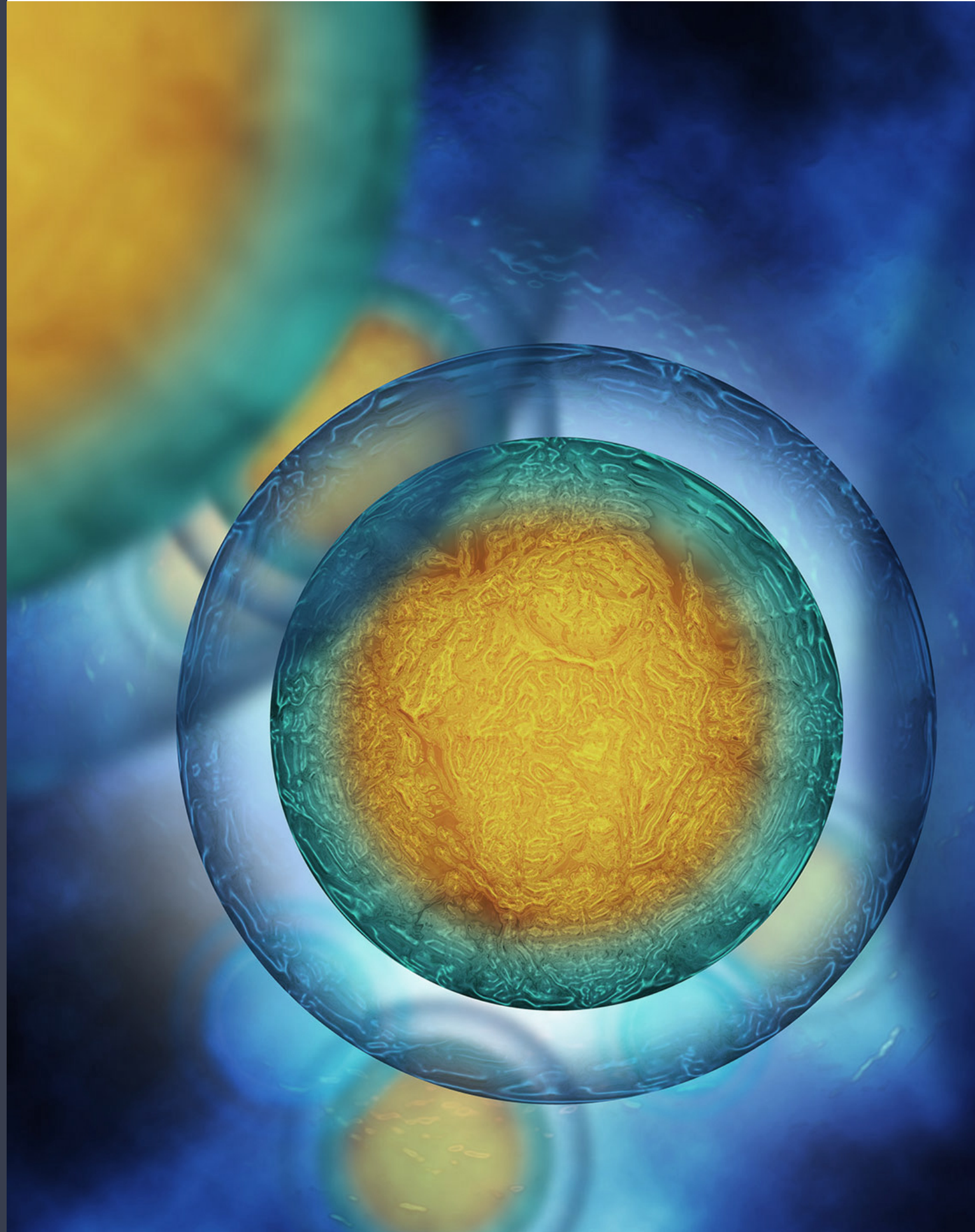
FE3CWS

МЕЖДУНАРОДНО ОБУЧЕНИЕ НА ПРЕПОДАВАТЕЛИ ПО КОМБИНИРАНОСТ, РАЗБИРАЕМОСТ И КОРЕКТНОСТ

НА РАБОТЕЩ СОФТУЕР

Интелектуален резултат 2 на проекта ERASMUS + 2017-1-SK01- KA203-035402



Някои думи по

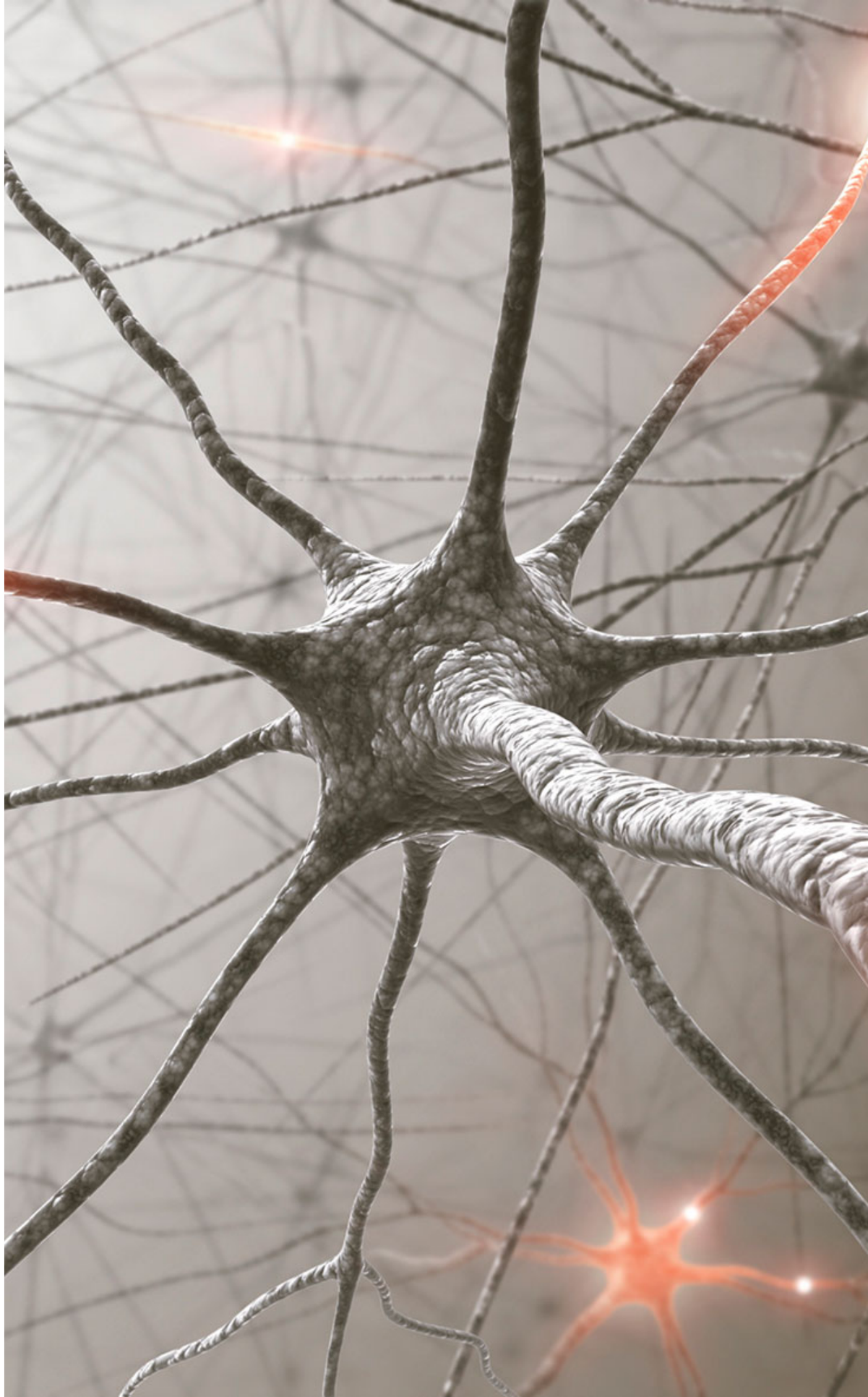
съдържанието

- **6 теми, свързани със състава на софтуера, разбирането и коректността**
- **Предлага се на 7 езика: английски, унгарски, словашки, хърватски, румънски, български и португалски**

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

Международно обучение на преподаватели по комбинираност, разбираемост и коректност на работещ софтуер

Съдържание



Облачни изчисления в образованието: фокус върху потребителя

Облачните изчисления се превърнаха в ключова технология и следователно част от много учебните програми по компютърни науки. Потребителско-ориентираните изследвания се фокусират върху стратегии за оценка на времетраенето и разходите за внедряване на приложения в реалния свят в облака. В областта на облачните технологии важен аспект е подпомагането на потребителите при вземането на решения. Такива решения отговарят на следните въпроси:

- Как работи приложението върху виртуализираните ресурси?

- Колко виртуални ресурси, от какъв тип, от кой облачен доставчик трябва да бъдат придобити за внедряване на приложения?

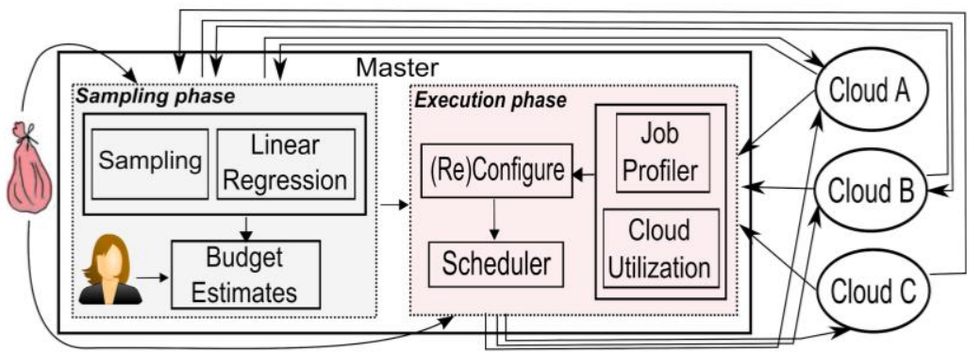
- За колко дълго? Колко ще струва?

Тези въпроси обикновено се моделират като проблем при планирането, като се работи с предположението, че няма предходни познания за приложението. Едно от типичните, семпли изисквания е, че приложението трябва да е успешно внедрено и разходите са сведени до минимум.

Архитектурата на планировчик за облачно приложение

Планировчикът на BaTS [4] е разработен, за да помогне на потребителите при разполагане на приложенията в облака. За постигането на това е необходим подход за самопланиране и той редовно проверява напредъка на внедряването.

