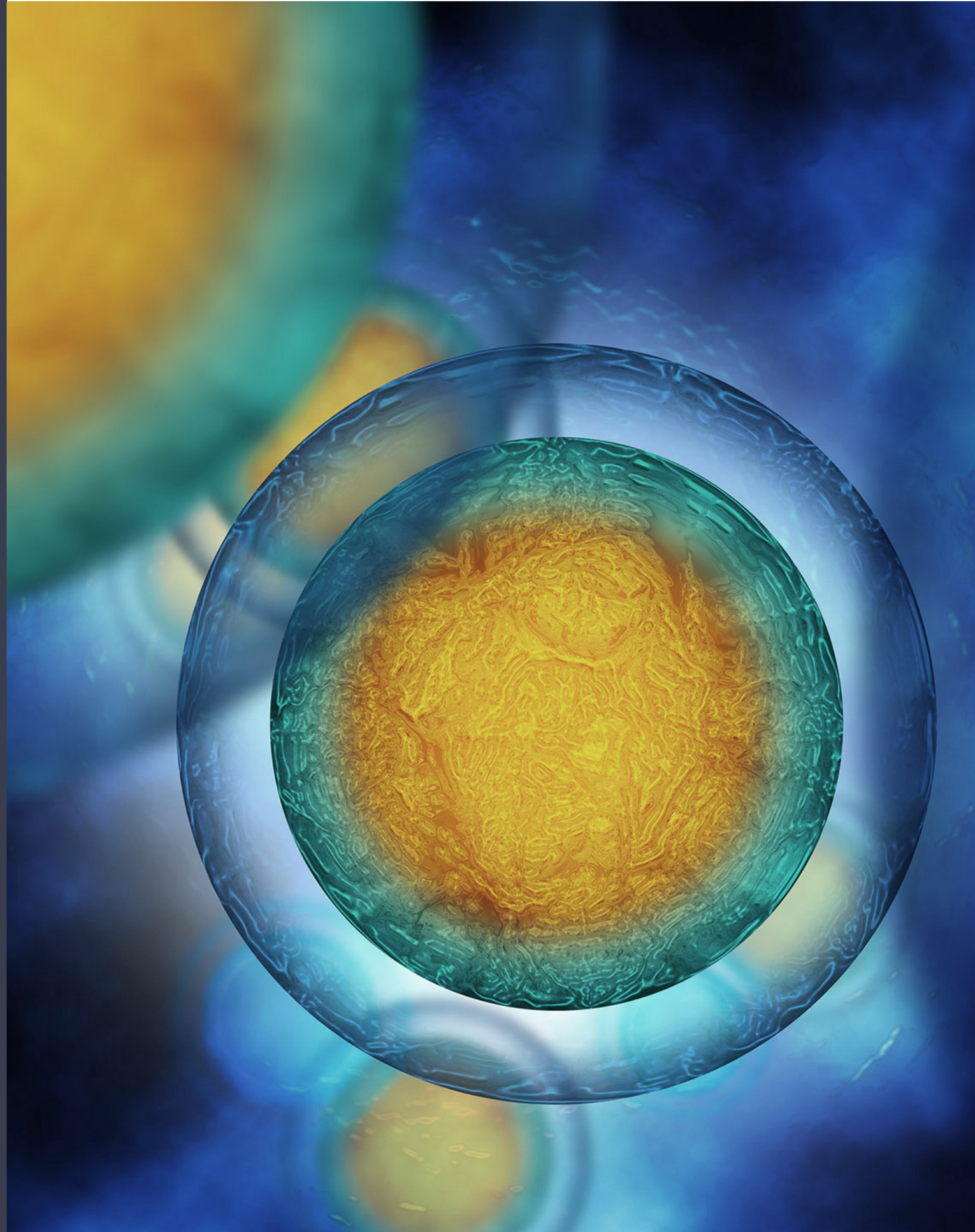
FE3CWS

SUMARUL EXECUTIV AL SCOLII AMȘTER-DAM

Cod:
Output intelectual nr. 2 al proiectului Erasmus+ 2017-1-SK01-KA203-035402



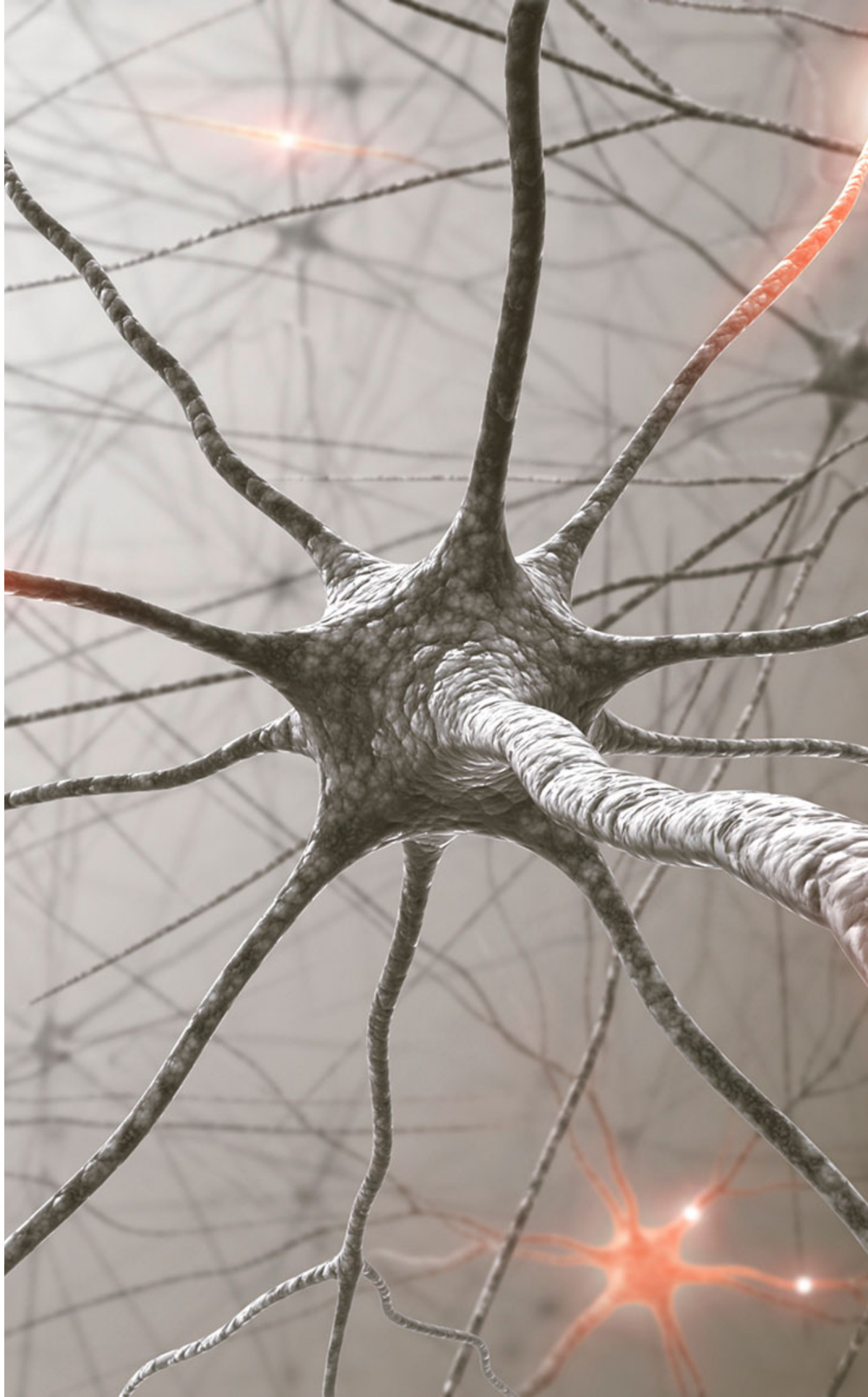
CUPRINS

abreviat:

- 6 teme, legate de dezvoltarea programelor complexe: compoziția sistemelor, inteligibilitatea codului, corectitudinea codurilor
- **Materialul didactic este disponibil în 7 limbi: bulgară, croată, engleză, maghiară, portugheză, română, slovacă**

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

TABLE OF CONTENTS



CALCUL ÎN "CLOUD" CENTRAT PE UTILIZATOR

Tehnologia modernă "Cloud computing" a devenit o tehnologie cheie și, prin urmare, face parte din multe programe de informatică. Cercetarea centrată pe utilizator se concentrează pe strategii de estimare a timpilor de rulare și a costurilor pentru implementarea aplicațiilor din lumea reală pe cloud.În domeniul "cloud computing", un aspect important este asistarea utilizatorilor în deciziile lor. Aceste decizii se referă la următoarele întrebări:

– Cum se comportă aplicația asupra resurselor virtualizate?

– Câte resurse virtuale de la ce tip de la ce furnizor de cloud trebuie achiziționat pentru implementarea aplicației?

– De cât timp va rula? Cât va costa?

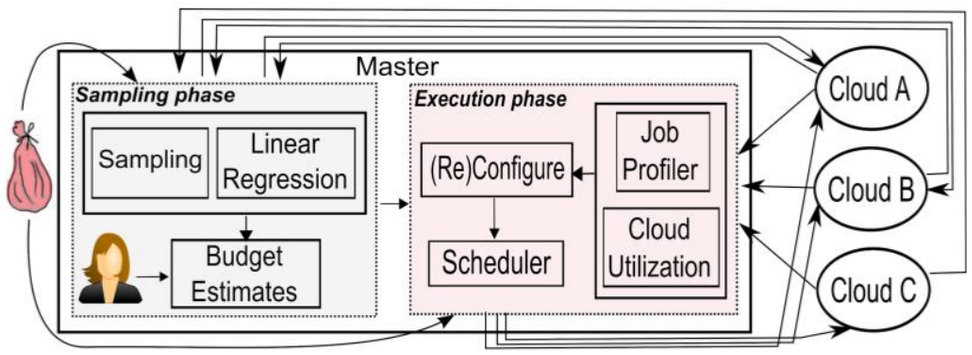
Aceste întrebări sunt modelate ca o problemă de planificare, funcționând cu presupunerea că nu există cunoștințe a priori despre aplicație. Un set tipic de cerințe simple este acela că aplicația este implementată cu succes și costurile sunt reduse la minimum.

ARHITECTURA COMPONENTEI DE SCHEDULER A UNEI APLICAȚII "CLOUD"

Planificatorul BaTS[4] a fost dezvoltat pentru a ajuta utilizatorii în a rula -- seta -- aplicațiile proprii în cloud. Este nevoie de o abordare de planificare automată pentru a realiza acest lucru și de a verifica în mod regulat progresul setării aplicației.

Figure 1 prezintă arhitectura BaTS. În timpul fazei de eșantionare, BaTS colectează statistici privind timpul de execuție al unora dintre sarcinile aplicației, folosind eșantionarea -- sampling-ul -- cu înlocuire. Aici este nevoie doar de un eșantion mic (30-50 de sarcini) pentru a calcula media și abaterea standard a timpului de rulare a sarcinilor pe diverse oferte "cloud". Regresia liniară este utilizată pentru a optimiza calculul bugetului și estimărilor de creștere a mărimii.

În timpul fazei de execuție, la intervale regulate de timp, configurația curentă este reevaluată, pentru a verifica dacă programul selectat este încă posibil.



Dacă se așteaptă o încălcare a bugetului, se obțin mașini mai profitabile (raport de preț/performanță mai bun). Dacă se așteaptă o încălcare a defecțiunii, sunt achiziționate mai multe mașini mai rapide.

FOLOSIREA TEHNICII BATS PE RESURSE AWS

În această secțiune prezentăm metoda supervizării proprietarilor de aplicații care doresc să selecteze cele mai bune opțiuni privind resursele virtualizate atunci când desfășoară aplicația lor pe resurse Amazon EC2 (AWS)[7].

Etapa de eșantionare optimizată

Ideea principală este de a utiliza timpul mediu de execuție pentru fiecare tip de resursă virtualizat pentru a calcula bugetul și mărimea estimării.

Cu toate acestea, obținerea acestor statistici poate suporta costuri semnificative, având în vedere numeroasele tipuri de oferte AWS EC2 (în prezent, 123[8]). Figura 2 arată sarcini selectate aleatoriu dintr-o aplicație în care timpul de rulare al sarcinilor urmează o anumită distribuție. Runtimes-urile acestor sarcini sunt utilizate pentru a calcula statisticile pentru o ofertă "cloud". După colectarea statisticilor pentru fiecare ofertă de "cloud" disponibilă, este posibil să calculăm bugetul și estimările de creștere. Presupunând că am dori să evaluăm fiecare ofertă curentă AWS EC2, asta ar însemna executarea a 30 de sarcini pe fiecare din cele 123 de tipuri de mașini. Dacă am executa pur și simplu diferite seturi de sarcini selectate la întâmplare (în total 3690), aceasta ar duce la o fază lungă (și costisitoare) de eșantionare, ceea ce face posibil ca orice decizie a utilizatorului să nu fie relevantă din două motive: a) rămân prea puține sarcini pentru a fi executat și b) bugetul utilizatorului ar putea fi deja depășit. Dacă am executa același set de selecții aleatorii pe fiecare tip de mașină, tot ar duce la o fază lungă (și costisitoare) de eșantionare.