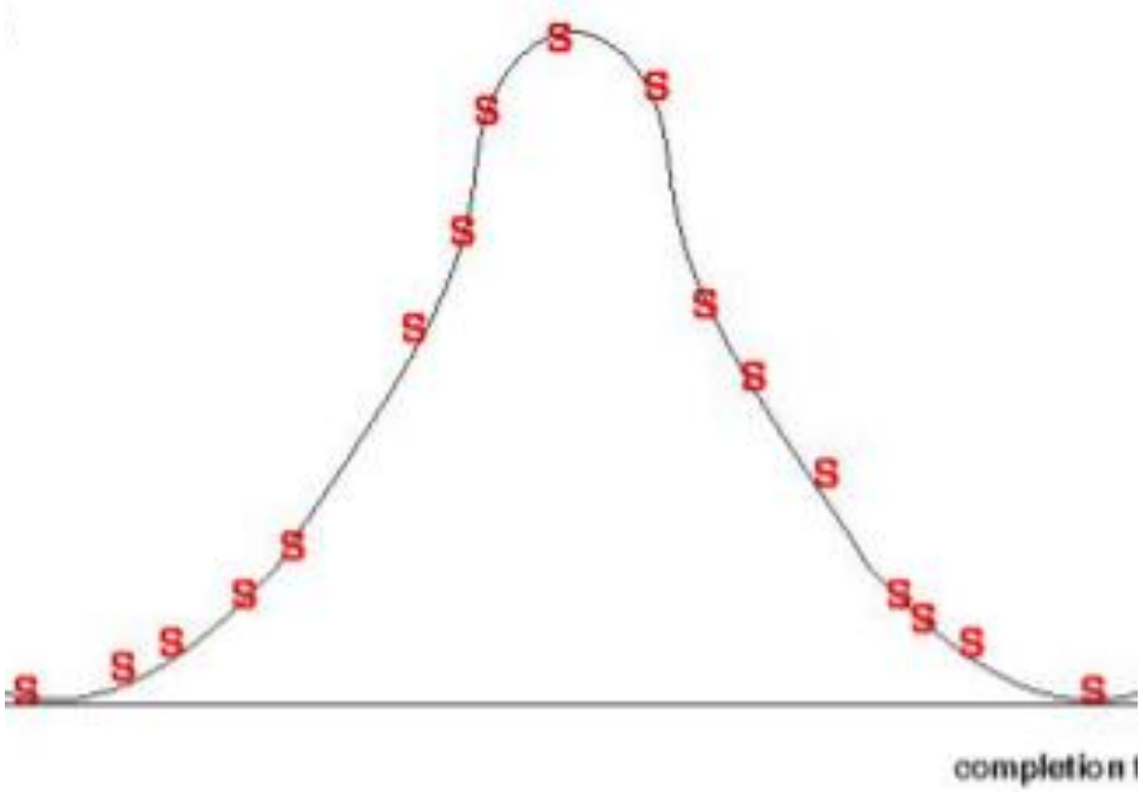
На първия етап BaTS събира статистически данни при вземане на проби със замяна. Тук е необходима само малка извадка (30-50 задачи), за да се изчисли средната стойност и стандартното отклонение на изпълнението на задачите за различни облачни предложения. След това модулът за оценка на бюджета извършва линейна регресия, за да оптимизира тази фаза.



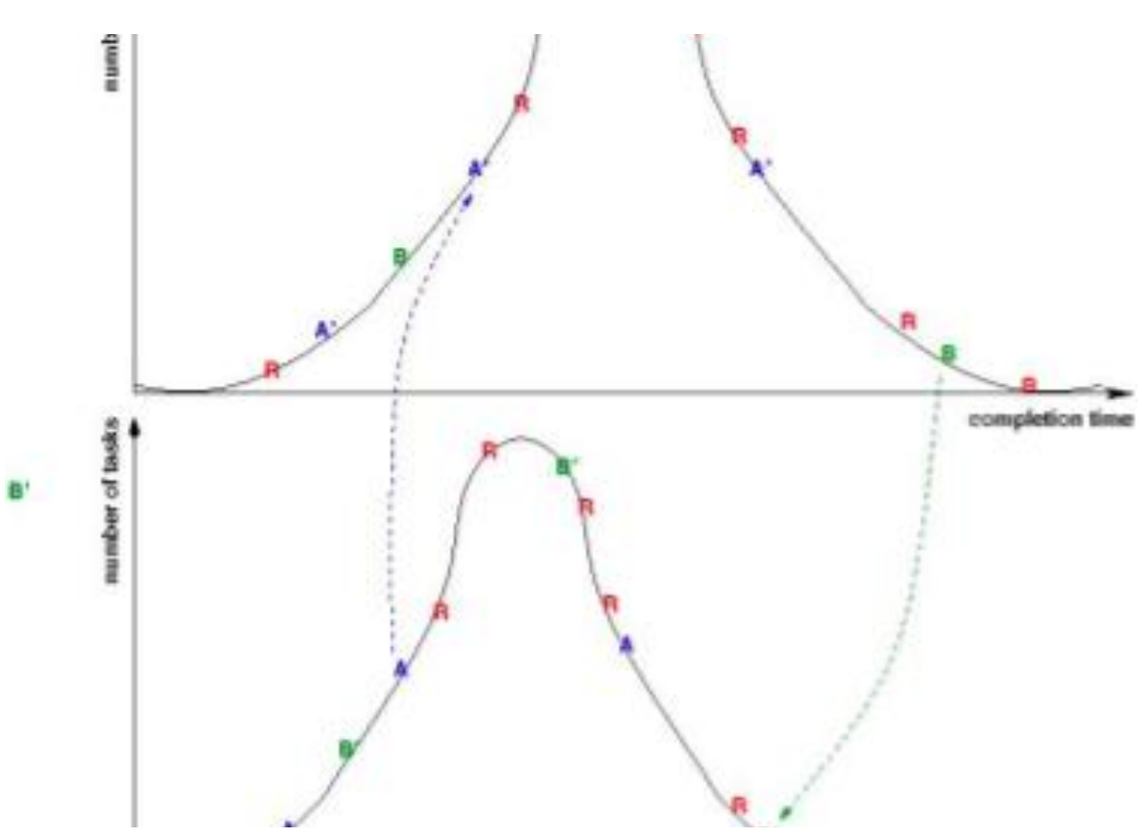
Използване на методологията на BATS за AWS ресурси

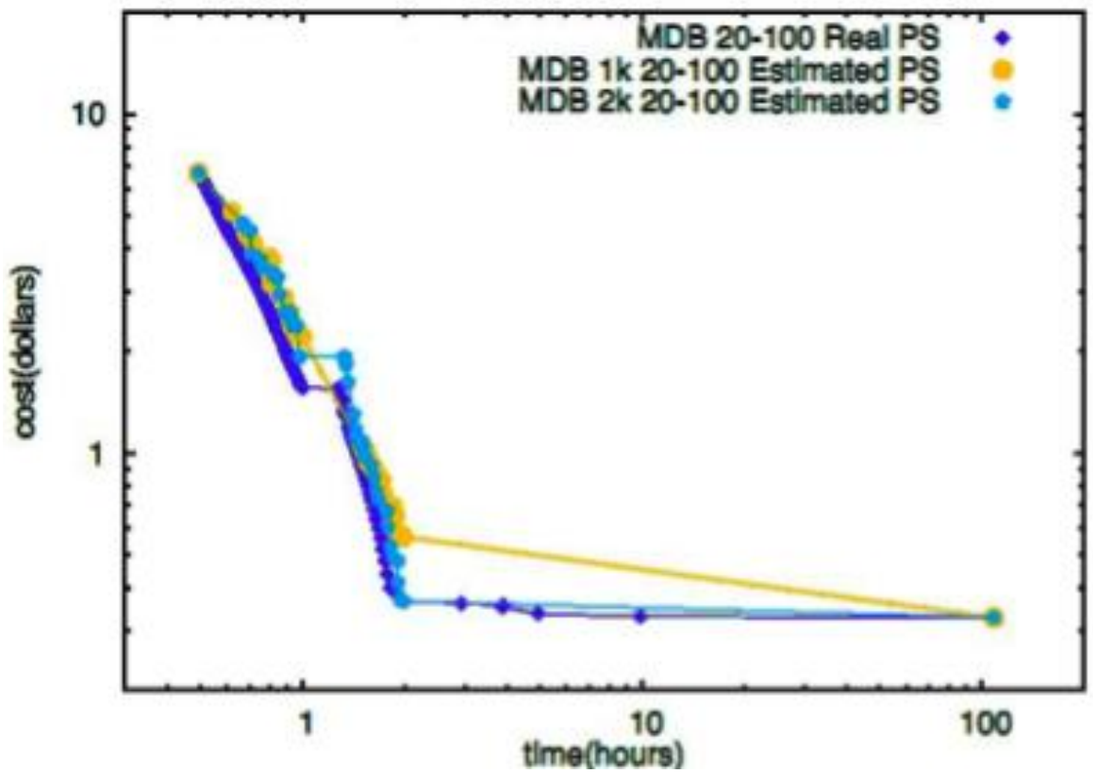
Въвеждаме проблема с подкрепата на собствениците на приложения, които искат да изберат най-добрия избор по отношение на виртуализирани ресурси при разполагане на приложението им на AWS ресурси.

Първоначалната стъпка е да се използва средното време за изпълнение за всеки виртуализиран тип ресурс за изчисляване на бюджета и очакваните прогнози. На редовни интервали от време текущата конфигурация се преоценява, за да се провери дали избраният график е все още осъществим. Ако се очаква нарушение на бюджета, се придобиват по-печеливши машини (по-добро съотношение цена / производителност). Ако се очаква нарушение на очакванията, се придобиват по-бързи машини.



В последната фаза на изчислението трябва да се обърне внимание на основното предположение, че времето за изчисляване е относително. На този етап приложението съдържа твърде малко задачи, за да се задържи предположението. BaTS следи прогнозните неизползвани окончателни фракции отчетни времеви единици. BaTS предоставя възможност за използване от външни лица, които се намират извън крайната отчетна времева единица и добавя виртуализирани ресурси и / или време към графика. Обаче Външните лица, обаче, могат да доведат до нарушения.





Заключение и бъдеща работа

Стохастичните подходи за облачно планиране, ориентирано към потребителя, са обещаващи. Много е важно включването на изследвания в образованието веднага щом технологията достигне стабилно състояние.

Като бъдеща работа бихме искали да подкрепим Haskell AWS Lambda функции, внедрени чрез реализацията на Haskell AWS API.

Източници

[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.

Фокусиране на образованието върху енергийната ефективност

Измервания при тестване на софтуер

Мисията на учителите по софтуерно инженерство е да подготвят бъдещи софтуерни инженери, които да овладеят всеки проблем в жизнения цикъл на софтуера. Освен уменията, свързани с разбирането на нуждите на заинтересованите страни и работата на съответния софтуер, съществената роля играе и способността за проверка на точността на резултатите.

В тази статия се фокусираме върху последния набор от гореспоменатите умения. Акцентът пада върху тестване, където нивото на автоматизация не е толкова важно. Наблягаме на характера на измерването, като се фокусираме върху софтуерното измерване на консумацията на енергия, което може да бъде направено по време на тестове. Всички тези аспекти са представени от гледна точка на преподавател по софтуерно инженерство, с цел да се представи как да се създаде сесия за лаборатория по софтуерно инженерство, която да се фокусира върху измерването на енергийната ефективност при тестване на софтуер, придружени с коментарите на авторите относно тази разработка.

Описание на проблема

Едно от предизвикателствата пред производителите на батерии е колко дълго може да работи батерията, без да се зарежда непрекъснато. Разбира се, има много други предизвикателства като размера, който значително влияе върху формата на устройството, и другият фактор, който е важна характеристика на батерията е теглото на идентификатора. Батерията се счита за малко по-лека в сравнение с устройството, което се нуждае от нея, за да работи.
Предизвикателството тук е как да се намалят размерите – да станат по-малки и по-леки, както и да се осигури висока ефективност по отношение на времето на работа на мобилното устройство. На върха на това хардуерно предизвикателство съществува и софтуерно предизвикателство, а именно, че самият софтуер трябва да поддържа икономия на енергия. Постигането на това без ограничаване на потребителското изживяване в днешно време се счита за задколисна, но важна цел на всеки софтуер, която е насочена към всякакъв вид преносими устройства. Тъй като консумацията на енергия на всяко мобилно устройство се влияе от много фактори като работещите приложения, основните услуги и потреблението, разработването на софтуер за такива устройства вече е предизвикателство [1]. Може да се каже, че предизвикателството за разработка на софтуер винаги е едно и също, но тук трябва да посочим „мобилността“ като ключова системна собственост. Състоянието на захранване на батерията също определя производителността на системата поради конфигурация на ниво операционна система - добре позната като „energy saving preferences“.