Optimizăm această fază folosind regresia liniară pentru a reduce numărul total de sarcini care trebuie executate înainte de pregătirea statisticilor. Executăm același set de 7 sarcini selectate aleatoriu pe fiecare tip de mașină și colectăm rulări [9].

În continuare, executăm 23 de sarcini selectate aleatoriu pe utilajele care devin disponibile pentru prima dată. Figura 3 ilustrează abordarea noastră. Folosind rulajele celor 7 sarcini replicate, stabilim o relație liniară între timpii de execuție pentru sarcinile aplicației pe toate tipurile de mașini. Folosim aceste relații liniare pentru a cartona cele 23 de rulări la toate celelalte tipuri de mașini. După terminarea mapării, avem un set de 30 de rulări pentru fiecare tip de mașină, în timp ce executăm doar 884 de sarcini în loc de 3690.

Magnitudini diferite în monetizare

Odată obținute estimările de execuție, estimările de buget și de creștere a calculului sunt calculate folosind un algoritm modificat "Bounded Knapsack" [11].

Cu toate acestea, atunci când se iau în considerare resursele AWS EC2 cu un model de preț diferit, cum ar fi instanțele la fața locului, această abordare nu se extinde din cauza diferitelor ordine de mărime.

Pentru a rezolva această problemă, am renunțat la determinismul abordării din "Bounded Knapsack" pentru scalabilitatea unei abordări cu ajutorul algoritmului genetic.

Folosind un algoritm genetic, am putea aproxima frontul Pareto (set optim) de planificări realizabile pentru o anumită aplicație și un set de tipuri de mașini. Figura 4 prezintă frontul Pareto real și două estimări pentru o aplicație care are o distribuție multimodală a timpilor de rulare. Soluția noastră generează fronturi Pareto precise în 1 secundă.

Constatarea cheie aici a fost că pentru a asigura o acoperire bună a frontului real Pareto, funcția de fitness ar trebui să răsplătească și cea mai rapidă sau cea mai ieftină performanță.

Faza de sfârșit al calculului

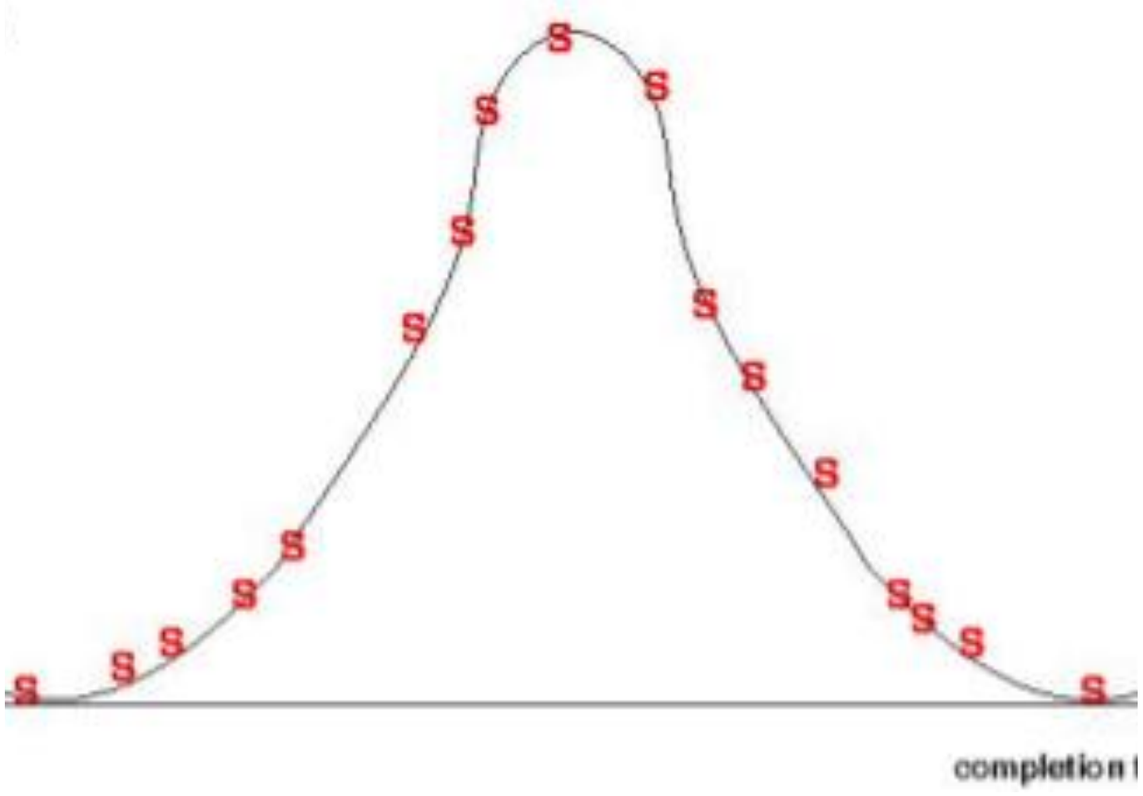
În faza finală a calculului, trebuie să se abordeze presupunerea că timpul de calcul este „fluid"[10]. În acest moment, prin definiție, aplicația conține prea puține sarcini pe care presupunerea să le păstreze. Am analizat mai multe abordări pentru a rezolva această problemă, punându-ne accent pe o estimare mai bună a nevoilor finale de calcul ale aplicației, astfel încât alocarea sarcinilor finale către mașini să fie optimă. Abordările noastre variau de la cunoașterea perfectă a perioadelor de rulare rămase la cunoștințe zero (rulări aleatorii). Impactul a fost nesemnificativ. Prin urmare, problema trebuia abordată într-o altă etapă, și anume la faza de estimare a bugetului și a evaluării.

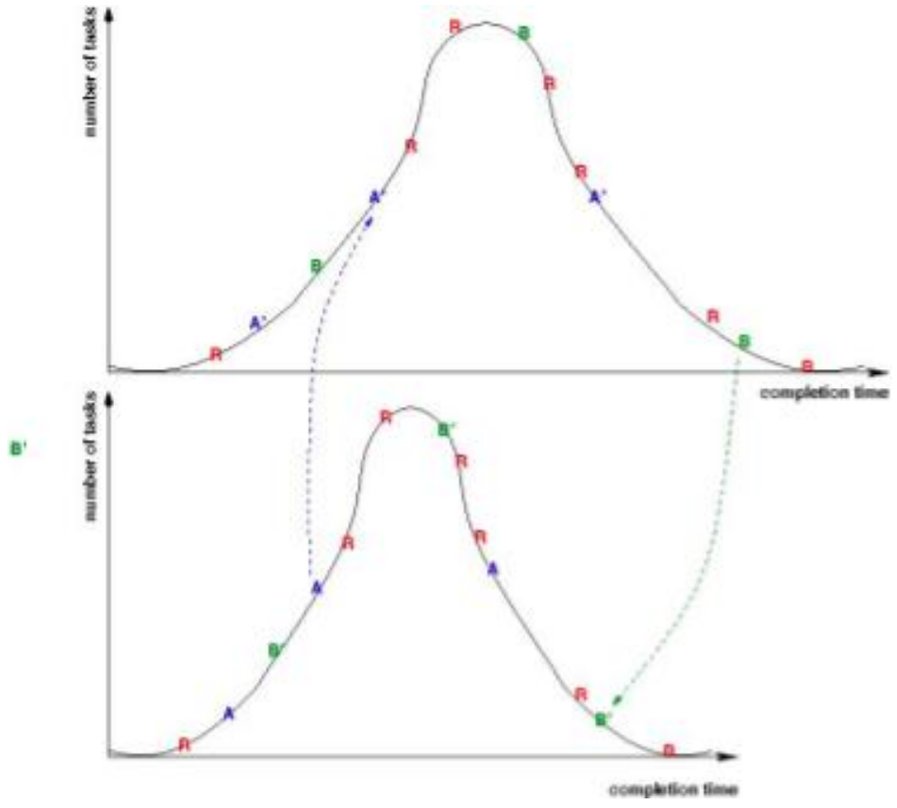
În acest scop, BaTS ține evidența fracțiilor estimate de unități de timp responsabile finale neutilizate. BaTS furnizează o pernă pentru a aborda valorile care rulează în afara unității de timp responsabile finale și adaugă resurse virtualizate și / sau timp la program. Cu toate acestea, valorile externe pot duce la încălcări.

Folosirea metodologiei BaTS în educație

În programa de masterat în informatică a Universității din Amsterdam, specializarea „Servicii web și sisteme bazate pe cloud" rulează de câțiva ani.

Un obiectiv important al acestei specializări este familiarizarea studenților cu provocările sistemelor bazate pe cloud.





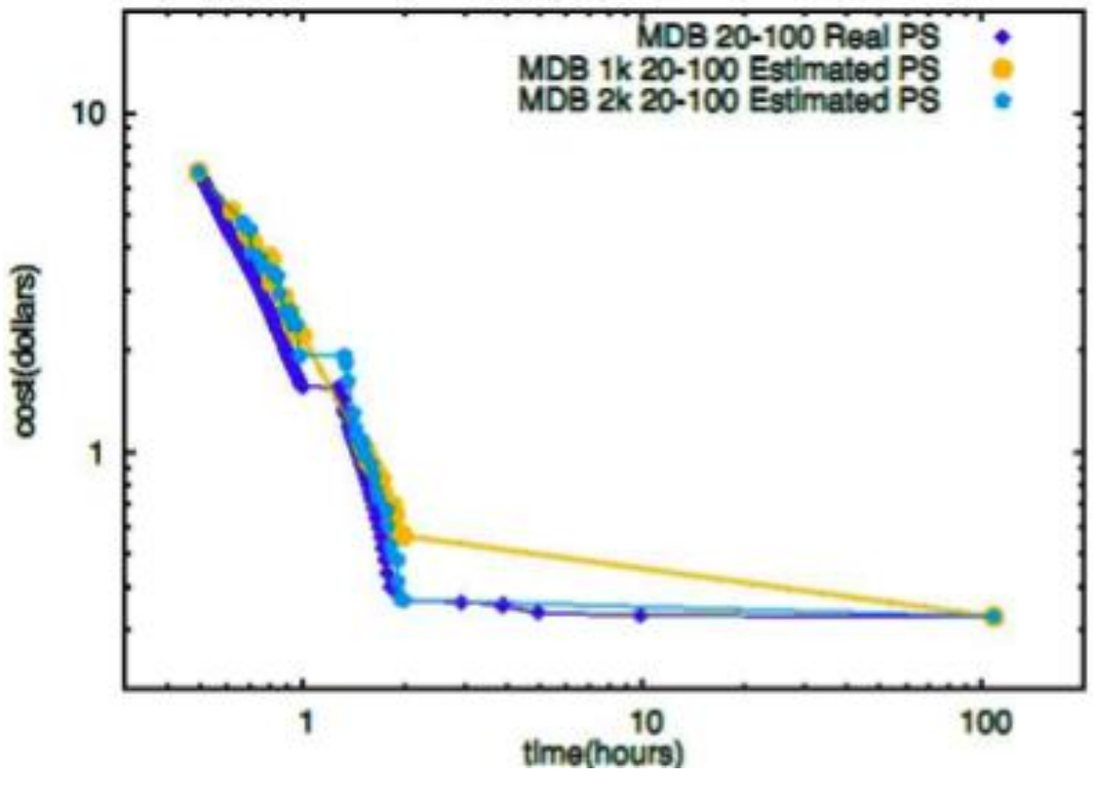
Lucrarea practică pentru acest curs presupune dezvoltarea unui planificator rudimentar pentru resurse virtualizate și analizarea comportamentului său într-un cadru intern față de un cloud comercial.

Sistemul intern din universitatea noastră constă dintr-o implementare OpenNebula pe DAS, clusterul de calcul național olandez. OpenNebula este o soluție open-source pentru implementări Infrastructura ca serviciu: resursele fizice sunt gestionate și oferite ca resurse virtuale. Aici, studenții au o limită superioară a numărului de mașini virtuale care pot fi lansate simultan.

Setarea "cloud" comercială constă din ofertele de cloud Amazon EC2[7]. Aici, studenții au un buget pe care îl pot utiliza pentru orice achiziție de resurse legate de laborator. Evaluarea activității lor de laborator are în vedere orice încălcare a bugetului.

Volumul de muncă pe care programatorul trebuie să-l gestioneze este o aplicație extrem de paralelă: o colecție de sarcini de procesare a șirurilor, legate de analiza sentimentelor mesajelor Twitter.

În general, rezultatul lucrării de laborator a arătat că studenții au fost capabili să înțeleagă diferența dintre cel mai bun efort și achiziția de resurse virtuale comerciale.



CONCLUZII & DIRECȚII DE CERCETARE

Abordările stocastice pentru programarea în cloud centrate pe utilizatori sunt promițătoare.

Încorporarea cercetării în educație de îndată ce ajunge la o stare stabilă este foarte importantă.

Ca activitate viitoare, am dori să susținem funcțiile Haskell AWS Lambda desfășurate prin implementarea API Haskell AWS.

Referințe

[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.