Още по-трудно е, ако някой (в нашия случай учителят) трябва да подготви учениците за подобни предизвикателства. Всички вече известни „най-добри практики“ и „съвети за пестене на енергия“ трябва да бъдат представени в контекст, който е лесно разбираем за студентите. В нашия случай, това може да стане чрез позициониране на концепциите в известна среда като софтуерно тестване и тестова автоматизация [2]. Това е целта на разработката.

Тя съдържа в предстоящите раздели, предложението, оценката и завършва с допълнителни съвети за подобряване.

Предложение за решение

Както беше посочено по-горе, трябва да намерим най-подходящата среда за въвеждане на измерване на потреблението на енергия [3] и практики за оценка на енергийната ефективност. Това може да бъде както първоначална разработка на софтуер, така и еволюция на софтуера, тъй като и двете фази на развитие на общия жизнен цикъл на софтуера предлагат възможности за измерване на продукта, който се разработва [4].

При избора на първоначална разработка на софтуер, предимството е, че всички дейности могат да се съсредоточат върху въпросите, свързани с икономия на енергия, докато изборът на алтернативата за еволюция на софтуера предлага възможност за оценка на подобрението при внедряване на продукта. От друга страна, развитието на софтуера изисква наличието на работещ софтуер, докато първоначалната разработка на софтуер е процесът, който създава продукта, като се започне с първата задача на софтуера.

При съпоставяне на двата възможни подхода в учителски контекст, най-доброто решение би биол да се използват и двете. Един семестър за първоначално разработване на софтуер и втори за еволюцията на съответния софтуер. Обикновено учителят няма два семестъра един след друг със същите студенти, за да представи съдържанието на курса по начина, представен в предишния параграф. Това е причината, поради която трябва да решим какъв вид обучение да използваме, за да въведем избраните практики. От гледна точка на архитектурата на процеса на развитие и поради факта, че първоначалната разработка може да бъде и еволюционна, ние заставаме зад еволюцията на софтуера. Тя включва много дейности от първоначалното развитие (с изключение на ранното събиране и анализ на изискванията) и подчертава важността на тестването и оценката.

Този избор позволява на учителя да:

- насочи обучаемите да погледнат назад в развитието си (минали проекти), за да бъдат критични към себе си.

- насочи обучаемите да оценят своите резултати, като използват кодови показатели и измерване (или оценка) на потреблението на енергия.

- води обучаемитев процеса на интегриране на горните дейности в стандартните процеси за проверка и валидиране на еволюцията на софтуера.

Езикът за програмиране на разработката не е важен, следователно студентът може да избере всеки от предишните си проекти - или всички тях, ако се конкурират в броя на използваните разработени проекти или езици за програмиране. Но програмният език обикновено определя или ограничава средата за разработка и използваните инструменти. Изборът на тези инструменти и техните plugins също предлага добра поддръжка за оценка на кодови показатели.

Измерването на енергопотреблението и оценката на енергийната ефективност обикновено изискват различен инструмент, тъй като до този момент съществуват само малко среди за развитие, които да извършват измерване или оценка на потреблението на енергия.

По отношение на тестването като основа за измерване, трябва да отбележим, че анализът на статичния код се използва като част от тестването на софтуера. Освен това, по време на white box testing (тестване главно на единица) се изпълняват избрани части от кодовата база на приложението, които с малко разширение на измерването на консумацията на енергия могат да представят консумацията на енергия в тестовия случай - индиректен поглед върху консумацията на енергия на тестван код. По време на black box testing, цялото приложение се тества с помощта на тестови сценарии. Тези тестови сценарии са основно сравними в случаите на целодневна употреба на софтуера, а другите представляват извънредни сценарии - когато потребителят е в лошо настроение. Измерването на консумацията на енергия при изпълнението на black box tests дава приблизително (но директно) изчисление на консумацията на енергия на дадения продукт. Това е основната полза от еволюцията на софтуера (в сравнение с първоначалната разработка на софтуер). Учителят може да подготви стартовата версия за еволюцията, включително код, който може да бъде

подобрен, списък на известни бъгове и тестова база! Съществуването на тест за повторно измерване и регресионно изчисление е много важно, тъй като продуктивността на еволюцията може да бъде увеличена с тази характеристика.

Ако приемем, че са направени всички горепосочени стъпки, от гледна точка на обучаемите, интегрирането на енергийната ефективност в процеса на тестване изглежда по следния начин:

1. Изберете продукта, който би могъл да бъде ваш предишен проект или изберете такъв от от хранилище.

2. Оценете го с помощта на статичен анализ на кода, тестване, измерване на енергия, проучване на използваемостта и др.

3. Подобрете го (различни видове еволюция като добавяне / промяна на функционалността, поправяне или адаптиране)

4. Проверете повторно, за да сте сигурни, че сте елиминирали всички дефекти или повреди

5. Регресионен тест (включително повторна оценка)

6. Обобщете на резултатите (направете окончателна оценка на всички налични данни, включително енергийна ефективност)

Дискусия

Тъй като измерването на консумацията на енергия е сравнително ново, съпоставено с други техники като статичен анализ на кода, black / white box testing и отстраняване на грешки, може да е проблемът. Но ако се комбинира с тези по-стари принципи, изграждайки комплексен резултат за всеки обучаем, вероятността за критични ситуации е много по-ниска. Разглеждайки възможностите за степенуване, можем да намерим различни „нива на свобода“, които могат да се използват отделно или като част от съставна степен:

1. брой различни проекти,
2. брой приложени / използвани езици за програмиране,
3. качество на кода на крайния продукт,
4. енергийна ефективност на крайния продукт,

5. подобряване на качеството на кода по време на развитието на софтуера,

6. подобряване на енергийната ефективност по време на развитието на софтуера.

Като се имат предвид всички „нива на свобода“ на изпълнение на задачите, много състезания могат да бъдат организирани за мотивираните обучаеми, като целеустремените постигнат „малки победи“ в редица проекти, а перфекционистите ще имат възможноста да оптимизират всички кодови показатели и да сведе до минимум енергията потребление. За средния ученик подобряването на енергийната ефективност и разбираемостта на кода може да бъде постижимата цел.

Конкуренцията на студентите може да бъде мотивирана, като не ги насочите да се върнат към предишните си проекти, но им позволите да избират от посочено хранилище. Както при компютърните игри, всички нива на играта са еднакво достъпни за всички. В подкрепа на справедливостта също може да се създаде общ обществен форум на студенти и учители.

Нашата бъдеща работа в тази област ще се съсредоточи върху конфигурирането на преносима интегрирана среда за разработка, тестване и оценка на потреблението на енергия. Тази среда ще бъде използвана в рамките на еволюцията на софтуера или в началните теми за развитие, за да се подпомогне обучението по измерване на енергийната ефективност по време на тестване на софтуера. Това може да ограничи креативността на студентите, като предлага полузатворена пясъчна кутия, тъй като хардуерът играе много важна роля в съвременната архитектура на оценка на енергийно потребление. Някои изследвания имат за цел да прескочат това ограничение - очакваме тези резултати да ги интегрират.

Източници

[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.

Към инженерна дисциплина за Зелен софтуер?

Този технически доклад описва изследванията, разработени в лабораторията за зелен софтуер в университетите в Коимбра и Миньо, които бяха представени на първата среща за обучение на учители по проект Еразъм + „Фокусиране на образованието върху комбинираността, разбираемостта и коректността на работния софтуер“. Той представя като зелена класация за програмни езици и структури от данни, така и техники за локализиране на ненормално използване на енергия в софтуерните системи.

Мотивация

Днес, широкото използване на безжични, но мощни изчислителни устройства като смартфони, лаптопи и др. променя начина, по който производителите на компютри и софтуерните инженери разработват своите продукти. Всъщност казусът време за изпълнение на компютър / софтуер, което беше основната цел през миналия век, вече не е единственият проблем. Консумацията на енергия се превръща във все по-голямо препятствие както за хардуерните, така и за софтуерните системи. В резултат на това изследванията върху зеления софтуер са важна и диманична област за изследване.

Този доклад накратко описва изследванията, които се разработват в зелен софтуер в Зелената софтуерна лаборатория (GSL). GSL се състои от различни португалски изследователски групи, включително два сайта на проекта „Фокусиране на образованието върху комбинируемостта, разбираемостта и коректността на работещия софтуер“.

GSL е инициатива за разработване на техники и инструменти, насочени към намаляване на потреблението на енергия в различни изчислителни системи (мобилни, програми, бази данни и др.). GSL се фокусира изключително върху софтуерната страна, където прилага (изходен код) анализ и техники за трансформация за откриване на аномалии в консумацията на енергия и за определяне на оптимизации за намаляване на такова потребление.

През миналия век ефективността на софтуерна система главно се свързва с времето за изпълнение и ефективността на потреблението на памет. В наши дни разработчиците на софтуер често задават въпроса „по-бърза и по-екологична програма ли е?“. Има много аспекти на софтуерната система, които влияят върху нейните енергийни характеристики: езикът на програмиране и нейният модел на изпълнение (компилиран в двоичен код или към виртуална машина, интерпретиран код, обща срещу строга оценка, използване на частична оценка на изпълнение и т.н.). Ефективността на модела и езика на библиотеките също влияят на производителността. Сложността на алгоритъма, използван за изпълнение на желания компютърен проблем, също влияе върху производителността: ако внедреният алгоритъм трябва да свърши повече работа от това, което е строго необходимо, тогава ще се използва повече процесорна мощ и енергия.

В този документ накратко отчитаме резултатите от изследванията, постигнати в GSL, а именно при анализ на енергийната ефективност на програмните езици (Раздел 2), библиотеките на структурата на данните (Раздел 3) и на изходния код на софтуера (Раздел 4).

Зеленина в програмните езици

Интересен въпрос, който възниква при обсъждане на енергия при програмните езици е дали по-бързият език е и енергийно ефективен език, или не. Сравняването на софтуерни езици, обаче, е изключително сложна задача, тъй като работата на един език се влияе от качеството на неговия компилатор, виртуална машина, контейнер за ненужна информация и т.н.

В лабораторията за зелен софтуер проучихме, оценихме и сравнихме работата на (общо) 27 от най-широко използваните софтуерни езици. Използвахме две различни компютърни хранилища: Computer Language Benchmark Game (CLBG) 3 и хранилищата на Rosetta Code4 [1–3]. И двете хранилища определят набор от компютърни задачи и предоставят реализации в голяма група езици за програмиране. Докато CLBG е пригоден да анализира изпълнението на езиците по време на изпълнение, кодът Rosetta е определен с повече цели за разбиране на програмата.

Написахме такива програми, използвайки най-съвременните компилатори, виртуални машини, интерпретатори и библиотеки за всеки език. След това наблюдавахме времето за изпълнение, пиковата и общата консумация на паметта и потреблението на енергия на процесора / - DRAM / GPU. Съставихме енергийна класация на 27-те езика и анализирахме тези резултати според типа на изпълнение на езиците (компилиран, виртуална машина и интерпретиран) и използваната парадигма за програмиране (императивно, функционално, обектно-ориентирано, скриптиране). За типовете изпълнение и парадигмите за програмиране, съставихме софтуерно класиране на езика според всяка отделно разгледана цел (например потребление на време или енергия). Първите ни експерименти показват очакваните резултати - езикът C е и по-бързият, и по-зеленият език, но по-бавни езици се оказват по-енергийно ефективни от другите [2, 3].