FOCUSUL EDUCAȚIONAL PE MĂSURAREA EFICIENȚEI ENERGETICE ÎN PROCESUL DE TESTARE

Misiunea profesorilor de inginerie software este de a pregăti viitorii ingineri software care pot stăpâni fiecare problemă pe parcursul întregului ciclu de viață al software-ului. Pe lângă abilitățile legate de înțelegerea nevoilor părților interesate și de compunerea software-ului de lucru corespunzător, capacitatea de a verifica corectitudinea rezultatelor joacă un rol semnificativ. În această lucrare, ne concentrăm pe ultimul set de abilități. Ne concentrăm pe testare, unde nivelul de automatizare este mai puțin important. Subliniem natura măsurării testării, mai precis, ne concentrăm pe măsurarea consumului de energie software care poate fi furnizat în timpul testării la diferite niveluri. Toate aceste aspecte sunt prezentate din punctul de vedere al unui educator în domeniul ingineriei software, cu scopul de a prezenta modul de configurare a unei sesiuni de laborator de inginerie software care se concentrează pe măsurarea eficienței energetice în timpul testării software și, cu comentariile autorilor la această propunere.

INTRODUCERE ȘI DEFINIȚII

Una dintre provocările pentru producătorii de baterii este cât timp bateria poate funcționa fără a fi reîncărcată. Desigur, există multe alte provocări, cum ar fi dimensiunea care afectează foarte mult forma dispozitivului și celălalt factor care este o caracteristică importantă a bateria este greutatea identificatorului. Bateria este considerată a fi ceva mai ușoară în comparație cu dispozitivul care are nevoie de o baterie pentru a funcționa. Provocarea de aici este modul în care dimensiunea acestuia este mai mică și mai ușoară și cu siguranță o eficiență ridicată în ceea ce privește timpul de funcționare al dispozitivului mobil fără a fi încărcat. O componentă importantă a acestei provocări este fratele mic a părții de hardware: o provocare software, și anume că software-ul în sine ar trebui să sprijine economisirea de energie. A face acest lucru fără a limita experiența utilizatorului este considerat astăzi drept un obiectiv important al fiecărei dezvoltări de software care vizează orice fel de dispozitive portabile.

Amintirea faptului că consumul de energie al oricărui dispozitiv mobil este influențat începând cu rularea aplicațiilor prin nivelul de acces al serviciilor de bază până la dispoziția reală a utilizatorului, dezvoltarea software-ului pentru astfel de dispozitive este deja o provocare [1]. S-ar putea spune, provocarea de dezvoltare software este întotdeauna aceeași, dar trebuie să subliniem ``mobilitatea'' ca proprietate de sistem cheie aici. Starea de alimentare a bateriei determină, de asemenea, performanța sistemului datorită configurației nivelului sistemului de operare - cunoscut sub numele de ``preferințe de economisire a energiei''.

Este și mai greu dacă unul (în cazul nostru profesorul) trebuie să pregătească elevii pentru astfel de provocări. Toate ``cele mai bune practici'' și ``sfaturi de economisire a energiei'' deja cunoscute trebuie să fie prezentate într-un context, ușor de înțeles pentru elevi. Acest lucru se poate realiza prin poziționarea conceptelor într-un mediu cunoscut, cum ar fi testarea software și testarea automatizării [2], în cazul nostru. Este ceea ce ne propunem cu această lucrare, acesta este conținutul secțiunilor viitoare, începând cu propunerea urmată de o evaluare și încheind cu sfaturi suplimentare privind îmbunătățirea.

SOLUȚIA PROPUSĂ

După cum s-a menționat mai sus, trebuie să găsim cel mai adecvat mediu pentru a introduce practicile de măsurare a consumului de energie [3] și de evaluare a eficienței energetice.

Acestea ar putea fi atât dezvoltarea inițială a software-ului, cât și evoluția software-ului, deoarece ambele faze de dezvoltare ale ciclului de viață al software-ului oferă oportunități pentru a măsura produsul dezvoltat și evoluat[4].

Cu alegerea procesului de dezvoltare a software-ului, avantajul este că toate activitățile ar putea pune accentul pe problemele legate de economisirea de energie, în timp ce alegerea alternativei de evoluție a software-ului oferă posibilitatea de a evalua îmbunătățirea implementării produsului.

Pe de altă parte, evoluția software necesită existența unui software la începutul procesului de dezvoltare, în timp ce dezvoltarea inițială a software-ului este procesul în care se creează produsul începând cu prima cerință a software-ului.

Pentru introducerea celor două abordări posibile în mediul didactic, cea mai bună opțiune ar fi utilizarea ambelor, dedicând un semestru pentru dezvoltarea inițială de software și un alt semestru pentru evoluția aceluiași software.

De obicei, profesorul nu are două semestre la rând pentru a prezenta conținutul cursului în modul prezentat în paragraful anterior. Acesta este motivul pentru care trebuie să decidem ce tip de dezvoltare să folosim pentru a introduce practicile selectate. Din punct de vedere al arhitecturii procesului de dezvoltare și prin faptul că dezvoltarea inițială ar putea fi și evolutivă, ne decidem pentru evoluția software-ului. Acesta include multe activități de dezvoltare inițială (cu excepția colectării și analizei timpurii a cerințelor) și subliniază importanța testării și evaluării.

Această alegere permite profesorului să::

– Permiteți studenților să privească înapoi în istoria dezvoltării lor (proiecte anterioare) pentru a fi critici pentru ei înșiși.

– Permiteți-le să își evalueze rezultatele folosind metricele de cod și măsurarea consumului de energie (sau estimarea).

– Permiteți-le să integreze activitățile de mai sus în procesele standard de verificare și validare a evoluției software.

Limbajul de programare al dezvoltării nu este important, de aceea elevul poate selecta oricare dintre proiectele sale anterioare pentru evoluție - sau toate, dacă concurează în numărul de proiecte evoluate sau limbaje de programare utilizate. Dar, limbajul de programare determină sau limitează de obicei mediul de dezvoltare și instrumentele utilizate. Selectarea acestor instrumente și a pluginurilor lor oferă, de asemenea, un suport bun pentru evaluarea valorilor codului.

Măsurarea consumului de energie și evaluarea eficienței energetice necesită, de obicei, un instrument diferit, deoarece există doar puține medii de dezvoltare care integrează măsurarea sau estimarea consumului de energie până acum.

În ceea ce privește testarea ca bază de măsurare, trebuie să reținem că analiza codului static este utilizată pentru a face parte din testarea software-ului. În afară de aceasta, anumite părți ale bazei codului aplicației sunt executate în timpul testării cutiei albe (în principal testarea unității), care printr-o mică extensie a măsurării consumului de energie poate prezenta consumul de energie al cazului de testare - o privire indirectă asupra consumului de energie a codul testat. În timpul testării cutiei negre, cererea completă este testată folosind scenarii de testare. Aceste scenarii de testare sunt în principal comparabile cu cazurile de utilizare a software-ului de o zi întreagă, altele reprezintă scenariile de graniță - inclusiv cele în care utilizatorul are o dispoziție proastă. Măsurarea consumului de energie al execuției acestor teste din cutia neagră oferă apoi o privire aproximativă (dar directă) asupra consumului de energie al produsului.

Iată principalul beneficiu al evoluției (în comparație cu dezvoltarea inițială de software). Profesorul poate pregăti versiunea de pornire pentru evoluție, inclusiv codul care poate fi îmbunătățit, lista de bug-uri cunoscute și baza de testare! Existența testului pentru testarea de testare și regresie este foarte importantă aici, întrucât productivitatea evoluției poate fi crescută prin această caracteristică.

Presupunând că s-au făcut toate etapele de mai sus, integrarea măsurătorilor de eficiență energetică în procesul de testare privește din perspectiva activităților studentului, astfel:

1. Selectați produsul care ar putea fi proiectul dvs. trecut sau dintr-un depozit.

2. Evaluează-l folosind analiza codului static, testarea, măsurarea energiei, sondajul de utilizare, etc.

3. Îmbunătățește-l (diferite tipuri de evoluție, cum ar fi adăuga / schimba funcționalitatea, repara sau adapta)

4. Încearcă din nou pentru a fi sigur că ai eliminat un defect sau o defecțiune

5. Test de regresie (inclusiv reevaluare)

6. Concludeți rezultatele (faceți un verdict final pentru toate datele disponibile, inclusiv eficiența energetică).

DISCUȚII

Întrucât măsurarea consumului de energie este relativ nouă în comparație cu alte tehnici, cum ar fi analiza codului static, testarea la cutie alb-negru și depanarea, ar putea fi punctul de eșec. Dar dacă se combină cu aceste principii mai în vârstă construind un scor compus pentru fiecare student, proprietatea critică este mult mai mică.

Analizând posibilitățile de clasare, putem găsi diferite ``grade de libertate'', care se oferă pentru a fi utilizate separat sau ca parte a unei compoziții de grad:

1. numărul articolului de proiecte diferite,

2. numărul de limbaje de programare aplicate / utilizate,

3. codul articolului calitatea produsului final,

4. eficiența energetică a produsului final,

5. îmbunătățirea calității codului articolului în evoluția software-ului,

6. îmbunătățirea eficienței energetice în timpul evoluției software-ului.

Având în vedere toate ``gradele de libertate'' ale finalizării sarcinilor, multe competiții ar putea fi definite pentru studenții competitivi, în timp ce câștigătorii vor colecta ``mici victorii'' în număr de proiecte, scopul perfecționiștilor va fi optimizarea tuturor codurilor. măsurători și minimizarea consumului de energie.

Pentru studenții obișnuiți, îmbunătățirea eficienței energetice și înțelegerea codului ar putea fi obiectivul realizabil.

Concurența studenților poate fi și mai susținută prin faptul că nu îi permite să revină la proiectele lor anterioare, ci le permite să aleagă dintr-un depozit selectat. Ca și în cazul jocurilor pe calculator, toate nivelurile de joc sunt apoi la fel de accesibile tuturor. Pentru a sprijini echitatea, de asemenea, ar putea fi creat un forum public comun de studenți și profesori.

Activitățile noastre viitoare în acest domeniu se vor concentra pe configurarea unui mediu portabil de dezvoltare integrată, testare și estimare a consumului de energie. Acest mediu va fi utilizat în cadrul evoluției software sau a subiecților de dezvoltare inițială pentru a sprijini educația privind măsurarea eficienței energetice în timpul testării software. S-ar putea să limiteze creativitatea elevilor prin oferirea unui sandbox semi-închis, deoarece hardware-ul joacă un rol foarte important în arhitectura actuală a estimării consumului de energie. Unele cercetări își propun să încalce această limitare - așteptăm cu nerăbdare aceste rezultate pentru a le integra.

Referințe

[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.

CERCETĂRI VIZÂND O DISCIPLINĂ INGINEREASCĂ DE SOFTWARE ”GREEN”

Acest raport tehnic descrie cercetările dezvoltate în grupul ”Green Software” al Universităților Coimbra și Minho, care a fost prezentat la prima întâlnire de formare a cadrelor didactice din proiectul Erasmus+, intitulat ``Focusul în educație pe compozibilitate, înțelegere și corectitudinea al unui software Funcțional'' -- (eng) ``Focusing Education on Composability, Comprehensibility and Correctness of Working Software'' (3COWS).

Prezentăm atât un clasament verde pentru limbajele de programare cât și structurile de date și tehnici pentru localizarea consumului de energie anormală în sistemele software.

MOTIVAȚIE

Utilizarea actuală pe scară largă a dispozitivelor de calcul fără fir, dar puternice, cum ar fi, smartphone-uri, laptopuri etc., schimbă modul în care atât producătorii de calculatoare cât și inginerii de software își dezvoltă produsele. De fapt, timpul de execuție al calculatorului / software, care a fost obiectivul principal în secolul trecut, nu mai este singura preocupare.

Consumul de energie devine un blocaj în creștere atât pentru sisteme hardware, cât și pentru sisteme software. În consecință, cercetarea pe software-ul verde este un domeniu relevant și activ al cercetării.

Acest raport descrie pe scurt cercetările în curs de dezvoltare software verde în Laboratorul Software Verde (Green Software Laboratory -- GSL). GSL este format din diferite grupuri de cercetare portugheze, inclusiv două site-uri ale proiectului ``Focusul în educație pe compozibilitate, înțelegere și corectitudinea al unui software Funcțional''. GSL este o inițiativă de dezvoltare a tehnicilor și instrumentelor care vizează reducerea consumului de energie pe diverse sisteme de calcul (mobil, programe, baze de date etc.).

GSL se concentrează în special pe partea de software, unde aplică tehnici de analiză și transformare (cod sursă) pentru a detecta anomalii în consumul de energie și pentru a defini optimizări pentru a reduce acest consum. În secolul trecut, eficiența unui sistem software a fost axată în principal pe timpul de execuție și eficiența consumului de memorie.

În zilele noastre, dezvoltatorii de software își pun adesea întrebarea ``este un program mai rapid sau este mai ecologic?''. Există multe aspecte ale unui sistem software care îi influențează performanța energetică: limbajul de programare și modelul său de execuție (compilat la codul binar sau la o mașină virtuală, cod interpretat, leneș versus evaluare strictă, utilizarea evaluării parțiale în timp de rulare, etc.). Eficiența modelului de memorie și a bibliotecilor de limbi influențează, de asemenea, performanța.