Зеленина в структурите на данните

Парадигмата относно езикът за програмиране и нейните мощни оптимизации на компилатора не са единственият аспект, който влияе върху консумацията на енергия в софтуерната система. Всъщност една програма може също да стане по-ефективна като „само" оптимизира своите библиотеки [4, 5]. Повечето езици предлагат мощни библиотеки за манипулиране на структури от данни. В GSL изследваме енергийните характеристики на две модерни структури от данни, широко използвани в езиците за програмиране на Java и Haskell.

В Java проведохме подробно проучване по отношение на консумацията на енергия в библиотеката 5 - Java Collections Framework (JCF). Разгледахме обичайните три групи структури от данни, а именно комплекти, списъци и карти. За всяка от тези групи, изследвахме консумацията на енергия на всяко от нейните различни приложения и методи на работа [4]. Тази енергийна осведоменост на JCF може да се използва не само за насочване на софтуерните инженери за писане на по-екологичен Java софтуер, но и за оптимизиране на наследения Java код. Разработихме инструмент за refactoring на структурата на Java, наречен jStanley, който рефакторира изходния код на Java, когато е налична по-зелена колекция [6]. Също така извършихме първоначална оценка със седем обществено достъпни Java проекта, при които успяхме да подобрим консумацията на енергия с между 2% и 17%.

В Haskell изследвахме консумацията на енергия на Edison6, напълно зряла и добре документирана библиотека с чисто функционални структури от данни [7]. Edison предоставя различни функционални структури от данни за прилагане на три вида абстракции:

Последователности (списъци, опашки и стекове), колекции (набори и купища) и асоциативни колекции (карти и ограничени отношения). Анализирахме 16 разработки на такива структури от данни, като измервахме подробни показатели за енергия и време [5]. Освен това проучихме въздействието на потреблението на енергия, използвайки различни оптимизации за компилация.

Стигнахме до заключението, че потреблението на енергия е пряко пропорционално на времето за изпълнение и че потреблението на енергия на DRAM представлява между 15% и 31% от общото потребление на енергия. Накрая заключихме също, че оптимизациите могат да имат както положително, така и отрицателно въздействие върху консумацията на енергия.

Зеленина в изходния код

Не само езиците и библиотеките на структурата на данните влияят върху консумацията на

енергия, алгоритмите и практиките на програмиране също играят ключова роля за ефективността на програмите. В GSL сме приспособили добре познатите техники за локализация на неизправности, за да локализират статично „изтичане на енергия“ (разглеждано като енергийна неефективност, следователно, енергийни неизправности) в изходния код на приложенията [8–11]. Дефинирахме SPELL - базирана на SPectrum локализация на изтичане на енергия, за да определим червените (енергийно неефективни) области в софтуера.

Първото експериментално проучване показва, че експертните програмисти с достъп до изтичане на енергия от SPELL успяха да оптимизират по-добре енергийното потребление на програмите (между 15% и 74%), отколкото експерти без информация или информация, предоставена от стандартен програмен (runtime) profiler. Изследвали сме и енергийното потребление на програмите C / C ++ [12].

Широкото използване на безжични устройства и появата на интернет-файлове променя начина, по който софтуерните инженери разработват своя продукт. Софтуерът трябва да работи на различни мобилни устройства и консумацията на енергия да е основна грижа при разработването на програма. Software Product Lines (SPL) се превърнаха във важна дисциплина от софтуерното инженерство, позволяваща разработването на софтуер, който споделя набор от функции. В GSL сме дефинирали техники за статичен анализ, за да разсъждаваме за потреблението на енергия в SPL на базата на условна компилация. Подобни техники позволяват на инженерите да идентифицират (не)зелени продукти и / или функции в SPL [13].

Android е широко използвана екосистема за безжични устройства, а анализът и оптимизацията на софтуерната енергия са активна област на изследване. Екипът на GSL е разработил няколко техники [14, 15] и инструменти за анализ и оптимизиране на консумацията на енергия в изходния код на приложенията за Android [16, 17].

В днешно време повечето от данните, съхранявани в нашите мобилни устройства (файлове, снимки, видеоклипове), също се съхраняват в облака, предоставен от екосистемата на операционната система на устройството. Такива облачни системи са центрове за данни, които ежедневно изпълняват голямо количество процеси за запитване на данни, контролирани от високо усъвършенствани системи за

управление на бази данни, които отговарят за създаването на ефективни планове за обработка на заявки. Системите от бази данни обикновено разчитат на планове, които оптимизират времето за отговор. Ние моделирахме и разработихме алтернативен метод за определяне на планове за потребление на енергия за заявки в база данни [18, 19]. Първите ни експериментални резултати показват, че използването на евристика за оптимизация позволява значителни подобрения, както по отношение на консумацията на енергия, така и времето за изпълнение.

Заключения

Този технически доклад описва изследванията, разработени в Зелената софтуерна лаборатория, а именно зелена класация на програмните езици и структурите на данните, техники за откриване на енергийна неефективност в изходния код на софтуерната система и план за изпълнение на заявки за системи с бази данни, съобразен с енергията.

Източници

[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

Преподаване на Програмиране ориентирано към задачи (TOP)

По време на зимната школа за Composability, Comprehensibility, Correctness ще има две лекции за програмиране, ориентирано към задачи и конкретни системи, базирани на тази парадигма. Системата iTask предлага уеб базиран интерфейс за потребителите, за да виждат задачите си и да споделят напредъка си в тези задачи. Системата mTask прилага същите концепции за уточняване на задачите, изпълнявани от микропроцесорите. В този принос ние оправдаваме решенията, взети в обучението на учители в Амстердам за това какво и как ще бъдат представени тези теми в зимното училище. Поради ограниченото време и разнообразието от публика ще се съсредоточим върху практическото използване на парадигмата. Времето за справяне с предизвикателствата и красотата на изграждането на тези системи е недостъчно.

Въведение

Програмно ориентирано програмиране, TOP, е стил на програмиране, концентриран около концепцията за задачи, изпълнявани от хора и машини. Тези задачи се задават от обикновените функции на функционален език за програмиране. Във всички наши примери ще използваме Clean [6]. Семантиката на задачите е доста различна от обикновената оценка на функциите. Задачата се оценява отново и отново, докато не даде стабилна стойност или резултатът от нея не е необходим вече. Междинните резултати от задачите могат да се наблюдават и от други задачи. Задачите могат да бъдат съставени от комбинатори на задачи.

По време на зимното училище за Composability, Comprehensibility, Correctness в Кошице ще има две сесии на TOP.

Те са озаглавени „Защо TOP има значение“ и „Функционално програмиране на устройства“. И двете сесии ще се състоят от лекция и практическа работа с аудиторията. В този материал обсъждаме решенията относно съдържанието и организацията на тези сесии след дискусиите в обучението на учителите.

Аудитория

Зимното училище по Composability, Comprehensibility, Correctness е за бакалаври, магистри, докторанти, както и за преподаватели. Дискусиите по време на обучението разкриха, че опитът във функционалното програмиране е многообразен, както в големината на опита, така и в езиково отношение. Използваните езици варират от чисти и лесни езици като Haskell и Clean до Erlang, Scheme и Scala.

Това означава, че не можем да приемем солиден общ функционален опит в програмирането. Само част от аудиторията ще бъде запозната с понятия като силен тип, функции от по-висок ред, класове тип конструктор, Monads и generics. Въпреки че тези теми са градивните елементи на TOP, не можем да предположим, че са известни на всички участници.

Програмно ориентирано програмиране

Програмно ориентираното програмиране се основава на малък брой примитивни задачи, специфични за домейна. Тези примитивни задачи обикновено взаимодействат със средата, например хора, изпълняващи част от задачите, или хардуер, взаимодействащ с физическия свят. Комбинаторите на задачи се използват за съставяне на задача от по-малки задачи. Задачите могат да комуникират чрез своите резултати и чрез общи източници на данни, SDSs. Такъв SDS съдържа въведени данни, които могат да бъдат достъпни чрез основни команди като get и set. Тези основни команди повлияват на задачата, за да осигурят референтна прозрачност.

TOP система често се изгражда като DSL, вграден в съществуващ (функционален) език за програмиране, за да се използват повторно типовете данни и изчисления на съществуващ език.

Системата iTask

Системата iTask беше първата реализация на TOP [5]. Това е DSL, вграден във функционалния език за програмиране Clean. Системата iTask улеснява взаимодействието с потребителя от генерираното от типа поколение уеб страници. Тези страници се показват в един от съществуващите браузъри. Страницата предоставя информация за текущите задачи за даден потребител. Потребителят може да взаимодейства със системата iTask чрез попълване на формуляри и натискане на бутони.

Използвани техники

ITask системата преминава състояние много подобно на състояние monad. Операторите за връщане, свързване (>> =) и последователност (>> |) са много подобни на добре познатите monad версии [4, 7]. Това изисква функции от по-висок ред и дефинирани от потребителя инфиксиращи оператори. За да използват повторно символа на оператора за различни monads, те се дефинират като типове конструкторски класове.

Комбинаторите на задачи са всички функции за по-висок ред, обикновено определени от потребителя инфикс оператори, манипулиране на резултатите от задачите и глобалното състояние на задачата. Специфичното за TOP е, че задачите дават междинни резултати, които могат да се наблюдават, докато задачите се повтарят отново и отново, и не дадат стабилен резултат или резултатът им вече не се използва. Това изисква функции от по-висок ред, леснота и автоматично събиране на ненужната информация.

Системата iTask генерира уеб сървър, който се използва от потребителите, за да намерят своята задача. Както всички уеб сървъри, това изисква сериализация и десериализация на състоянието, за да се съхранява и извлича текущото състояние. Чрез попълване на уеб-формуляри за произволни алгебрични типове данни потребителите посочват напредъка си със задачите. Всички тези функции се реализират с общо програмиране.

За ефективно изпълнение на задачите за наблюдение на стойността на SDS съществува скрита система за публикуване и абониране за всеки SDS, която активира задачи, използващи този SDS, когато стойността му е актуализирана.

Освен тези свойства, системата iTask използва много допълнителни техники. Например, изпълнението на части от задачата в интерпретатор, работещ в thebrowser, за да се осигури бърз отговор на системата за силно интерактивни задачи като рисуване.

Преподаване в зимното училище

Преподаване на програмиране изисква практически опит в програмирането и използване на образованите техники за овладяването им. Това важи и за TOP. В резултат на това ние разделяме четирите часа, определени затемата „Защо TOP“ на две части с почти еднаква дължина.

В първата част ще очертаем концепцията на TOP, използвайки системата iTask. Предвид знанията на по-голямата част от обучаемите, трябва да пропуснем почти всички подробности относно внедряването на системата и трябва да се съсредоточим върху използването на библиотеката. Тази библиотека всъщност е вграден DSL за TOP. В лекцията използваме този набор от primitives, без да отделяме много време, за да обясним неговата архитектура и изпълнение.

За практическата работа ще разделим съществуващия основен примерен проект в набор от малки независими TOP проекти. Задачите ще се състоят от малки вариации на тези проекти, за да усетят какво представлява TOP. По-опитните функционални програмисти могат да прескачат повечето от основните упражнения и да скочат директно към по-разширеното задание. По този начин ще можем да се адаптираме към индивидуалните знания и умения на всеки участник.

Системата MTASK

Микропроцесорите са компютърни системи с много ограничени изчислителни възможности. Те обикновено имат доста ниска тактова честота и сериозни ограничения на паметта, като няколко KB памет за съхранение на данните на работеща програма. Тези евтини процесори са движещата сила на много елементи в Интернет на нещата, IoT. В такива микропроцесорни системи обикновено трябва да се наблюдават няколко входни порта, както и да се контролират някои изходи на база на направените наблюдения. Поради хардуерните ограничения обикновено няма операционна система, предлагаща поддръжка.