Complexitatea algoritmului folosit pentru implementarea problemei dorite de computer, influențează și performanța: dacă algoritmul implementat trebuie să facă mai multă muncă decât ceea ce este strict necesar, atunci se vor folosi mai mult procesor și energie.

În acest document raportăm pe scurt rezultatele cercetărilor în cadrul grupurilor GSL, și anume în analiza eficienței energetice a limbajelor de programare, bibliotecile pentru structuri de date și codul sursă al software-ului.

CARACTERUL ”GREEN” AL LIMBAJELOR DE PROGRAMARE

O întrebare interesantă ce apare în discuții despre consum de energie în limbajele de programare este dacă un limbaj mai rapid este, de asemenea, un limbaj eficient din punct

de vedere energetic. Compararea limbajelor software este însă o sarcină extrem de complexă, întrucât performanța unui limbaj este influențată de calitatea compilatorului, al mașinii virtuale, al colectorului de gunoi -- garbage collection, etc.

În Laboratorul de Software Verde (GSL) am studiat, am evaluat și am comparat performanța în total a 27 dintre cele mai utilizate limbaje de programare.

Am folosit două "repozitoare" diferite de coduri surse a problemelor: Computer Language Benchmark Game (CLBG) și Rosetta Code [2, 3].

Ambele repozitoare definesc un set de probleme pe calculator și furnizează implementări într-un grup mare de limbaje de programare. În timp ce CLBG a fost adaptat pentru a analiza performanța timpului de execuție a limbajelor de programare, Rosetta Code a fost definit mai mult pentru scopuri de înțelegere a codului sursă.

Am compilat / executat astfel de programe folosind compilatoare de ultimă generație, mașini virtuale, interpretoare și biblioteci pentru fiecare limbaj de programare în parte. Apoi, am monitorizat timpul de execuție, consumul de vârf și memoria generală și consumul de energie CPU/DRAM/GPU. Am realizat un clasament energetic al celor 27 de limbaje de programare și am analizat, de asemenea, rezultatele în funcție de tipul de execuție al limbajelor (compilat, mașină virtuală și interpretată) și de paradigma de programare (imperativ, funcțional, orientat pe obiect, limbaj script).

Pentru fiecare dintre tipurile de execuție și fiecare paradigmă de programare, am alcătuit un clasament de al limbajelor în funcție de fiecare obiectiv individual (de exemplu, consumul de timp sau de energie). Primele noastre rezultate arată cele așteptate, cum ar fi limba C, atât cea mai rapidă cât și cea mai ecologică. Cu toate acestea, rezultatele noastre arată că limbajele "mai lente", tot au tendința de a fi mai eficiente din punct de vedere energetic [2, 3].

EFICIENȚĂ -- GREENNESS -- ÎN STRUCTURI DE DATE

Limbajul / paradigma de programare și compilatorul său puternic optimizat nu este singurul aspect care influențează consumul de energie al unui sistem software.

De fapt, un program poate deveni și mai eficient prin optimizarea ``doar'' a bibliotecilor sale [4, 5].

Majoritatea limbajelor de programare oferă biblioteci puternice pentru a manipula structurile de date.

În GSL am studiat performanța energetică a două structuri avansate de date utilizate pe scară largă în limbajele de programare Java și Haskell. În Java, am efectuat un studiu detaliat privind consumul de energie al bibliotecii Java Collections Framework (JCF).

Am considerat obișnuitele trei grupuri diferite de structuri de date, și anume Seturi, Liste și Maps (hărți asociative), iar pentru fiecare dintre aceste grupuri, am studiat consumul de energie al fiecăreia dintre diferite implementări și metode[4]. Această conștientizare energetică a JCF poate fi utilizată nu numai pentru a direcționa dezvoltatorii de programe software în Java înspre un sistem mai ecologic, ci și în optimizarea codului Java vechi.

Am dezvoltat un instrument de refactorizare a structurii de date Java, numit jStanley, care refactorizează codul sursă Java când avem disponibil o bibliotecă cu caracteristici mai ecologice [6]. De asemenea, am executat o evaluare inițială cu ajutorul a 7 proiecte publice Java, și pentru aceste proiecte am putut îmbunătăți consumul de energie între 2% și 17%.

În Haskell, am studiat consumul de energie al sistemului Edison, o bibliotecă completă și bine documentată a structurilor de date pur funcționale[7]. Edison oferă diferite structuri funcționale de date pentru implementarea a trei tipuri de abstractizări: secvențe (liste, cozi și stack-uri), colecții (seturi și heap-uri), și colecții asociative (hărți și structuri de relații finite -- finite relations). Am analizat 16 implementări ale unor astfel de structuri de date, în timp ce măsuram metrici detaliate legate de consumul de energie și de consumul de timp [5].

Am investigat în continuare impactul asupra consumului de energie folosind diferite optimizări de compilare. Am ajuns la concluzia că consumul de energie este direct proporțional cu timpul de execuție și că consumul de energie al DRAM-ului reprezentând între 15 și 31% din consumul total de energie. În cele din urmă, am ajuns la concluzia că optimizările pot avea atât un impact pozitiv, dar și unul negativ asupra consumului de energie.

EFICIENȚĂ -- GREENNESS -- IN CODURI SURSĂ

Nu numai limbile și bibliotecile de structură a datelor influențează consumul de energie, de asemenea, un rol cheie are algoritmi și practici de programare privind eficiența programelor. În GSL am adaptat tehnici de localizare a erorilor bine cunoscute pentru a localiza în mod static ``scurgeri de energie'' (ineficiență energetică, defecțiuni energetice) în codul sursă al aplicațiilor [8–11].

Am definit SPELL - Localizarea scurgerilor energetice bazate pe SPectrum- pentru a determina zonele roșii (ineficiente de energie) din software. Un prim studiu experimental arată că programatorii experți, cu acces la scurgerile de energie detectate de SPELL, au fost capabili să optimizeze mai bine consumul de energie al programelor (între 15% și 74%), decât experții fără informații sau informații furnizate. de către un ”profiler” de programe standard (runtime). De asemenea, am studiat comportamentul energetic al programelor C/C++ [12].

Utilizarea pe scară largă a dispozitivelor fără fir și apariția internetului lucrurilor modifică modul în care inginerii software își dezvoltă software-ul. Software-ul trebuie să funcționeze pe o varietate de dispozitive mobile, iar consumul de energie este o preocupare principală atunci când dezvoltă software. Liniile de produse software (SPL) au apărut ca o importantă disciplină de inginerie software care permite dezvoltarea de programe care împărtășesc un set comun de caracteristici - features. În GSL am definit tehnici de analiză statică care să motiveze consumul de energie în SPL bazate pe compilarea condiționată. Aceste tehnici permit dezvoltatorilor de software să identifice caracterul verde (sau nu) a produselor soft și/sau a caracteristicilor -- features într-un SPL[13].

Android este un ecosistem utilizat pe scară largă pentru dispozitivele mobile -- în general dispozitivele fără fir --, iar analiza și optimizarea consumului de energie a software-ului este un domeniu activ de cercetare. Echipa GSL a dezvoltat mai multe tehnici[14, 15] și instrumente pentru a analiza și optimiza consumul de energie prin analiza codului sursă al aplicațiilor Android[16, 17].

În zilele noastre, majoritatea datelor stocate pe dispozitivele noastre mobile (fișiere, fotografii, videoclipuri) sunt de asemenea stocate în "cloud" care sunt furnizate de ecosistemul sistemului de operare al dispozitivului.

Astfel de sisteme cloud sunt centre de date care administrează zilnic o mare cantitate de procese de interogare a datelor, monitorizate și controlate de sisteme de gestionare a bazelor de date, care sunt responsabile să stabilească planuri de procesare a interogărilor care să fie eficiente.

Sistemele de baze de date se bazează de obicei pe planuri care optimizează timpul de răspuns. Am conceput și dezvoltat o metodă alternativă pentru definirea planurilor de consum de energie pentru interogările bazelor de date[18, 19]. Primele noastre rezultate experimentale arată că utilizarea euristicii de optimizare permite câștiguri semnificative, atât în ceea ce privește consumul de energie, cât și în timpul petrecut cu executarea interogărilor.

CONCLUZII

Acest raport tehnic a descris cercetările dezvoltate la Green Software Laboratory (GSL), și anume un clasament verde al limbajelor de programare și al structurilor de date, tehnici de detectare a ineficienței energetice în codul sursă al unui sistem software și un plan de execuție de interogare pentru sistemele de baze de date.

Referințe

[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

PREDAREA PROGRAMĂRII ORIENTATE SPRE TASK-URI

La școala de iarnă despre compozibilitate, înțelegere și corectitudine -- Composability, Comprehensibility, Correctness (3CoWs) --, sunt două tutoriale despre programarea orientată spre task-uri și sisteme bazate pe această paradigmă: sistemul iTask oferă o interfață web pentru ca dezvoltatorii să-și definească sarcinile și relația dintre acestea. Sistemul mTask aplică aceleași concepte pentru a specifica sarcinile executate de microprocesoare. În această contribuție, argumentăm programul de la școala profesională din Amsterdam. Datorită timpului limitat și al publicului cu cunoștințe divergente, ne vom concentra asupra utilizării practice a paradigmei TOP (task-oriented programming), cu mențiunea că timpul alocat prezentării este extrem de minimal și cu încurajarea ca doritorii să consulte materiale adiționale.

INTRODUCERE

Programarea orientată spre sarcini, TOP, este un stil de programare centrat în jurul conceptului de sarcini, executat de oameni și mașini. Aceste sarcini sunt specificate de funcțiile obișnuite într-un limbaj de programare funcțional. În toate exemplele noastre, vom folosi Clean [6]. Definirea semanticii sarcinilor este destul de diferită față de funcțiile simple: o sarcină este evaluată în mod repetat până când produce o valoare stabilă sau rezultatul acestui sarcini -- task -- nu mai este necesar. Rezultatele unei sarcini intermediare pot fi observate de alte sarcini. Sarcinile pot fi compuse de combinatori de sarcini.

La școala “Composability, Comprehensibility, Correctness (3CoWs)” din Kosice, vor avea loc două sesiuni pe TOP. Acestea sunt intitulate De ce contează ''Programarea orientată spre sarcini'', respectiv Programare funcțională a dispozitivelor IoT.

Ambele sesiuni vor consta dintr-o prezentare respectiv o parte practică pentru participanți.

În această lucrare, motivăm deciziile cu privire la conținutul și organizarea acestor sesiuni, care au rezultat după discuțiile de la pregătirea profesorilor din Amsterdam.

AUDIENȚĂ

Școala de iarnă "Composability, Comprehensibility, Correctness" este destinat pentru studenți BSc, MSc, PhD, dar și profesorilor doritori să învețe paradigme noi. Discuțiile din cadrul sesiunii de formare a cadrelor didactice de la Amsterdam au relevat faptul că experiența în programarea funcțională este variată; atât cantitativ, cât și relativ la limbajul de programare. Limbajele de programare cunoscute variază de la limbi pure și lenește, cum ar fi Haskell și Clean până la Erlang, Scheme și Scala.

Această variabilitate implică faptul că nu avem la dispoziție un fundament de programare funcțională solidă, ci doar o parte a publicului este familiarizată cu concepte precum tipizarea puternică, funcțiile de nivel superior, clase de constructor tipuri, Monad-uri și funcții generice. Deși aceste subiecte sunt elementele de bază ale TOP, nu putem presupune că sunt cunoscute de toți participanții.

PROGRAMAREA ORIENTATÂ SPRE TASK-URI

Programarea orientată spre sarcini se bazează pe un număr mic de sarcini primitive specifice domeniului. Aceste sarcini primitive interacționează de obicei cu mediul, de exemplu, oamenii care execută o parte din sarcini sau hardware care interacționează cu lumea fizică. Combinatoarele de activități sunt utilizate pentru a compune sarcina din sarcini mai mici. Sarcinile pot comunica prin rezultatele lor, precum și prin Surse de date partajate -- Shared Data Sources (SDS). Un astfel de SDS conține date tipizate puternic, care pot fi accesate prin intermediul primitivelor precum get și set. Aceste primitive acționează asupra stării sarcinii pentru a asigura transparența referențială.

Pentru a reutiliza tipuri de date și calcule ale unui limbaj existent, un sistem TOP este adesea construit ca un limbaj specific DSL, încorporat într-un limbaj de programare (funcțională).

Sistemul iTask a fost prima implementare a TOP. Este un DSL încorporat în limbajul funcțional de programare Clean. Sistemul iTask facilitează interacțiunea cu agenți umani prin generarea tipului de pagini web. Aceste pagini sunt afișate într-unul dintre browserele existente. Pagina oferă informații despre sarcinile curente pentru un utilizator. Utilizatorul poate interacționa cu sistemul iTask completând formulare și apăsând butoane.

TEHNICI FOLOSITE

Sistemul iTask transmite "stări" într-un stil asemănător cu o monadă -- monad -- de stare. Operatorii corespunzători funcțiilor return, bind (>>=), și înșiruire (>>|) sunt asemănătoare versiunilor monadice cunoscute [4, 7]. Înțelegerea acestora necesită funcții de ordin superior și operatori infix definiți de utilizator.

Combinatoarele de sarcini -- task combinators -- sunt tot funcții de ordin superior, de obicei operatori tip infix definiți de programator, care manipulează rezultatele sarcinii și starea globală a sarcinii. Specific sistemului TOP este producerea rezultatelor intermediare de către task-uri; aceste rezultate pot fi observate cât timp observare este necesară sau rezultatul lor nu mai este utilizat.Acest lucru necesită funcții de ordin superior, caracterul lazy și mecanismul de "garbage collection".

Sistemul iTask generează un server web care este folosit de agenții umani pentru a-și găsi sarcina. Ca toate serverele web, aceasta necesită serializare și deserializare a stării pentru a stoca și a recupera starea curentă. Prin completarea formularelor web pentru tipuri de date algebrice arbitrare, utilizatorii indică progresul lor cu sarcinile. Toate aceste caracteristici sunt implementate folosind programare generică.

Pentru implementarea eficientă a sarcinilor care monitorizează valoarea unui SDS, există un sistem ascuns de publicare-abonare pentru fiecare SDS care activează sarcini folosind acest SDS atunci când valoarea acestuia este actualizată. În afară de aceste proprietăți, sistemul iTask folosește multe tehnici suplimentare. De exemplu, execuția părților de sarcină într-un interpret care rulează în browser pentru a asigura un răspuns rapid al sistemului pentru sarcini extrem de interactive.

PREDAREA LA ȘCOALA DE IARNĂ

Orice predare a programării necesită experiență de programare practică folosind tehnicile educate pentru a le stăpâni; acest lucru este valabil și pentru TOP.

În consecință, împărțim cele patru ore disponibile pentru De ce contează ''Programarea orientată spre sarcini'' în două părți aproape identice: în prima parte, vom prezenta conceptul de TOP folosind sistemul iTask.

Având în vedere nivelul studenților, trebuie să omitem aproape toate detaliile despre implementarea sistemului și trebuie să ne concentrăm spre utilizarea bibliotecii de funcții.

Această bibliotecă este de fapt un DSL încorporat pentru realizarea sistemului TOP.

În prezentare definim numai funcționalități de bază, fără să intrăm în detalii.

Pentru partea practică, vom studia un proiect existent dintr-un set de mici proiecte TOP independente.

Problemele practice vor consta în mici variații ale acestor proiecte pentru a experimenta lucrul prin programare orientată spre sarcini.

Programatorii mai experimentați pot sări peste majoritatea exercițiilor de bază și pot rezolva probleme mai avansate. În acest fel, vom putea să ne adaptăm la abilitățile individuale ale fiecărui participant.

SISTEMUL MTASK

Microprocesoarele sunt sisteme de calculator cu capacități de calcul foarte limitate.

De obicei, acestea au o viteză destul de mică și restricții de memorie severe, având doar câțiva KB de memorie pentru a stoca date.

Aceste procesoare ieftine sunt esența a mai multor elemente top Internet of Things (IoT).

În astfel de sisteme trebuie să monitorizăm simultan mai multe porturi de intrare, precum să și controlăm unele ieșiri pe baza observațiilor din aceste porturi.

Datorită restricțiilor hardware, nu există de obicei un sistem de operare care să ofere suport.

Paradigma TOP oferă ”thread”-uri mici care sunt foarte potrivite pentru a monitoriza și a coordona progresul în astfel de sarcini simple. Aceste sarcini pot rula la viteza proprie în timp ce combinatoarele și sursele de date partajate sunt folosite pentru a le coordona în mod asincron. Rularea sistemului iTask pe dispozitivele IoT ne permite să construim sisteme care rulează parțial pe un server web, și pe dispozitive IoT. Limitările microprocesoarelor fac imposibilă rularea unui program \iTask complet pe dispozitive IoT.

Pentru a aproxima situația ideală, avem sistemul mTask [2,3]. Acesta este un DSL ”multiview”, care poate fi utilizat ca parte a sistemului iTask. Sistemul acceptă paradigma TOP, inclusiv aceleași rezultate ale activității ca sistemul iTask, combinatoarele de sarcini și surse de date partajate. Prin construcție, acest DSL nu are funcții de ordine superioare și nu are tipuri de date recursive. Datorită restricțiilor impuse în acest DSL, programele mTask pot fi compilate la codul care se execută pe microprocesoare.

În ciuda acestor restricții, DSL este foarte potrivit pentru a specifica sarcinile care trebuie executate pe dispozitivele IoT ușor și foarte concis.

TEHNICI FOLOSITE

Sistemul mTask este un DSL ”multiview” bazat pe clase de constructor de tip [1]. Fiecare instanță a acestor clase definește o interpretare, numită view, a unui program construit din aceste primitive. Vizualizările obișnuite implementează tipărirea destul de bună, generarea de coduri pentru microprocesoare și simularea programelor mTask ca un program iTask.

Acest DSL este extensibil prin construcție pentru a reutiliza bibliotecile existente pentru periferice, cum ar fi senzori de temperatură, afișaje și servomotoare.

Este simplu să adăugați o astfel de bibliotecă ca un limbaj primitiv la sistemul mTask, introducând o clasă de constructor de tip nou și instanțele necesare.

Pentru o implementare mTask portabilă și pentru a facilita reutilizarea bibliotecilor C++ existente, generarea de cod produce cod C++ pentru platforma Arduino în loc de cod de mașină nativ pentru un anumit procesor.

Compilatorul avr-gcc din platforma Arduino poate traduce codul C++ generat și bibliotecile utilizate pentru codul autohton pentru mai multe microprocesoare diferite.

PREDAREA LA ȘCOALA DE IARNĂ

A fi capabil să execute programe TOP la nivel înalt pe un mic microprocesor care interacționează cu perifericele este atrăgător pentru un tutorial în școala de iarnă.

Cu toate acestea, executarea unui program mTask pe un microprocesor real este dificil pentru o mare parte din studenți; ei trebuie să scrie un program mTask în sistemul iTask, să execute programul iTask pentru a obține codul C++, să alimenteze acest cod C++ la IDE Arduino, să conecteze ID-ul Arduino la microprocesor și să selecteze opțiunile potrivite, să încărcă programul compilat în microprocesor și, în cele din urmă să execute.

Toate aceste etape sunt destul de ușoare, dar întregul proces produce un rezultat doar atunci când fiecare pas este efectuat corect.

Întrucât programul generat va rula pe un microprocesor fără un sistem de operare și cu periferice de intrare/ieșire foarte limitate, depanarea unui astfel de program este dificilă.

După discuții ample, această secvență de pași a fost considerată a fi prea ambițioasă pentru timpul și audiența dată de școala de vară.

Din fericire, simulatorul sistemului mTask oferă o alternativă mai ușor de utilizat. Această vizualizare transformă programul mTask într-un program obișnuit iTask.

Simulatorul oferă o execuție pas cu pas a programului mTask: afișează o stare a ultimei etape sau a ultimei sarcini -- task -- executate și a stării tuturor perifericelor și a surselor de date partajate.

Ceasul, valoarea SDS-urilor, precum și starea perifericelor pot fi schimbate interactiv pentru a controla execuția și a investiga diverse scenarii.

Datorită celor de mai sus, activitatea practică a acestui tutorial este succesorul direct al tutorialului iTask.

Programarea acestor tutoriale este în același zi cu modulul iTask și cu modulul practic asociat. În acest fel, sesiunea \mTask se poate baza direct pe cunoștințele și abilitățile dobândite în sesiunea iTask.

Înțelegerea TOP-ului care s-a afirmat dimineața va fi aprofundată după-amiaza.

CONCLUZII

Pentru ambele tutoriale pe TOP există multe alte subiecte interesante decât pot fi abordate în timpul acordat pentru publicul școlii de iarnă.

În sesiuni, ne vom concentra pe înțelegerea și utilizarea paradigmei TOP prin exemple simple relative.

Exemple puțin avansate vor fi utilizate pentru a ilustra capacitățile acestei abordări.

În lucrarea practică, ne vom concentra pe exerciții ilustrative care sunt în mare parte variante de exemple utilizate în tutorial.

Pentru participanții avansați, vor exista câteva sarcini provocatoare, precum și oportunitatea de a discuta în profunzime aspecte ale sistemelor.

Referințe

[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

MOD DE PREDARE INTERACTIVĂ A REȚELELOR PETRI COLORATE

Metodele formale aparțin tehnicilor care, dacă sunt utilizate corespunzător, pot contribui la corectitudinea unui software sau al unui sistem hardware.

Una dintre metodele adecvate modelarea sistemelor concurente sau nedeterministice este limbajul de modelare cu Rețele Petri Colorate. În această lucrare descriem o activitate didactică pentru explicația principiilor de bază ale limbajului și a unor caracteristici funcționale vizavi de principiile de programare. Durata activității a fost de două ore și jumătate și a constat în construirea interactivă de modele cu participarea activă a publicului.

INTRODUCERE

Având în vedere dependența din ce în ce mai mare a societății contemporane de sistemele informatice, corectitudinea acestora ar trebui să fie de maximă importanță. Una dintre abordările care pot contribui în mod semnificativ la corectitudinea acestora este utilizarea metodelor formale în timpul dezvoltării de software și hardware. O metodă formală este o tehnică bazată matematic, care furnizează un limbaj formal cu sintaxă și semantică definite fără ambiguitate și un aparat, care permite efectuarea de sarcini de verificare, dezvoltare și simulare cu specificațiile sistemului, scrise în limbaj. Unul dintre membrii importanți ai familiei de metode formale este limbajul de modelare ”Coloured Petri Nets” (CPN). CPN combină formalismul rețelelor Petri [1] cu un limbaj funcțional pentru gestionarea procedurilor de manipulare și de decizie a datelor. Limbajul funcțional se numește CPN ML și este o versiune ușor modificată a limbajului ”Standard ML”. Limbajul CPN și sarcinile de specificație, verificare și simulare corespunzătoare sunt acceptate de software-ul CPN Tools.

De mai bine de un deceniu, CPN este o parte a cursurilor de licență legate de metodele formale, modelarea și simularea la instituția de origine a autorului. Una dintre metodele, aplicată de autor la explicarea conceptelor CPN este o abordare interactivă cu o participare activă a publicului. Aici, publicul alege domeniul și procesul pentru care va fi proiectat un model CPN și ajută la crearea părților sale selectate. Experiența dintr-o implementare particulară a acestei abordări într-o activitate de formare pentru profesorii universitari este descrisă în restul lucrării.

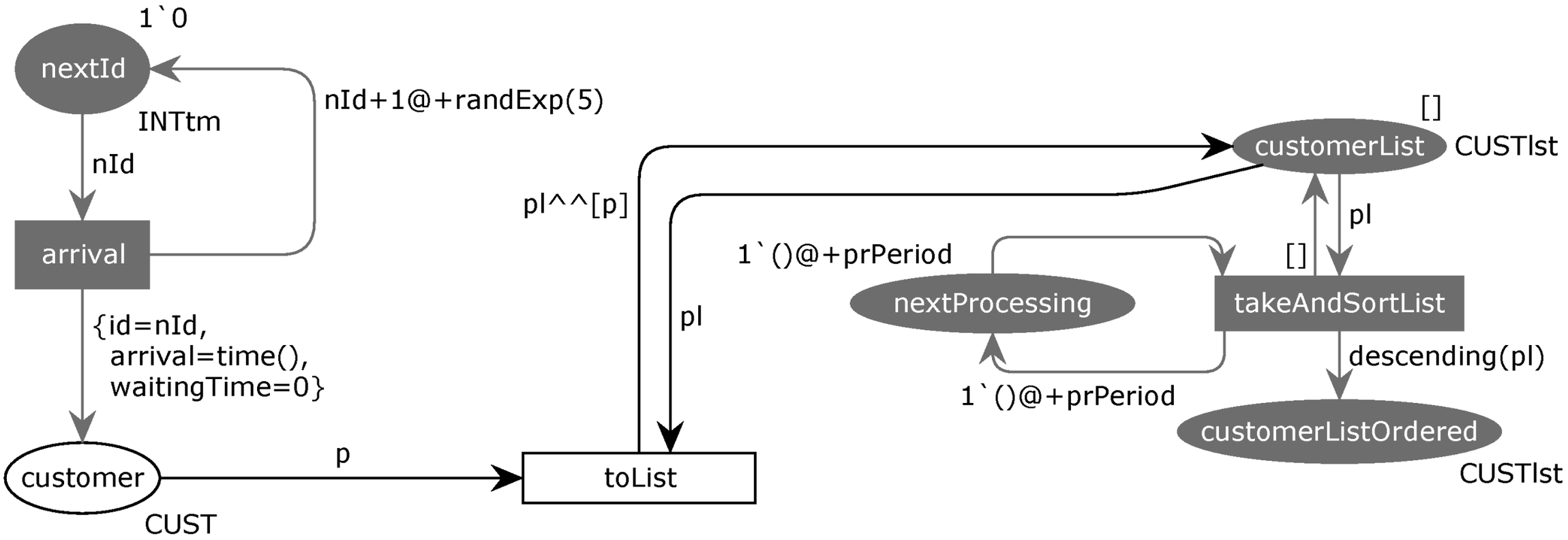
ACTIVITATE DE PREDARE CU MODELE CPN INTERACTIVE

Activitatea de instruire a fost organizată pentru aproximativ 10 participanți, care erau profesori universitari cu anumite limbi funcționale. Participanții s-au limitat la nicio cunoștință anterioară de CPN. Durata totală a activității a fost de aproximativ 2,5 ore, cu excepția pauzelor și a fost împărțită în trei faze.