Парадигмата TOP осигурява леки на тежест нишки, които са много подходящи за наблюдение и координиране на напредъка на такива добре дефинирани прости задачи. Тези задачи могат да се изпълняват със собствена скорост, докато за тяхното координиране се използват комбинатори и споделени източници на данни. Стартирането на системата iTask на IoT устройства би ни позволило да конструираме програми, които се изпълняват частично на уеб сървър, както и на IoT устройства. Ограниченията на микропроцесорите правят невъзможно стартирането на пълноценна програма iTask на IoT устройства.

За да се доближим идеалното решение, разработихме системата mTask [3, 2]. Това е многоизгледен DSL, плитко вграден, който може да се използва като част от iTask системата. Той поддържа TOP парадигмата, включваща същите резултати от задачите като iTask системата, комбинатори на задачи и споделени източници на данни. По конструкция този DSL няма функции от по-висок ред и няма рекурсивни типове данни. Поради наложените ограничения в този DSL, mTask програми могат да се компилират в код, който се изпълнява на микропроцесори. Въпреки тези ограничения, DSL е много подходящ да определи задачите, които трябва да се изпълняват на IoT устройства лесно и много бързо.

Използвани техники

Системата mTask е многоизгледен DSL, плитко вграден, изграден на базата на типове конструкторски класове [1]. Всеки екземпляр от тези класове определя интерпретация, наречена изглед, на програма, изградена от тези primitives. Типичните изгледи реализират доста печат, генерират на код за микропроцесори и симулация на mTask програмите като iTask програма. DSL може да се разширява чрез конструкция, за да се използват повторно съществуващите библиотеки за периферни устройства като температурни сензори, дисплеи и servo motors. Лесно е да добавите такава библиотека като примитивен език към системата mTask чрез въвеждане на нов тип конструкторски клас и необходимите екземпляри.

За да направи внедряването на mTask преносим на много различни микропроцесори и да направи лесно повторното използване на съществуващи C ++ библиотеки, изгледът за генериране на код създава C + + код за платформата Arduino, вместо основния машинен код за някакъв специфичен процесор. Компилаторът avr-gcc вътре в платформата Arduino може да преведе генерирания C ++ код и библиотеките, използвани за основния код за много различни микропроцесори.

Преподаване в зимното училище

Да можеш да разработваш TOP програми на високо ниво на миниатюрен микропроцесор, с периферни устройства, е привлекателно за урок в зимното училище. Изпълнението на mTask програма на действителен микропроцесор, обаче, изисква моного от учениците; те трябва да съставят програма mTask вътре в системата iTask, да изпълнят програмата iTask, за да получат C ++ код, да подадат този C ++ код към ID на Arduino, да свържат ID на Arduino към микропроцесора и да изберат правилните опции, след което да качат

програмата в микропроцесора и накрая да я стартират. Всички тези стъпки са сравнително лесни, но целият процес дава резултат само когато всяка стъпка се изпълни правилно. Тъй като генерираната програма ще работи на микропроцесор без операционна система и много ограничени входни и изходни периферни устройства, отстраняване на грешки такава програма е предизвикателство. След широко обсъждане на последователността от стъпки беше бе очертано че времето за лекцията е много ограничен. За щастие, изгледът на симулатора на системата mTask предлага алтернатива, която е много по-лесна за използване. Този изглед трансформира програмата mTask в обикновена iTask програма. Симулаторът предлага поетапно изпълнение на програмата mTask. Той показва следа от последната стъпка на последната изпълнена задача и състоянието на всички периферни устройства и споделени източници на данни. Часовникът, стойността на SDSs, както и състоянието на периферните устройства могат да се променят интерактивно, за да се контролира изпълнението и да се изследват различни сценарии. Това прави практическата работа на тази лекция директно свързана с практическата работа на предишния урок за iTask.

Беше решено да се планират тези уроци за един ден с лекцията iTask и свързаната с нея практическа работа сутрин и урока mTask следобед. По този начин сесията mTask може директно да надгражда знанията и уменията, придобити в iTask сесията. Разбирането на TOP, дискутирано сутринта, ще се задълбочи следобед.

Заключение

И за двата урока за TOP засягат още много по-интересни теми, които за съжаление не могат да бъдат обхванати в определеното време за темата в зимното училище. В сесиите ще се съсредоточим върху разбирането и използването на TOP парадигмата чрез относително прости примери. Ще се използват леко усъвършенствани примери за илюстриране на възможностите на този подход. В практическата работа ще се съсредоточим върху илюстративните упражнения, които са предимно варианти на примери, използвани в урока. За напредналите обучаеми ще има някои предизвикателни задачи, както и възможност да се обсъдят подробно аспектите на системите.

Източници

[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

Интерактивен подход към COLOURED PETRI NETS TEACHING

Официалните методи принадлежат на техниките, които при подходяща употреба могат значително да допринесат за точността на софтуерната или хардуерната система в процеса на разработка. Един от подходящите методи за системи с едновременна или недетерминирана форма на работа е Coloured Petri Nets. В тази статия е описана учебната дейност, насочена към обяснение на основните принципи на езика и някои от неговите програмни функционалности. Продължителността на обучението беше два часа и половина и включва интерактивна изграждане на модел с активно участие на аудиторията.

Въведение

Като се има предвид нарастващата зависимост на съвременното общество от компютърните системи, тяхната точност трябва придобива изключително значение. Един от подходите, които могат значително да допринесат за това, е използването на формални методи по време на разработката на софтуер и хардуер. Официалният метод е математически базирана техника, която осигурява формален език с недвусмислено дефиниран синтаксис и семантика, както и апарат, който позволява извършване на задачи за проверка, разработка и симулация на система с езикови спецификации. Един от важните членове в групата на формалните методи е Coloured Petri Nets (CPN). CPN [4, 3] комбинира формализма на Coloured Petri Nets [1] с функционален език за обработка на данни и процедури за вземане на решения. Функционалният език се нарича CPN ML и представлява леко модифицирана версия на Standard ML [2, 5]. Езикът на CPN и съответните задачи за спецификация, проверка и симулация се поддържат от софтуера CPN Tools [6].

Повече от десетилетие CPN са част от бакалавърски курсове, свързани с формални методи, моделиране и симулация. Един от методите, прилагани от автора при обяснение на понятията CPN, е интерактивен подход с активно участие на аудиторията. Тук аудиторията избира домейна и процеса, за който ще бъде проектиран CPN модел и помага за създаването на избраните от него части. Опитът от конкретно прилагане на този подход в обучителна дейност за университетските преподаватели е описан в останалата част от тази разработка.

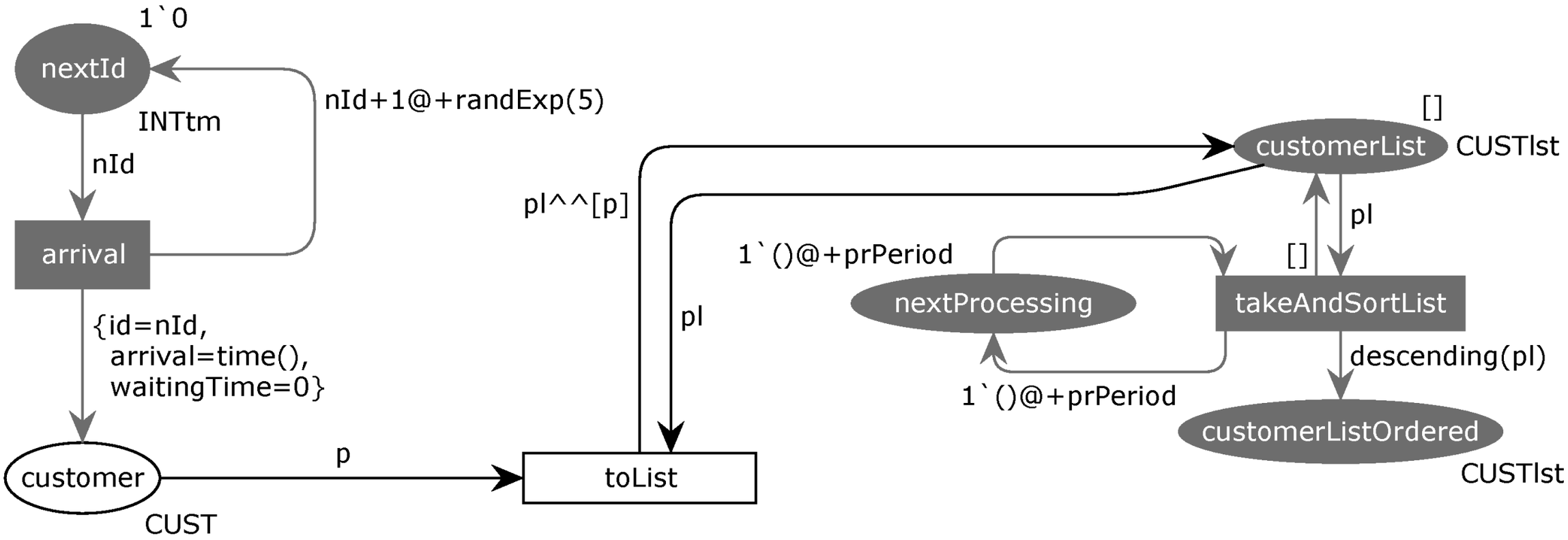
Обучителна дейност с интерактивен CPN модел

Обучителната дейност беше организирана за около 10 участници, които са университетски преподаватели с определен опит във функционални езици. Участниците нямаха предходни познания за CPN. Общата продължителност на дейността беше около 2,5 часа, без почивките и тя беше разделена на три фази.

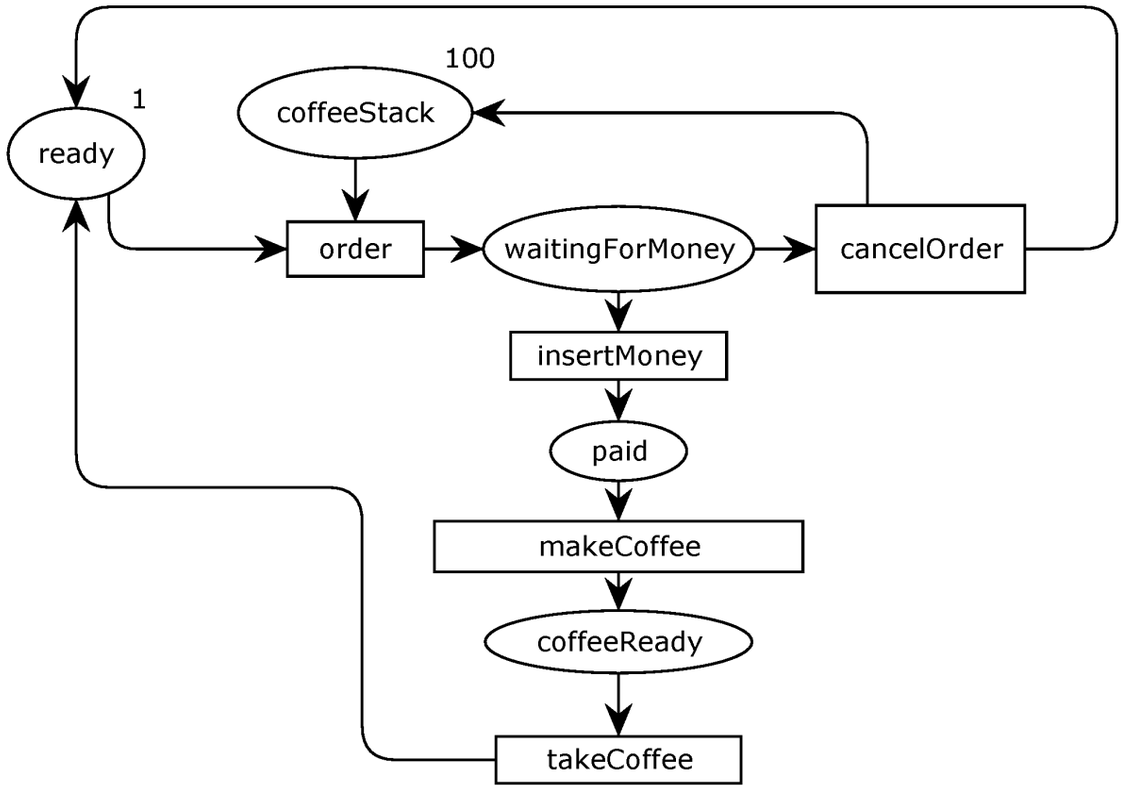
Първата фаза е около 30 минути и обяснява основните принципи на CPN. А именно, че CPN има графична форма, двустранна графика с два вида върхове: места, начертани като елипси и преходи, нарисувани като правоъгълници. Местата съдържат tokens (жетони), които представляват състояние на мрежата и преходите могат да бъдат разбрани като събития, които променят състоянието като поемат съществуващи tokens и създават нови.