Prima fază a durat aproximativ 30 de minute și a explicat principiile de bază ale CPN. Și anume că acel CPN are o formă grafică, un grafic bipartit cu două tipuri de vârfuri: locuri, desenate ca elipse și tranziții, desenate drept dreptunghiuri. Locurile conțin jetoane, care reprezintă o stare a plasei și tranzițiile pot fi înțelese ca evenimente, care schimbă starea consumând jetoane existente și creând altele noi.

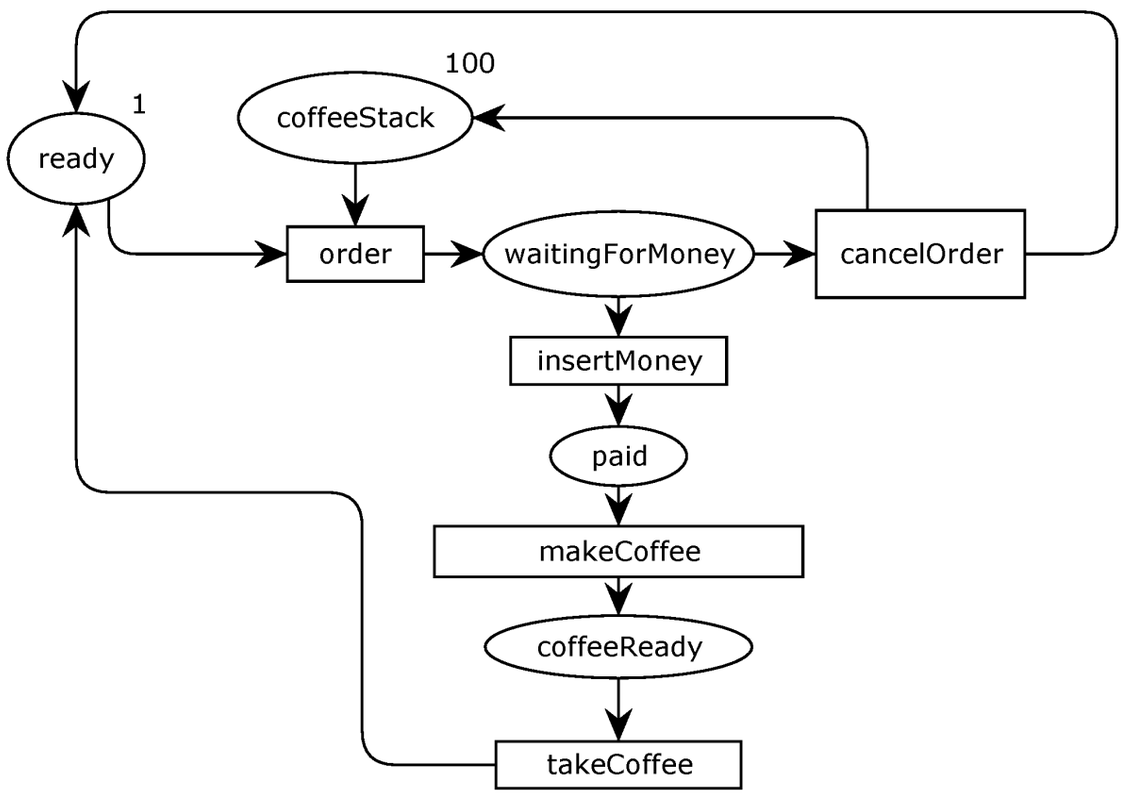
A doua și a treia fază au fost dedicate creării unui model CPN. Întrucât unul dintre obiectivele activității a fost să arate modul în care unele concepte Standard ML mai avansate, și anume structuri și functori, pot fi utilizate în modelele CPN, participanții au primit un model CPN de început, care deja a folosit conceptele, înainte de al doilea faza începută. Modelul de pornire este prezentat în Fig. 1.

Partea constând din nodurile |nextId|, |arrival| și |customer| reprezintă o sosire a clienților, care ajung unul câte unul pentru a fi deserviți. Servirea în sine nu este prezentată în modelul de pornire. În schimb, există tranziția |toList |, care ia un simbol din |customer| și își adaugă valoarea într-o listă, ținută în loc |customerList|.



Tranziția |takeAndSortList| este tras la intervale regulate, definite prin valoarea | prPeriod|. Fiecare tragere de | takeAndSortList| goleste lista in |customertList|, sortează conținutul acesteia și stochează versiunea comandata în | customerListOrdered|. Locul |nextProcessing| este una auxiliară și se asigură că |takeAndSortList| este rulat numai la intervale regulate.

Sortarea este asigurată de o funcție numită |descending|, care implementează algoritmul Quicksort. Funcția utilizează structuri standard ML și functori.



Pentru partea de servire, participanții la activitate au decis să modeleze un automat de cafea (coffee vending machine). În a doua fază au participat la crearea unui model CPN care să realizeze funcționarea de bază a mașinii.

Modelul este prezentat în Fig. 2. În starea sa inițială, mașina este gata să servească un client (un token în locația |ready|) și este umplută cu 100 de doze de cafea (100 de jetoane în |coffeeStack|). Servirea începe cu un client care comandă o cafea prin apelul tranziției |order|. Apoi, aparatul așteaptă următorul pas al clientului (un jeton în |waitForMoney|). Clientul poate introduce bani (prin rularea |insertMoney|) sau anula comanda (prin rularea |cancelOrder|).

Anularea returnează aparatul în starea ``ready''.

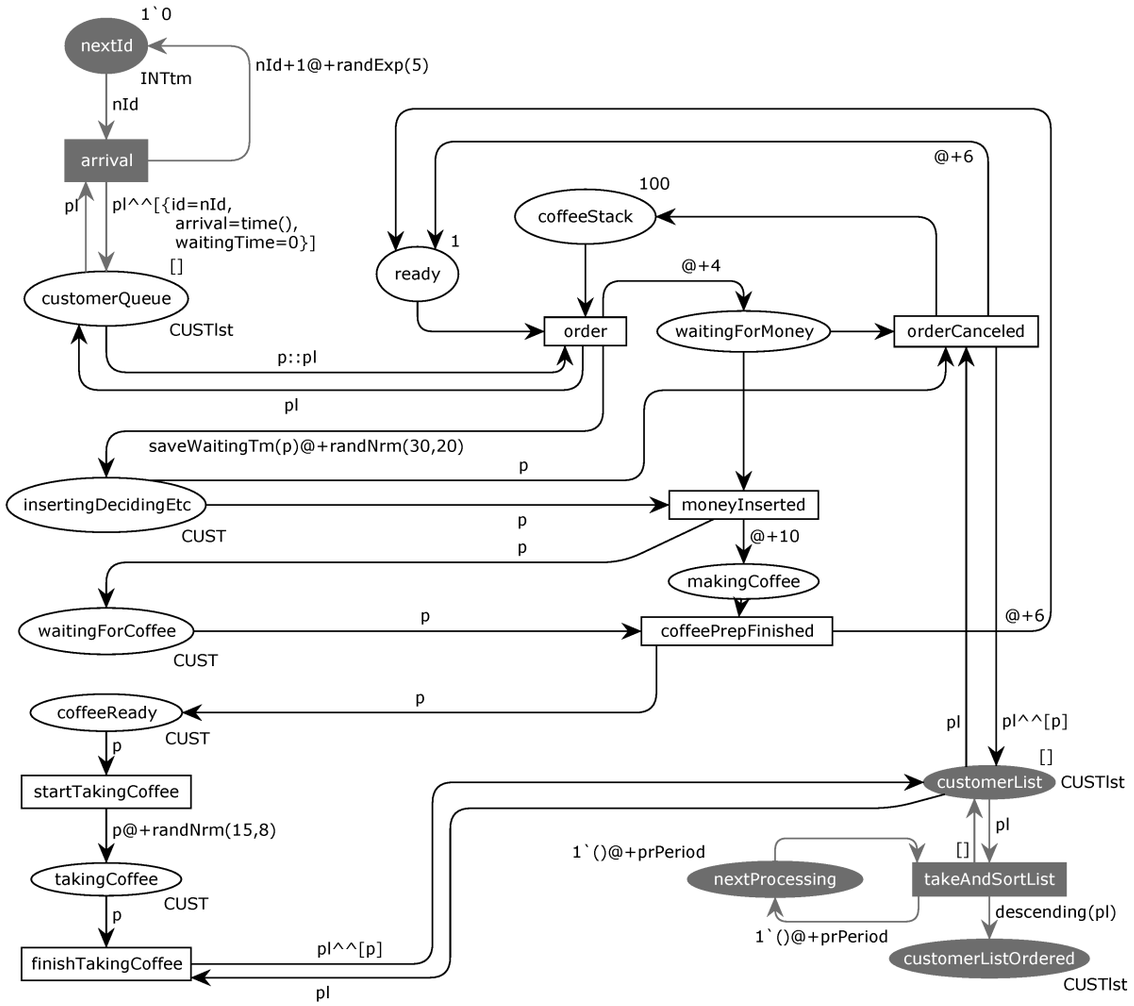
Dacă banii sunt introduși, mașina pregătește cafeaua (prin rularea |makeCoffee|). În final, printr-o secvență |takeCoffee|, clientul servește cafeaua pregătită, iar aparatul revine la starea de ”ready”.

După a doua fază au fost o pauză lungă de aproximativ 70 de minute. În timpul pauzei, profesorul a conectat modelul din faza 2 la părțile din modelul de pornire și a adăugat vârfuri și arcuri care descriu comportamentul clientului. De asemenea, instructorul a corectat unele neconcordanțe în model, care a fost menționat de unul dintre participanți.

Modelul CPN rezultat -- cel final -- poate fi văzut în Fig. 3. Vârfurile preluate de la modelul de pornire (Fig. 1) fără modificare sunt redate în gri.

Locul |customer| este înlocuit cu |customerQueue|, care ține un jeton cu o listă de valori, reprezentând o coadă de clienți care așteaptă la mașină. În loc de tranziția |toList| există o parte de servire, creată din rezultatul celei de-a doua faze (Fig. 2). Partea care servește la modelul final diferă de Fig. 2 în trei aspecte cheie:

- Jetoanele poartă informații despre clientul servit, iar expresiile de pe arc definesc durata acțiunilor corespunzătoare.



- Incoerențele privind rolul locurilor și tranzițiile sunt corectate. Acum, toate tranzițiile reprezintă evenimente instantanee. De exemplu, tranziția |makeCoffee| din Fig. 2 se înlocuiește cu locația |makingCoffeee| și tranziția |takeCoffee| se înlocuiește cu vertexurile |startTakingCoffee|, |takingCoffee| și |finishTakingCoffee|.

- Acțiunile și stările clienților și ale mașinii sunt modelate separat. Vârfurile |customerQueue|, |insertingDecidingEtc|, |waitForCoffee|, |coffeeReady|, |startTakingCoffee|, |takingCoffee| și |finishTakingCoffee| aparțin clienților în timp ce restul piesei de servire reprezintă mașina sau ambele părți.

A treia fază a activității de instruire a fost dedicată explicării modelului final și unei discuții despre locul acestor modele în dezvoltarea sistemelor corecte de calculatoare. A durat aproximativ 30 de minute.

CONCLUZIE

Activitatea de instruire interactivă, prezentată aici, este potrivită pentru cursuri scurte și intense, care se desfășoară adesea în timpul școlilor de vară sau a altor evenimente didactice similare. Procesul de test descris al activității a relevat că repartizarea inițială a timpului, care a fost de 2 ore, nu a fost suficientă. Prin urmare, a fost necesară a treia etapă, unde instructorul a prezentat modelul final. Având în vedere timpul necesar pentru construirea modelului final de către instructor, va fi nevoie de încă cel puțin două ore pentru a realiza întregul proces de creare a modelului în mod interactiv cu audiența cursului.

Toate modelele CPN prezentate sau menționate aici pot fi obținute la cererea autorului.

Referințe

[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: UN SISTEM EXTENSIBIL DE ANALIZĂ AL CODULUI SURSĂ

CodeCompass este un instrument open-source pentru a ajuta înțelegerea sistemelor software vechi. Bazat pe infrastructura compilatorului LLVM/Clang, CodeCompass oferă informații despre elementele complexe ale limbajului de programare C/C++. Gama largă de vizualizări include diagrame de apeluri de clasă și de funcții; diagrame arhitecturale, componente și interfață și diagrame ``pointere'', etc.

CodeCompass utilizează informații de compilare pentru a explora arhitectura sistemului, precum și informații de control al versiunilor, dacă avem disponibil. Rezultatele analizei statice bazate pe CLANG sunt de asemenea integrate în sistem. Deși instrumentul este bazat pe C și C++, acesta acceptă și limbile Java și Python. Având o arhitectură extensibilă, bazată pe web și plugin-uri, CodeCompass poate fi o platformă deschisă pentru îmbunătățirea înțelegerii codului, analizei statice și eforturilor de măsurare a software-ului.

INTRODUCERE

Corecția erorilor sau dezvoltarea noilor funcționalități necesită o înțelegere a detaliilor și consecințelor modificărilor planificate. Instrumentele de înțelegere a codului pot ajuta la descifrarea intențiilor originale și a detaliilor de implementare a sursei prin construirea unui model din codul sursă și alte informații disponibile. Deși o serie de astfel de instrumente sunt disponibile; fie ca software proprietar sau unul gratuit, setul lor de funcționalități este limitat.

CodeCompass a fost dezvoltat pentru a elimina aceste restricții. Proiectul este efort comun open source al Ericsson Ltd. și Eötvös Loránd University, Budapesta pentru a ajuta la înțelegerea sistemelor software mari.

Pentru a furniza informații exacte despre elemente complexe din C/C++ precum supraîncărcarea, moștenirea, utilizarea variabilelor și a tipurilor, utilizarea pointer-elor și a funcțiilor virtuale - diverse instrumente le acceptă doar parțial - CodeCompass se bazează pe un compilator real, LLVM/Clang. Prin aceasta se elimină punctele slabe ale instrumentelor obișnuite, cum ar fi OpenGrok.

Cu toate acestea, CodeCompass nu este limitat la codul sursă; aceasta folosește informațiile de construire ale sistemului pentru a dezvălui conexiunile arhitecturale.

Pentru a ajuta la percepția rapidă și precisă, CodeCompass folosește atât reprezentarea textuală cât și cea grafică a sistemului software pentru a înțelege. O serie de diagrame (interactive) sunt accesibile din graficele de apeluri funcționale obișnuite la diagramele arhitecturale unice.

Pentru a oferi acces facil utilizatorilor, CodeCompass are o arhitectură bazată pe web. Clientul poate fi un browser web standard, un plug-in pentru editor sau orice altă aplicație terță parte. Comunicarea se bazează pe o API REST și se scalează bine pentru solicitările clientului paralel.

În această lucrare vom compara CodeCompass cu instrumentele de înțelegere existente și vom descrie setul de caracteristici. În secțiunea 2 prezentăm o privire de ansamblu asupra principalelor arhetipuri de instrumente existente pentru înțelegerea codurilor. Introduceți arhitectura extensibilă a CodeCompass în 3. Principalele caracteristici ale instrumentului sunt discutate în secțiunea 4. Rezumăm lucrarea din secțiunea 5.

RELATED WORK

Pe piața software sunt mai multe instrumente pentru înțelegerea codului sursă. Unele au analiză statică, altele examinează și comportamentul dinamic al programului. Putem să împărțim în diferite tipuri, bazat pe arhitectură și pe principiile de funcționare. Există instrumente cu arhitectură server-client: ele analizează proiectul și stochează toate informațiile necesare într-o bază de date. Clienții (de obicei web-based) sunt serviți din baza de date. Aceste instrumente pot fi integrate în fluxul de lucru prin CI nocturn.

În acest fel, dezvoltatorii pot naviga și analiza întotdeauna. De asemenea, există aplicații de client în care o parte mică din cod este analizată. Acesta este cazul IDE-urilor în care modificarea frecventă a sursei necesită o actualizare rapidă a bazei de date da analiză. În această secțiune prezentăm câteva instrumente utilizate în mediul industrial din fiecare categorie.

Woboq [3] este un browser de cod bazat pentru C/C++. Acest instrument are funcții extinse care urmăresc navigarea rapidă a unui proiect software.

Putem găsi rapid fișierele și entitățile cu un câmp de căutare care oferă completarea codului pentru o utilizare ușoară. Navigarea în baza de cod se face printr-o pagină web pe fișiere HTML statice. Aceste fișiere sunt generate în timpul analizei. Avantajul acestei abordări este că clientul web este rapid, deoarece nu sunt necesare calcule pe server în timpul navigării.

De exemplu, cu mouse-ul pe o funcție, o clasă, o variabilă, un macro, etc -- se afișează proprietățile acelui element. De exemplu, în cazul funcțiilor, se poate vedea semnătura acestuia, locul definiției și al folosirii.

Pentru clase se poate verifica dimensiunea obiectelor, aspectul clasei și diagrama moștenirii. Pentru variabile se poate inspecta tipul și locațiile unde sunt scrise sau citite.

În C/C++ macro-urile formează un sub-limbaj care este evaluat într-o etapă de precompilare. Această evaluare înseamnă că faza de compilare funcționează cu un alt cod decât în original. În Woboq, valoarea finală a expansiunilor de macro poate fi de asemenea inspectată.

O caracteristică foarte utilă a instrumentului este evidențierea semantică. Prin această caracteristică se pot distinge cu ușurință diferite elemente de limbaj: formatarea variabilelor locale, globale sau membre, funcții virtuale, tipuri, typedef-uri, etc.

Woboq poate furniza caracteristicile menționate mai sus, deoarece informațiile necesare sunt colectate într-o fază reală de compilare. Proiectul examinat trebuie mai întâi compilat și analizat de Woboq. Analiza este realizată prin infrastructura LLVM/Clang, construiește întreaga arbore de sintaxă abstractă. Astfel, toate informațiile semantice pot fi extrase. De aici dezavantajul principal: Woboq poate fi utilizat doar pentru navigarea în proiectele C și C++.

OpenGrok [4] este un motor de căutare rapidă a codului sursă. Contrar sistemului Woboq, aici nu avem o analiză detaliată, prin urmare nu este capabil să furnizeze informații semantice despre entități. În schimb, folosește Ctags [5] pentru a analiza doar textul codului sursă și pentru a determina tipul elementelor. Analiza sintactică simplă permite identificarea funcțiilor, variabilelor sau claselor, etc. Căutarea este extrem de optimizată și, prin urmare, foarte rapidă chiar și pe baze de coduri mari. Căutarea se poate realiza prin expresii compuse (de exemplu, \texttt{defs: target}), care conțin chiar și wild-card-uri, în plus, rezultatele pot fi limitate la subdirectori. Pe lângă căutarea textului, există posibilitatea de a găsi simboluri sau definiții.

Lipsa analizei semantice permite Ctags să susțină mai multe (41) limbaje de programare. Un avantaj al acestei abordări este actualizarea treptată a bazei de date a indexului. OpenGrok oferă, de asemenea, oportunitatea de a colecta informații din sistemele de control de versiuni precum Mercurial, SVN, CSV etc.

Understand [6] nu este doar un instrument de navigare a codului, ci și un IDE complet, având avantajul editării codului sursă și rezultatele modificărilor pot fi văzute imediat.

Pe lângă funcțiile de navigare prin cod menționate deja pentru instrumentele anterioare,

Understand oferă o mulțime de metrici și de rapoarte: liniile de cod (total/mediu/ maxim la nivel global sau pe clasă), numărul de clase cuplate/de bază/ derivate, lipsa de coeziune [2], complexitatea McCabe [1] și multe altele. Treemap este o metodă comună de reprezentare pentru toate valorile. Este o vedere dreptunghiulară ierarhică a elementelor, iar dimensiunile și culorile reprezintă metrica aleasă de utilizator.

Pentru bazele de coduri mari, inspecția arhitecturii este necesară. Sistemul ”Understand” poate afișa diagrame de dependență bazate pe diverse relații, cum ar fi ierarhia apelurilor funcționale, moștenirea clasei, dependența fișierului, includerea/importul. Utilizatorii pot crea, de asemenea, tipul diagramei personalizate prin API-ul furnizat de instrument. În programare, conceptele de bază sunt comune între limbi, dar există unele concepte care sunt interpretate diferit într-un anumit limbaj. Sistemul ”Understand” poate gestiona 15 limbaje de programare și poate furniza informații specifice limbii despre cod, de ex. analiza indicatoarelor funcționale în diagrama C/C++ sau a ierarhiei pachetelor în Ada.

Sistemul "Understand" construiește o bază de date din cod. Toate informațiile pot fi culese prin intermediul unei API. Astfel, utilizatorul poate interoga toate informațiile necesare care nu sunt incluse în vizualizare.

CodeSurfer [7] este similar cu "Understand" în sensul că este o aplicație de analiză statică. Obiectivul său este înțelegerea proiectelor în cod C/C++ sau x86. CodeSurfer realizează o analiză profundă a limbajului care oferă informații despre comportamentul software-ului. De exemplu, implementează o analiză a pointer-elor, listează instrucțiunile care depind de o declarație selectată prin analiza impactului și folosește analiza fluxului de date pentru a identifica unde i-a fost atribuită o variabilă etc.

ARHITECTURA SISTEMULUI CODECOMPASS

În secțiunea anterioară am enumerat câteva aspecte privind obiectivele și arhitecturile instrumentelor de înțelegere a codurilor; aici prezentăm sistemul CodeCompass.

CodeCompass are o arhitectură client-server care prezintă informațiile culese într-o fază anterioară de analiză. Motivul pentru care am ales această arhitectură este că, spre deosebire de editorii de coduri, CodeCompass a fost planificat să fie un instrument de înțelegere a codului.

Există diferențe fundamentale între aceste două cazuri de utilizare. În timpul scrierii codurilor, programatorii manipulează simultan doar câteva fișiere. În înțelegerea codului, însă, este necesar să se ia în considerare sursele mai multor module. În editoare completarea codului este una dintre cele mai utile caracteristici: programatorul nu dorește să-și amintească toate metodele și câmpurile unei clase, dar cere editorului să le enumere. În înțelegerea codului, este nevoie de o gamă largă de vizualizări. În timp ce editează sursa, programatorul se concentrează doar pe un fragment relativ mic al codului, cum ar fi o funcție sau o clasă. În înțelegerea codului, analiza nu este doar la nivelul funcțiilor, ci și dependențele și efectele acestora sunt luate în considerare la un nivel mai înalt.

Interfața principală de utilizare a CodeCompass este bazată pe web. Toate vizualizările și funcționalitățile menționate mai sus pot fi interogate printr-o API publică care este atribuită unei aplicații server. Interfața web gestionează cazurile de utilizare care vizează sarcini de navigare, inspecție și înțelegere rapide și la îndemână. Cu toate acestea, CodeCompass este mai mult decât un instrument de navigare a codului. Este, de asemenea, un cadru, adică un colector extensibil și prezentator al proceselor de analiză statică. De aceea, intenția nu a fost de a crea o aplicație grea pentru client, care să stocheze rezultatele analizei pe partea clientului, ci să poată satisface diferitele nevoi ale utilizatorilor. Astfel, este posibil să implementăm un script, de exemplu, care colectează setul de funcții care formează o închidere prin relația de apel funcțional, specificând astfel o felie coerentă a software-ului.

O altă cerință de proiectare a CodeCompass a fost să se ocupe de baze de cod pe scară largă și să răspundă la solicitările utilizatorului foarte rapid, adică în termeni de secunde cel mult. Acest lucru se realizează prin stocarea tuturor celor mai mici cantități de informații într-o bază de date, care sunt suficiente pentru a răspunde solicitărilor. Întrucât intenționam să oferim rezultate precise pentru interogări, este necesar un proces de analiză precedent. În primul, am stocat întregul arbore de sintaxă abstract al sursei, dar a rezultat un raport 1: 1000 între codul sursă și dimensiunea bazei de date. Cu toate acestea, s-a dovedit că majoritatea cazurilor în care utilizatorii sunt interesați doar de entități numite (funcție, variabile, clase, macro-uri etc.), astfel încât nu a fost necesar să stocați altceva, cum ar fi structurile de control sau alte declarații. Cu toate acestea, există unele sarcini care necesită mai mult decât informațiile stocate, cum ar fi un algoritm de tranșare. Dacă utilizatorul dorește să vadă efectele schimbării valorii unei variabile, atunci trebuie luate în considerare și declarațiile de modificare a stării. Aceasta necesită repararea codului din mers.

CARACTERISTICILE SISTEMULUI CODECOMPASS

În această secțiune vom oferi o imagine de ansamblu despre caracteristicile disponibile prin GUI standard.

Când descriem caracteristici specifice limbajului, precum listarea apelurilor unei metode, vom presupune întotdeauna limbajul proiectului este C++, deoarece are cel mai avansat suport în CodeCompass, dar funcții similare sunt disponibile pentru Java și Python.

Căutare

Probabil cea mai frecventă utilizare al unui instrument de înțelegere este căutarea. Se poate căuta fie un fișier, fie un cod sursă. Pentru găsirea elementelor de cod sursă, instrumentul oferă 3 posibilități de căutare diferite:

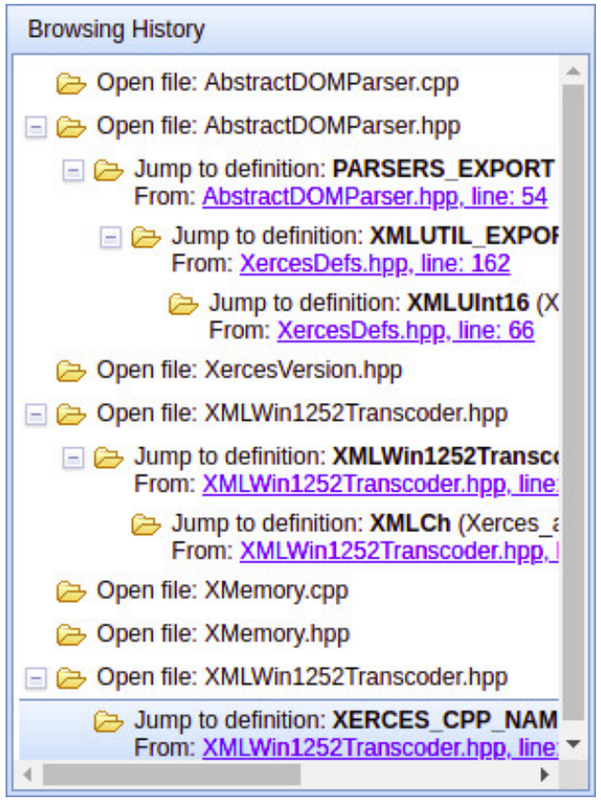
În modul căutare text completă -- full text --, fraza de căutare este un grup de cuvinte, cum ar fi ``returnează un astnode*''. O frază de interogare se potrivește cu un bloc de text, dacă cuvintele căutate sunt unul lângă altul în codul sursă în aceeași ordine. Caractere ”wildcards”, cum ar fi ”*”, sau ”?” pot fi folosiți, conform definițiilor. Operatorii logici, precum AND, OR, sau NOT pot fi folosiți pentru a compune mai multe fraze de interogare.