Втората и третата фаза са посветени на създаването на CPN модел. Тъй като една от целите на заниманието беше да покаже как някои по-модерни концепции на стандартния ML, а именно structures и functions, могат да се използват в CPN модели, на участниците беше даден стартов CPN модел, който вече използва концепциите, преди започването на втората фаза. Моделът на стартера е показан на фиг. 1.

Частта, състояща се от възлите nextId, arrival и customer, представлява редица от клиенти, които пристигат един по един, за да бъдат обслужвани. Самото обслужване не е представено в стартерния модел.



Вместо това има преход към списъка, който поучава сигнал от клиента и добавя стойността му към друг списък, който се съхранява на мястото customerList. Преходът takeAndSortList, дефиниран от стойността prPeriod, се задейства на равни интервали. Всяко пускане на takeAndSortList изпразва списъка в customerList, сортира съдържанието му и съхранява исканата версия в customerListOrdered. Мястото nextProcessing е гарантира, че takeAndSortList ще се задейства само на редовни интервали. Сортирането се осигурява от функция, наречена низходяща, която реализира Quicksort алгоритъма.

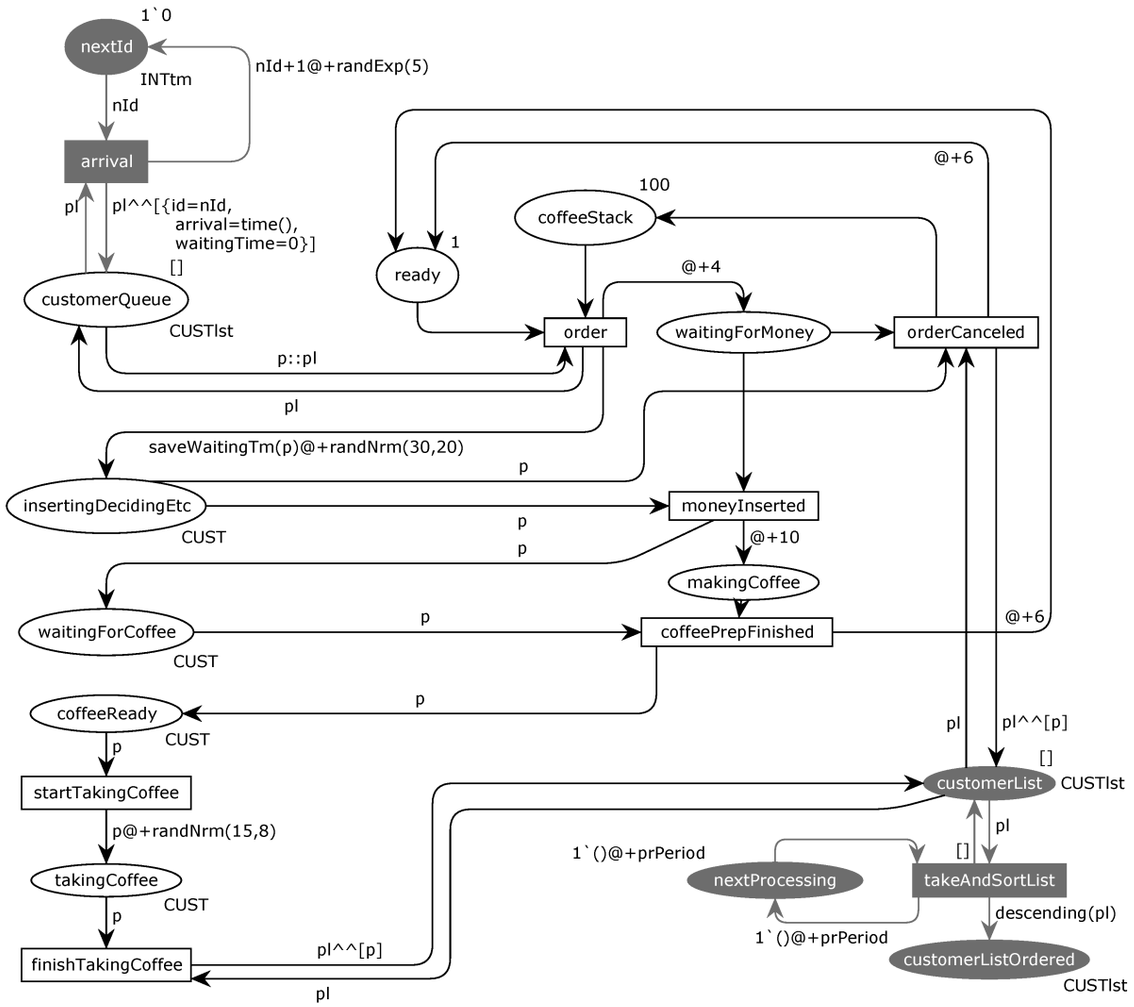


Функцията използва стандартни ML structures и functions.

За обслужващата част, участниците решиха да моделират машина за кафе. По време на втората фаза те участваха в създаването на CPN Фиг. 2. CPN модел на обслужващата част е създаден интерактивно по време на втория фазов модел, който кординира основната работа на машината. Моделът е показан на фиг. 2. В първоначалното си състояние машината е готова да обслужва клиент (един знак на място е готов) и се пълни със 100 дози кафе (100 символа в coffeeStack).

Обслужването започва с клиент, поръчващ кафе чрез изпичане на преходната поръчка. Тогава машината изчаква следващата стъпка на клиента (знак в изчакване за ForMoney). Клиентът може да вмъкне пари (чрез изстрелване на insertMoney) или да анулира поръчката (чрез изстрелване cancelOrder). Отмяната връща машината в състояние на готовност. Ако парите се вмъкнат, машината приготвя кафето (чрез изпичане на MakeCoffee). Накрая, чрез изстрелване на takeCoffee, клиентът взема готовото кафе и машината се връща в състояние „готово“.

След втората фаза имаше около 70 минути почивка. По време на почивката преподавателят свърза модела от фаза 2 към частите на стартерния модел и добави диаграми, описващи поведението на клиента. Той също коригира някои несъответствия в модела, посочени от един от участниците. Полученият, окончателен, CPN модел може да се види на фиг. 3. Върховете, взети от стартерния модел (фиг. 1) без промяна, са представени в сиво. Клиентът се заменя с customerQueue, който съдържа token със списък от стойности, представляващ опашка от клиенти, които чакат за машината. Вместо прехода към списъка има сервизна част, създадена от резултата от втората фаза (фиг. 2).



Обслужващата част на крайния модел се различава от фиг. 2 в три основни аспекта:

- Токените носят информация за обслужвания клиент, а изразите на дъгата определят продължителността на съответните действия.

- Несъответствията по отношение на ролята на местата и преходите се коригират. Сега всички преходи представляват мигновени събития. Например, преходът makeCoffee от фиг. 2 се заменя с мястото, което makeCoffee, а преходът takeCoffee се заменя с върховете startTakingCoffee, takeCoffee и finishTakingCoffee.

- Действията и състоянията на клиентите и машината се моделират отделно.

Вертикалите на customerQueue, insertingDecidingEtc, waitingForCoffee,

Третата фаза на обучителната дейностза около 30 минути, която беше посветена на обяснението на крайния модел и дискусия за мястото на такива модели в развитието на правилни компютърни системи.

Заключение

Представената тук интерактивна тренировъчна задача е подходяща за кратки, интензивни курсове, които често се провеждат по време на летни училища или други подобни учебни събития. Описанието на дейността показа, че първоначално определеното време (2 часа) не е достатъчно. Ето защо е необходима третата фаза, в която лекторът представи крайния модел. Като се има предвид времето, необходимо за изграждането на крайния модел от преподавателя, ще са необходими още поне два часа, за да се осъществи целия процес на създаване на интерактивен модел на обучение. Всички модели CPN, представени или споменати тук, могат да бъдат получени по заявка от автора.

Източници

[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: екстензивна рамка за разбиране

CodeCompass е инструмент с отворен код, който помога за разбирането на големи наследствени софтуерни системи. Въз основа на инфраструктурата за компилатор LLVM / Clang, CodeCompass дава точна информация за сложни езикови елементи C / C ++. Широкият диапазон от интерактивни визуализации включва диаграми за класови и функционални разговори; архитектурни, компонентни и интерфейсни диаграми и много други. За изследване на архитектурата системата, CodeCompass също използва информация за изграждане, както и информация за контрол на версиите, когато е налична. Резултатите от статичния анализ на базата на Кланг също са интегрирани. Въпреки че инструментът се фокусира главно върху C и C ++, той също поддържа Java и Python езици. CodeCompass е уеб-базирана, подвижна, разширяема архитектура, която може да бъде отворена платформа за по-нататъшно разбиране на кода, статичен анализ и усилия за софтуерни метрики.

Въведение

Поправянето на грешки или разработването на нови функции изисква уверено разбиране на всички детайли и последствия от планираните промени. Инструментите за разбиране на кода могат да помогнат за разкриване на първоначалните намерения и подробности за внедряване, като се изгради модел от изходния код на база на друга налична информация. Въпреки че редица такива инструменти се предлагат или като патентован, или безплатен софтуер, наборът от функции е ограничен.

CodeCompass е разработен за премахване на тези ограничения. Проектът CodeCompass, с отворен код, е съвместно усилие на Ericsson Ltd. и университета E¨otv¨os Lor´and, Будапеща. Целта е да помогне за разбирането на големи софтуерни системи. CodeCompass се основава на истински компилатор, инфраструктурата LLVM / Clang, за да предостави точна информация за сложни C / C ++ езикови елементи като претоварване, наследяване, използване на променливи и типове, възможни приложения на функционални указатели и виртуални функции - функции, които различните съществуващи инструменти поддържат само частично. По този начин програмата елиминира слабостите на обичайните инструменти за разбиране на „леко тегло“, като OpenGrok.

CodeCompass не е, обаче, ограничен до изходния код. Той използва информацията за изграждане на системата, за да разкрие архитектурни връзки. Освен това той използва информацията за контрол на версиите, ако е налична, така че човек може да идентифицира връзките между различни изходни файлове, „случайно“, променени в един и същи запис. За да помогне за бързото и прецизно възприемане, CodeCompass използва както текстово, така и графично представяне на софтуерната система. Редица (интерактивни) диаграми са достъпни от обичайните графики за извикване на функции до уникалните архитектурни диаграми. За да осигури лесен достъп за потребителите, CodeCompass има уеб базирана архитектура. Клиентът може да бъде стандартен уеб браузър, плъгин за редактор или всяко приложение на трета страна. Комуникацията се основава на REST API и е добре разпределена за паралелни клиентски заявки.

В тази статия сравняваме CodeCompass със съществуващите инструменти за разбиране и описваме набора от функции, които изппълнява. В раздел 2 разглеждаме основните архетипи на съществуващите инструменти за разбиране на кода. Въвеждаме екстензивната архитектура на CodeCompass в 3. Основните характеристики на инструмента са разгледани в раздел 4. Обобщаваме документа в раздел 5.

Работа по темата

На софтуерния пазар има няколко инструмента, които целят разбиране на изходния код. Някои от тях използват статичен анализ, а други изследват динамичното поведение на анализираната програма. Тези инструменти могат да бъдат разделени на различни архетипи въз основа на техните архитектури и техните основни принципи. От една страна инструментите имат архитектура сървър-клиент. Като цяло тези инструменти анализират проекта и съхраняват цялата необходима информация в база данни. Клиентите (обикновено уеб базирани) се обслужват от базата данни. Тези инструменти могат да бъдат интегрирани в работния процес, докато тече нощна CI. По този начин разработчиците винаги могат да разглеждат и анализират цялата, голяма, наследена база данни с кодове. Също така има и тежки за клиента приложения, където по-малката част от кодовата база е анализирана. Това е случаят на използване за редактори на IDE, където честата модификация на източника изисква бързо актуализиране на базата данни за анализираните резултати. В този раздел представяме някои инструменти, използвани в индустриалната среда от всяка категория. Woboq [3] е уеб базиран браузър за кодове за C и C ++. Този инструмент има широки функции, които имат за цел бързото сърфиране на софтуерен проект. Потребителят може бързо да намери файловете и имената на обекти чрез поле за търсене, което осигурява попълване на код за лесна използваемост. Навигацията в кодовата база е активирана чрез уеб страница, състояща се от статични HTML файлове. Тези файлове се генерират по време на синтактичния анализ. Предимството на този подход е, че ще бъде бърз, тъй като при сърфиране не е необходимо изчисляване в движение от страна на сървъра.

Задържане на мишката върху определена функция, клас, променлива, макро и др. може да покаже свойствата на всеки елемент. Например, в случай на функции може да се види неговият подпис, мястото на неговото определение и използване. За класовете човек може да провери размера на обектите, класовата подредба и изместването на членовете, както и схемата за наследяване. За променливите може да се провери типа и местонахождението, където са налични.

В C и C ++ макросите образуват подезик, който се оценява в етап на предварително компилиране. Тази оценка представлява текстово заместване на макротокени, което означава, че фазата на компилиране работи с друг код, различен от оригиналния. В Woboq може да се провери и крайната стойност на макро разширенията.

Много удобна функция на инструмента е семантичното подчертаване. Чрез тази функция лесно могат да бъдат разграничени различните езикови елементи: форматирането на локални, глобални или членски променливи, виртуални функции, типове, typedefs, класове, макроси и т.н. са различни. Woboq може да предостави горепосочените функции, тъй като необходимата информация се събира в реална фаза на компилиране. Разгледаният проект първо трябва да бъде съставен и анализиран от Woboq. Анализът се извършва от LLVM / Clang инфраструктура, което прави цялото абстрактно синтаксично дърво достъпно. По този начин всички части от семантичната информация могат да бъдат извлечени със същата семантика. Това също дава недостатък на инструмента, а именно Woboq може да се използва само за сърфиране на C и C ++ проекти. OpenGrok [4] е бърза машина за търсене на източници и препратки. За разлика от Woboq, този инструмент не извършва задълбочен анализ на езика, следователно не е в състояние да предостави семантична информация за конкретните образувания. Вместо това, той използва Ctags [5] за анализиране на изходния код само текстово и за определяне на типа на специфичните елементи. Простият синтактичен анализ дава възможност за разграничаване на имена, функции, променливи, класове и др. Търсенето сред тях е силно оптимизирано и следователно много бързо дори при големи бази от кодове. Търсенето може да се извърши чрез сложни изрази (напр. Defs: target), съдържащи дори подсказки. Освен това резултатите могат да бъдат ограничени до поддиректории. В допълнение, за търсене на текст има възможност да намерите символи или определения отделно.

Липсата на семантичен анализ позволява на Ctags да поддържа 41 програмни езика. Предимство на този подход е, че е възможно постепенно да се актуализира базата данни с индекси. OpenGrok също така дава възможност за събиране на информация от системи за контрол на версии като Mercurial, SVN, CSV и др. Understand [6] е не само инструмент за сърфиране на код, но и цялостен IDE.

Голямото му предимство е, че изходният код може да бъде редактиран и промените в анализа могат веднага да се видят.

Освен функциите за сърфиране на кодове, които вече бяха споменати за предишни инструменти, Understand предоставя много показатели и отчети. Някои от тях са кодови редове (общ / среден / максимум в световен мащаб или за клас), брой свързани / базови / производни класове, липса на свързаност [2], сложност на McCabe [1] и много други. Treemap е общ метод за представяне за всички показатели. Това е вграден правоъгълен изглед, в който гнездането представлява йерархията на елементите, а цвета и размера на величините представляват показателя, избран от потребителя.

За големи кодови бази е необходима проверка на архитектурата. Разбирането може да показва диаграми на зависимост въз основа на различни отношения като йерархия на извикванията на функции, наследяване на класове, зависимост на файлове, включване / импортиране на файлове. Потребителите могат също така да създават своя персонализиран тип диаграма чрез API, предоставен от инструмента. В програмирането основните понятия са често срещани в различните езици, но има някои понятия, които на определен език се интерпретират по различен начин. Understanding може да работи с 15 езика и може да предостави специфична информация за езика на кода, например, анализ на pointer function в C / C ++ или диаграми на йерархията на пакетите в Ada.

Understanding, изгражда база данни от кодовата база. Цялата информация може да бъде събрана чрез програмируем API. По този начин потребителят може да поиска цялата необходима информация, която не е включена в потребителския интерфейс. CodeSurfer [7] е подобен на Understand в смисъл, че е и сложно клиентско приложение за статичен анализ. Целта му е разбиране на проекти за машинен код на C / C ++ или x86. CodeSurfer извършва задълбочен анализ на езика, който предоставя подробна информация за работата на софтуера. Например, той осъществява анализ на pointer, за да провери кои pointers могат да показват дадена променлива. Освен това той прави списък на твърденията, които зависят от основното твърдение чрез анализ на въздействието и използва анализ на потока от данни, за да определи къде е била присвоена стойността на променливата.

Архитектурата на CODECOMPASS

В предишния раздел изброихме някои аспекти от целите и архитектурата на инструментите за разбиране на кода. Сега ще представяме мястото на CodeCompass сред тези инструменти.

CodeCompass има архитектура тип клиент-сървър, в която представя информацията, събрана в предходна фаза на анализа. Причината, поради която тази архитектура е избрана, е целта на инструмента. За разлика от редакторите на кодове, Code-Compass е планиран да бъде инструмент за разбиране на кода. Има основни разлики между тези два случая на използване. По време на писане на код програмистите манипулират само няколко файла едновременно. При разбирането на кода, обаче, е необходимо да се вземат предвид източниците на множество модули чрез кодова база. В редакторите попълването на код е една от най-полезните функции: програмистът не иска да помни всички методи и полета на клас, но изисква редактора да ги изброява. При разбирането на кода е необходим широк спектър от визуализации, за да се прегледат връзките на отделните части код. Докато редактира основния код, програмистът се фокусира само върху сравнително малък фрагмент от него, като функция или клас. При разбирането на кода програмистите не само следят функциите на ниско ниво, но зависимостите между тях и резултатите се разглеждат в контекста на жялостната модулна система на високо ниво.

Основният потребителски интерфейс на CodeCompass е уеб-базиран. Всички горепосочени визуализации и функционалности могат да бъдат заявени чрез публичен API, който е свързан със сървърно приложение. Уеб интерфейсът обработва случаите на използване, които са насочени към бързи и удобни задачи за сърфиране, проверка и разбиране. CodeCompass, обаче, е нещо повече от инструмент за търсене на код. Той е също така и рамка, т.е. разширяващ се колектор и презентатор на статични процеси за анализ. Това е така, защото намерението не беше да се създаде приложение, с което клиентът трудно да работи и което съхранява резултатите от анализа, но да бъде в състояние да обслужва различните нужди на потребителите. По този начин е възможно да се реализира скрипт, например, който събира набора от

функции, които образуват затваряне по отношение на функция извикване, като по този начин се посочва кохерентна част от софтуера.

Друго дизайнерско изискване на CodeCompass беше да борави с широкомащабни кодови бази и все пак много бързо да отговаря на потребителски заявки, т.е. най-много секунди. Това се постига чрез съхраняване на цялото количество информация в бази данни, които са достатъчни, за да отговорят на изискванията. Тъй като възнамеряваме да дадем точни информация за изискванията, е необходим предходен анализ. В първия запазихме цялото абстрактно синтаксично дърво на източника, но това доведе до съотношение 1: 1000 между изходния код и размера на базата данни. Оказа се обаче, че в повечето случаи потребителите се интересуват само от именуваните единици (функция, променливи, класове, макроси и т.н.), така че не е необходимо да се съхраняват данни като контролни структури или други твърдения. Независимо от това, има някои задачи, които изискват повече отсамо представяне на съхранената информация. Такъв пример е алгоритъм за нарязване, където потребителят иска да види ефекта от промяната на стойността на дадена променлива. В такъв случй също трябва да се вземат предвид извлеченията за промяна на състоянието. Това изисква повторното преразглеждане на кода в движение.

Функции CODECOMPASS

В този раздел преглеждаме функциите, налични чрез стандартния графичен интерфейс. Когато описваме специфични за езика функции, като изброяване на callers на метод, винаги ще приемаме, че езикът на проекта е C ++, тъй като има най-модерната поддръжка в CodeCompass, но подобни функции са налични за Java и Python.

Търсене

Вероятно най-фундаменталният случай на използване на инструмент за разбиране на код е търсенето. Човек може да търси или файл, или изходен код. За намиране на елементи от изходния код, инструментът предоставя 3 различни възможности за търсене:

В режим на пълнотекстово търсене фразата за търсене е група от думи като „returns an astnode*“. Въпросната фраза съвпада с текстов блок, ако търсените думи са една до друга в изходния код в този конкретен ред. Символи като *, или? могат да се използват, като съответства на всеки множествен или единичен знак. Логически оператори като AND, OR и NOT могат да бъдат използвани за свързване на няколко запитвания едновременно.

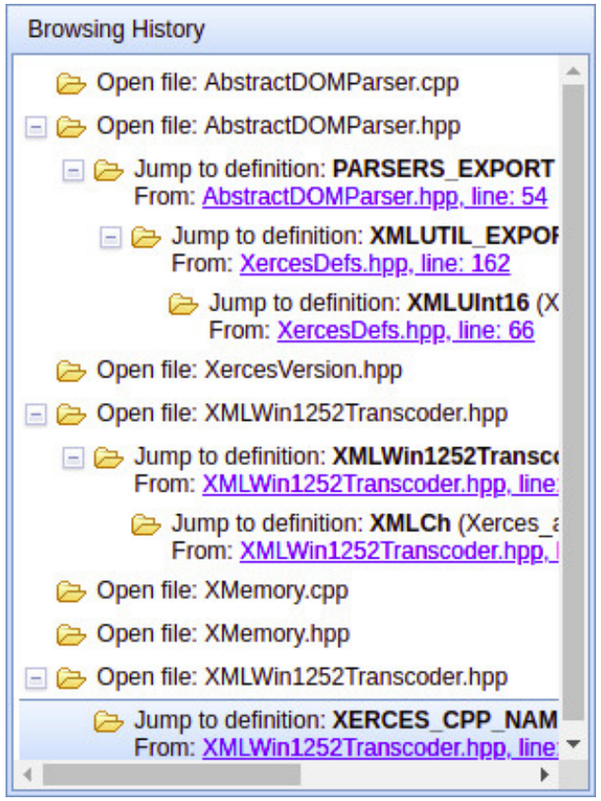
На по-високо ниво е възможно да намерите символи в изходните кодове чрез търсене по дефиниция. Тук използваме CTags за индексиране на кодовата база, като по този начин можем да намерим променливи, функции, класове, макроси и др. Важно е да знаем, че това търсене на езикова единица няма нищо общо с дълбокия анализ на езика.