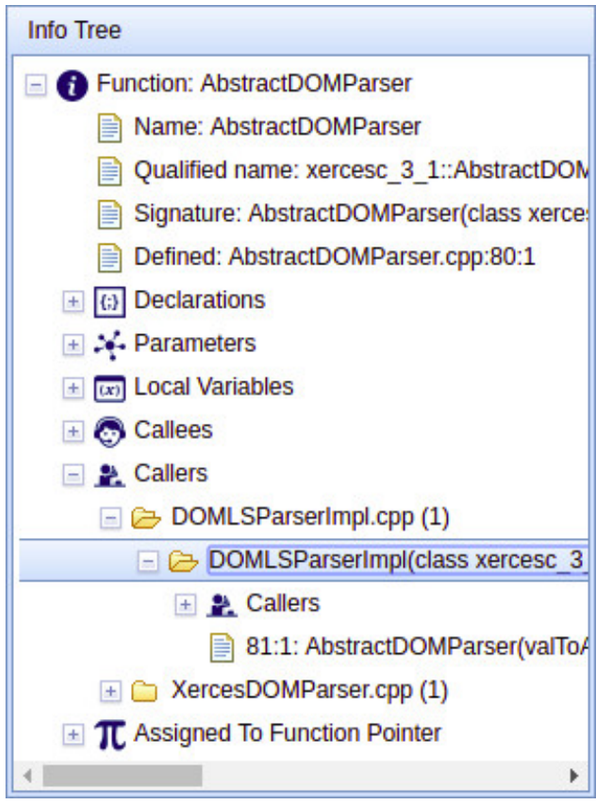
La un nivel superior este posibil să găsiți simboluri în codurile sursă prin definition search. Aici folosim CTag-uri pentru indexarea bazei de coduri, astfel încât să putem găsi variabile, funcții, clase, macro-uri etc.

În timp ce depanați un program, uneori singura informație pentru a începe este un mesaj de LOG. Aceasta este singura urmă de unde poate începe, de ex. "DEBUG INFO: TSTHan: sys _offset = -0.019821, drift _comp = -90.4996, sys _poll = 5". Este important că mesajul poate conține timestamp-uri sau alte fragmente generate dinamic, astfel încât este imposibil să găsiți acest mesaj ca o șir direct. Cu toate acestea, în CodeCompass o căutare confuză se poate face prin log search.

Informații legate de simboluri lingvistice

Când elementul a fost găsit, următorul pas este colectarea informațiilor despre acesta. Utilizatorul poate alege ``Info tree'' din meniul pop-up după ce a selectat o entitate numită. Acest arbore conține toate informațiile furnizate de un parser de limbă. În cazul C/C++, utilizăm compilatorul LLVM/Clang pentru a obține informații despre simboluri. Pentru funcții le putem verifica parametrii, variabilele locale, apelanții și apeluri. O caracteristică interesantă a arborelui este că apelanții sunt prezentați recursiv, adică copiii unui nod sunt apelanții unei funcții. Nodurile copiilor lor sunt apelanții acestor funcții, iar acest lucru continuă recursiv, teoretic, revenind la funcția principală.





Cu toate acestea, apelurile funcționale nu sunt întotdeauna directe, dar se pot întâmpla prin indicatoarele funcționale. Chiar dacă acesta este un comportament dinamic, CodeCompass convoacă toate aparițiile în care o funcție a fost atribuită unui pointer funcție și invocarea se întâmplă prin acest pointer.

În cazul claselor, informațiile colectate sunt pseudonimele (prin typedef clasa poate avea un sinonim), relații de moștenire (grupate după vizibilitate), prieteni, metode/câmpuri (directe sau moștenite) și utilizări (ca local/global, parametru funcție/tip retur sau câmp al altei clase).

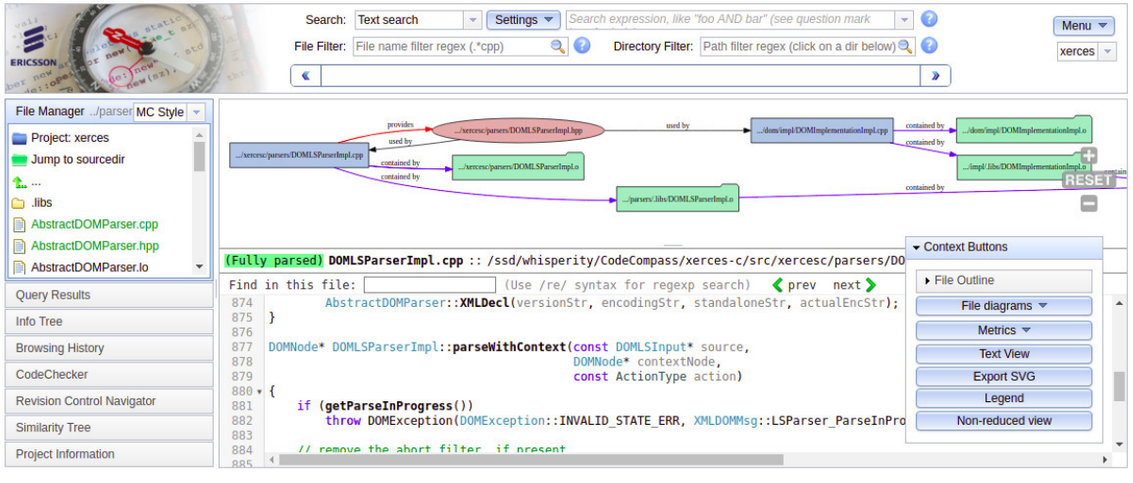
Pentru variabile este util să cunoaștem locurile din codul în care a fost scris și citit. Pentru tipurile de enumerare, constantele de enumerare sunt listate cu valorile lor întregi.

Diagrame

Vizualizările sunt cele mai utile reprezentări pentru a analiza un sistem. CodeCompass prezintă mai multe diagrame bazate pe simboluri și fișiere. Aceste diagrame sunt bazate pe grafic, adică reprezintă entități și conexiunile lor.

Acestea sunt diagrame interactive: trecând mouse-ul peste nodurile, entitatea reprezentată este afișată în vizualizarea textului, iar prin clic pe ele, entitatea selectată devine nodul central care arată relațiile sale în funcție de tipul diagramei.

Diagrama funcției apelante arată toate apelanții unei funcții dintr-un grafic. Moștenirea clasei UML arată schema de moștenire completă până la clasa de bază rădăcină și recursiv pentru toate clasele derivate. De asemenea, am implementat o diagramă pointer analysis care prezintă obiectele alocate și indicatoarele care le pot indica.



Desigur, aceasta este o informație dinamică care poate fi colectată doar parțial într-o analiză statică.

O diagrama de interfață solicitată pentru un fișier sursă C/C++ arată ce anteturi sunt ``utilizate doar'' sau ``implementate'' de fișierul dat. Utilizare înseamnă că un fișier sursă folosește un alt fișier dacă există un simbol al acestuia, care este declarat în celălalt fișier. Relația de implementare înseamnă că un simbol este declarat într-un fișier (formând astfel o interfață) și definit într-un alt. Aceste relații sunt aplicabile și pentru directorii care iau în considerare fișierele conținute.

În cazul unui limbaj compilat, există și fișiere de ieșire ca obiecte și executabile. Pe baza informațiilor de legătură, putem prezenta sursele care alcătuiesc un fișier binar.

CodeBites oferă o vizualizare diferită a codului sursă inspectat. În această perspectivă, nodurile graficului sunt definițiile unor simboluri numite specifice, cum ar fi clase, funcții, etc. Ideea este că un programator ar dori să descopere această entitate prin înțelegerea comportamentului, dar fără a pierde accentul. Deci, părțile textului codului dintr-un nod pot fi făcute clic, ceea ce declanșează adăugarea definiției elementului selectat.

Vizualizări a versiunilor

Vizualizarea informațiilor despre controlul versiunilor este un ajutor important pentru înțelegerea evoluției software-ului. Git blame view arată linie cu linie modificările (comitele) la un fișier dat. Modificările care s-au întâmplat recent sunt de culoare verde mai deschis, în timp ce modificările mai vechi sunt de un roșu mai închis. Această vizualizare este excelentă pentru a examina motivul pentru care anumite linii au fost adăugate la un fișier sursă. CodeCompass poate de asemenea afișa Git-ul comit într-o listă filtrabilă. Această facilitate de căutare poate fi utilizată pentru a enumera modificările făcute de o persoană sau pentru a filtra angajamentele prin cuvinte relevante din mesajul de angajare.

Metrici soft

CodeCompass poate afișa complexități: McCabe Cyclomatic Complexity [1], liniile de cod și numărul de erori găsite de "Clang Static Analyzer" pentru fișiere individuale și rezumate pe ierarhiile directoarelor. Aceste valori pot fi vizualizate pe un arbore, unde directoarele sunt indicate prin căsuțe. Mărimea cutiei și nuanța ei de culoare este proporțională cu metrica aleasă.

Browsing history

De Alwis și Murphy au studiat de ce programatorii se confruntă cu dezorientarea atunci când folosesc mediul de dezvoltare integrată Eclipse Java (IDE) [8]. Folosesc tehnica momentului vizual [9] pentru a identifica trei factori care pot duce la dezorientare: i) absența conectării contextului de navigație în timpul explorării programului, ii) care se întinde între afișaje pentru a vizualiza bucățile de cod necesare și iii) urmărirea unor subtask-uri uneori fără legătură. Primul factor înseamnă că programatorul, în timpul investigării unei probleme, vizitează mai multe fișiere după cum urmează un lanț de apeluri sau explorează utilizarea unei variabile.

La sfârșitul unei ședințe lungi de explorare, este greu de reținut de ce ancheta a ajuns într-un dosar specific. Al doilea motiv pentru dezorientare este schimbarea frecventă a diferitelor puncte de vedere în Eclipsa. Al treilea contribuitor la problemă este faptul că un dezvoltator, atunci când rezolvă o sarcină de schimbare a programului, evaluează mai multe ipoteze, care sunt toate subtask-uri individuale de înțelegere. Programatorii tind să suspende un subtask (înainte de a o termina) și să treacă la alta.

De exemplu, programatorul investighează modul în care se folosește o valoare de retur a unei funcții, dar apoi se schimbă la un subtask care înțelege implementarea funcției în sine. S-a observat că, pentru un dezvoltator, este greu să-și reamintească despre o subtask suspendat.

CodeCompass implementează o vizualizare istoricul navigării care înregistrează (într-o formă de arbore) calea de navigare în codul sursă. Un nou subtask este reprezentată de o nouă ramură a arborelui, în timp ce nodurile sunt salturi de navigare în codul etichetat de contextul de conectare (cum ar fi ``salt la definiția init''). Deci problema i) și ii) este abordată de nodurile etichetate din istoricul de navigare, în timp ce problema iii) este tratată de ramurile alocate subtask-urilor.

CodeChecker - raport erori C/C++

"Clang Static Analyzer" implementează un motor avansat de execuție simbolică pentru a raporta erori de programare. CodeCompass poate vizualiza erorile identificate de "Clang Static Analyzer" și "Clang Tidy" conectându-l la un server CodeChecker [11].

CodeCompass arată poziția erorilor și calea de execuție simbolică care duce la o defecțiune.

Namespace-uri și tipuri

Procesele CodeCompass procesează documentații Doxygen și le stochează pentru definiția funcției, tipului, variabilei. De asemenea, oferă o vizualizare type catalog care listează tipurile declarate în spațiul de lucru organizat de o vizualizare arbore ierarhică a spațiilor de nume.

SUMAR

Am prezentat CodeCompass, un instrument de analiză statică pentru înțelegerea software-ului la scară largă. A fost conceput pentru a evita diferitele deficiențe ale instrumentelor de înțelegere existente, care sunt fie ușoare, ușor de utilizat, dar fără cunoașterea profundă a unui compilator real; sau grele, ne-scalabile instalate pe mașina client. Având o arhitectură extensibilă, extensibilă, bazată pe web, cadrul poate fi o platformă deschisă pentru a înțelege mai mult codul, analiza statică și eforturile de măsurare a software-ului.

Feedback-ul inițial al utilizatorilor și statisticile de utilizare sugerează că instrumentul este util dezvoltatorilor în activități de înțelegere și este utilizat pe lângă IDE-urile tradiționale.

Referințe

[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.