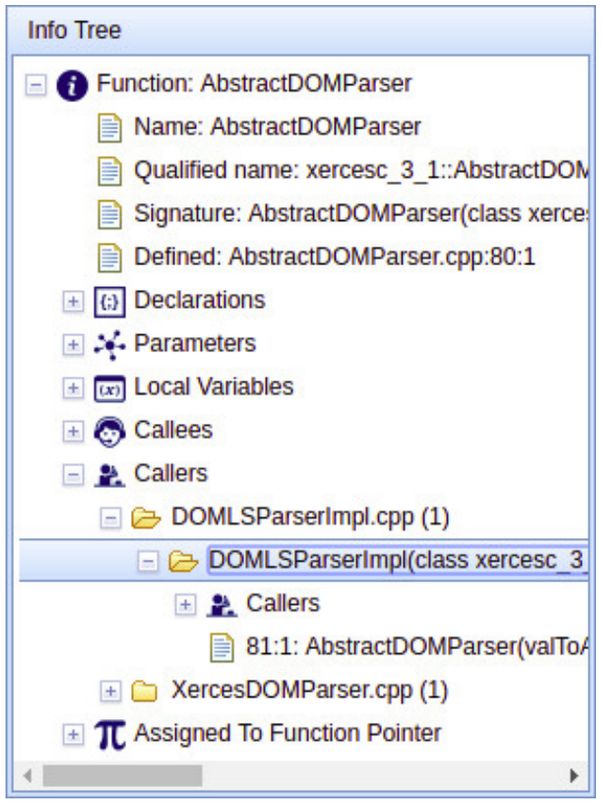
Докато отстранявате грешки в програмата, понякога единствената информация, която имате е изходно съобщение в log на конзолата, предоставен от нашия софтуер. Това е единствената следа, от която човек може да започне, напр. " DEBUG INFO: TSTHan: sys offset = -0.019821, drift comp = -90.4996, sys anketa = 5". Обърнете внимание, че такова съобщение може да съдържа времеви маркери или други динамично генерирани фрагменти, така че е невъзможно да се намери това съобщение като директен низ. Въпреки това, в CodeCompass н толкова конкретно търсене може да се извърши чрез търсене в log.

Информация за езиковите символи

Когато елементът е намерен, следващата стъпка е събирането на информация за него. Потребителят може да избере „Info tree“ от изскачащото меню, след като избере име. Това дърво съдържа цялата информация, която се предоставя от езиков анализатор. В случай на C / C ++ използваме компилатора LLVM / Clang, за да извлечем информация за символите.

За функции можем да проверим техните параметри, локални променливи, callers и callees. Интересна особеност на дървото е, че callers се се представят рекурсивно, т.е. децата на възел са callers на функция. Тяхните children nodes са callers се на тези функции и това продължава рекурсивно, теоретично обратно към основната функция.





Function calls, обаче, не винаги са директни, но могат да се случат чрез функционални указатели. Въпреки че това е динамична работа, CodeCompass посочва всички събития, при които функция е била в действие и извикването се случва чрез този pointer.

В случая на клас, събраната информация са псевдоними (по typedef класът може да има синоним), наследствени отношения (групирани по видимост), методи / полета (директни или наследени) и използвания (като локална / глобална променлива, параметър на функцията / тип връщане или поле от друг клас).

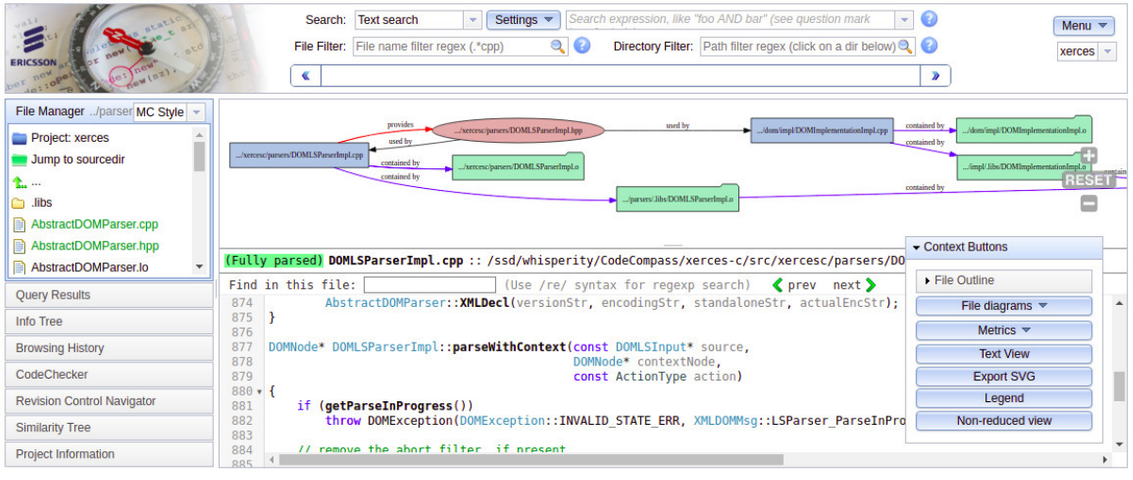
За променливи е полезно да знаете местата в кода, където е написано и прочетено. За типовете изброяване, константните за преброяване са посочени с цели числа.

Диаграми

Визуализациите са едно от най-полезните методи за представяне на хората при преглед на система. CodeCompass представя няколко диаграми, базирани на символи и файлове. Тези диаграми са базирани на графи, т.е. те представляват единици и техните връзки.

Това са и интерактивни диаграми: при придвижване на мишката върху възлите, представената единица се показва в текстовия изглед и с щракването върху тях избраното образувание става централен възел, показващ връзките му според типа на диаграмата.

Функционалната диаграма за повикване показва всички callers и callees на дадена функция в графика. Диаграмата за наследяване на UML клас показва цялата верига на наследяване до основния клас и рекурсивно за всички производни класове. Приложихме също диаграма за анализ на pointer, която показва разпределените обекти и pointers, които ги насочват към тях. Разбира се, това е динамична информация, която може да се събере само отчасти при статичен анализ.



Интерфейсна диаграма за изходен файл на C / C ++, показва кои заглавия са „само използвани“ или „изпълнени“ от дадения файл. Използвани означава, че изходният файл използва друг файл, ако в него има използване на символ, който е деклариран в другия файл. Изпълнени означава, че символ се декларира във файл (по този начин формира интерфейс) и се дефинира в друг. Тези отношения са приложими и за директории, които имат предвид съдържащите се файлове. При съставен език съществуват и изходни файлове като обекти и изпълними файлове. Въз основа на информацията за връзката можем да посочим кои източници съставят двоичен файл.

CodeBites осигурява различна визуализация на инспектирания изходен код. В този изглед възлите на графиката са дефинициите на конкретни именувани символи, като класове, функции и т.н. Идеята е програмистът да иска да открие този обект, като разбере как работи му, но без да губи фокус. Така частите от текста на кода в даден възел могат да се селектират, което активира добавянето на определението за избрания елемент.

Визуализации за управление на версиите

Визуализацията на информацията за контрол на версиите е важно помощно средство за разбиране на развитието на софтуера. Git blame view показва ред по ред промените (задачите) към даден файл. Скорошните промени са оцветени по-светло зелено, докато по-старите промени са по-тъмно червени. Този изглед е отличен за преглед, защото определени редове са добавени към изходния файл. CodeCompass може също така да показва Git задачи в филтраблен списък, подредени по време на извършване. Това средство за търсене може да се използва за изброяване на промени, направени от дадено лице, или за филтриране на задачи по съответните думи в съобщението за задачи.

Метрики

CodeCompass може да покаже McCabe Cyclomatic Complexity [1], редовете на кода и броя на грешките, открити от показателите на Clang Static Analyzer за отделни файлове и обобщени в директивните йерархии. Тези показатели могат да бъдат визуализирани на tree map, където директории са обозначени с полета. Размерът на полето и нейният цветен нюанс е пропорционален на избрания показател.

Browsing history

Де Алуис и Мърфи правят изследване в областта на история на сърфиране, защото програмистите изпитват дезориентация, когато използват интегрираната среда за разработка (IDE) на Eclipse Java [8]. Те използват визуална инерция [9], техника, с която да идентифицират три фактора, които могат да доведат до дезориентация: i липсата на свързващ навигационен контекст по време на проучването на програмата, ii смяната между дисплеи за преглед на необходимите части от кода и iii изследване на понякога несвързани подзадачи.

Първият фактор означава, че програмистът по време на изследване на проблем посещава няколко файла, като следва веригата за повикване или изследва използването на променлива. В края на дълга проучвателна сесия е трудно да си спомним защо изследването е попаднало в конкретен файл.

Втората причина за дезориентация е честата смяна на различни изгледи в Eclipse. Третият проблем е, че, когато програмист решава задача за промяна на програма, оценява няколко хипотези, т.е. всички са индивидуални подзадачи за разбиране. Програмистите са склонни да преустановят подзадача (преди да я завършат) и да преминат към друга. Например, ако програмистът изследва как се използва връщащата се стойност на дадена функция, но след това променя в подзадачата и разбира изпълнението на самата функция. Характерно е за програмист трудно да се сеща за подзадача, която не е в действие [10].

CodeCompass използва изглед на историята на сърфиране, който записва (под формата на дърво) пътя на навигацията в изходния код. Нова подзадача се представя от нов клон на дървото, докато възлите са навигационни скокове в кода, обозначени от свързващия контекст (като „jump to the definition of init“). Така проблем i) и ii) се адресират от възлите с етикети в историята на сърфирането, докато проблем iii) се решава от клоновете, присвоени от подзадачите.

CodeChecker - C/C++ отчитане на грешки

Clang Static Analyzer внедрява усъвършенствана машина за символично изпълнение, за да съобщава за програмни грешки. CodeCompass може да визуализира грешките, идентифицирани на Clang Static Analyzer и Clang Tidy, като го свърже към сървъра на CodeChecker [11]. CodeCompass показва позицията на грешката и символния път за изпълнение, които водят до грешка.

Пространство с имена и каталог

CodeCompass обработва документацията на Doxygen и ги съхранява за дефинициите на променливата за функция и тип. Той също така предоставя изглед на каталог за тип, който изброява типове, декларирани в работното пространство, организирани от йерархичен изглед на дърво за пространствено именуване.

Резюме

Представихме CodeCompass, инструмент за статичен анализ за разбиране на голям по мащаб софтуер. Той е създаден, за да се реши проблема с недостига на съществуващите инструменти за разбиране, някои от които са леки, лесни за използване, но без дълбокото познаване; а други, които представляват тежка, не мащабируема машина. Имайки уеб базирана, подвижна, скалируема архитектура, рамката може да бъде отворена платформа за по-нататъшно разбиране на кода, статичен анализ и изчисляване на софтуерни метрики. Първоначалните отзиви на потребителите и статистиката за използване предполага
че инструментът е полезен за програмистите при разбиране на код и се използва освен в традиционните IDE и в други инструменти за препратка.

Източници

[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.