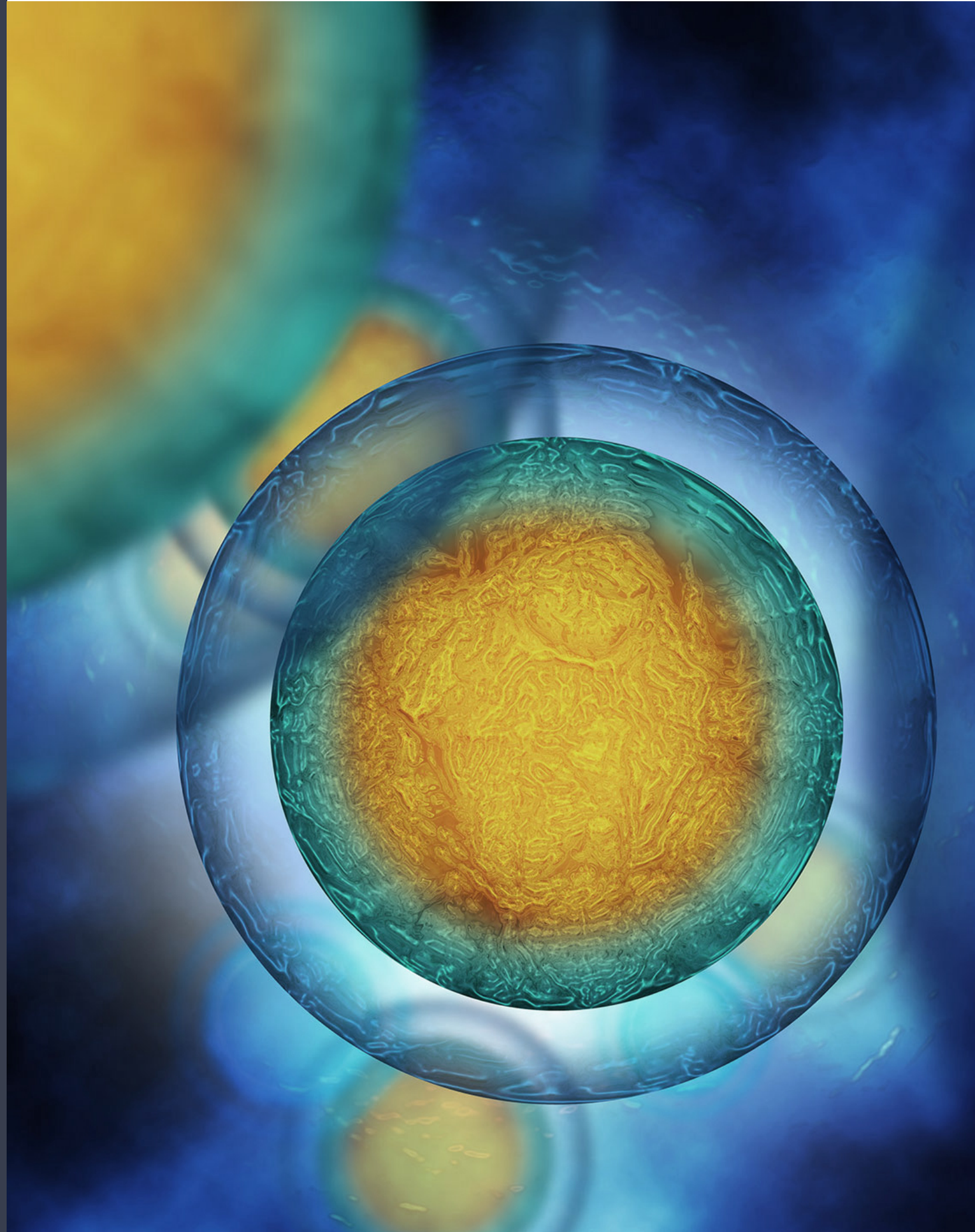
FE3CWS

AMSTER-DAM TEACHER TRAINING MATERIAL

Intellectual output 2 of the ERASMUS+ project 2017-1-SK01-KA203-035402



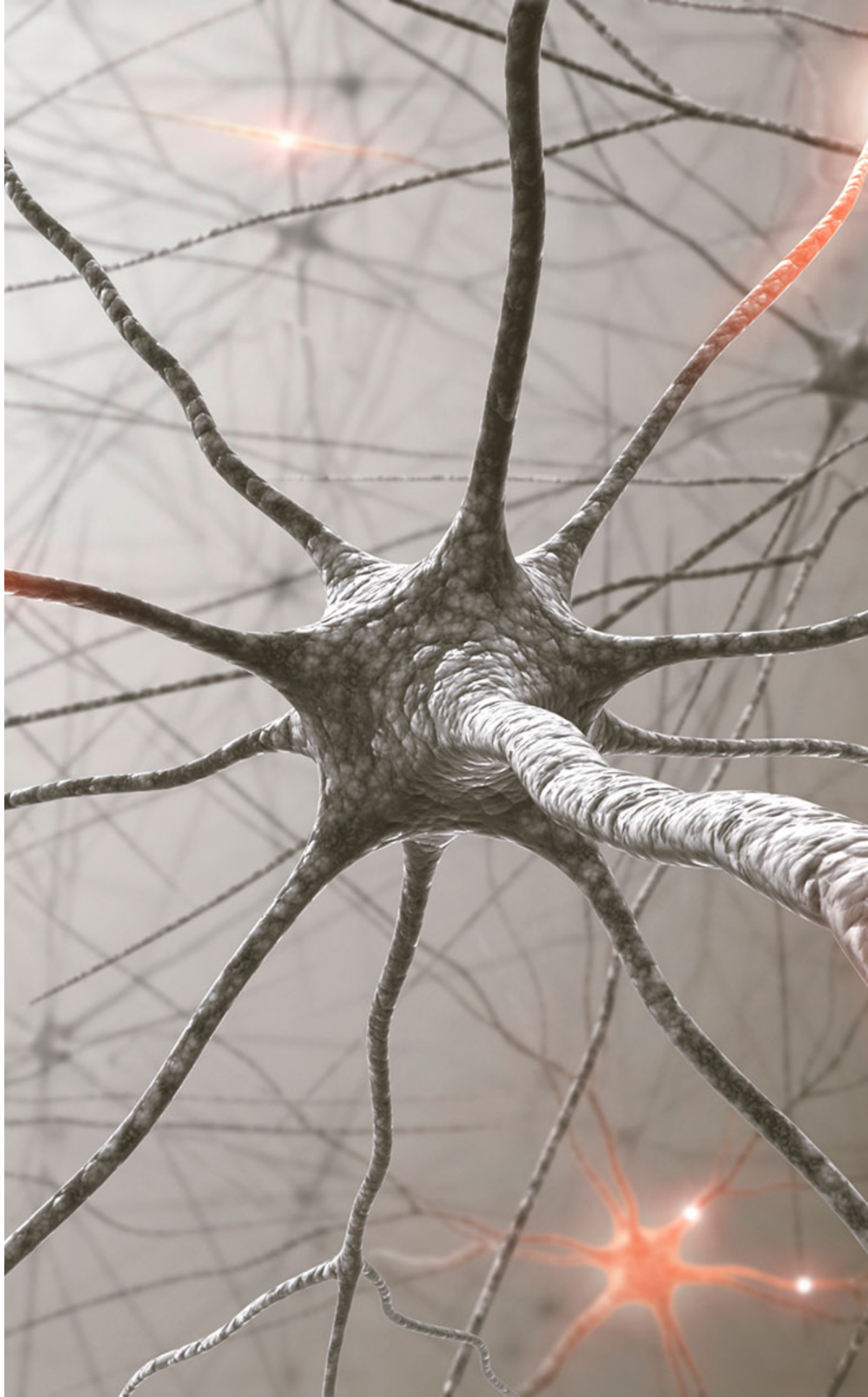
Some words about the

CONTENTS

- 6 topics related to software composition, comprehension and correctness
- Available in 7 languages: English, Hungarian, Slovak, Croatian, Romanian, Bulgarian and Portuguese

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

TABLE OF CONTENTS



USER-CENTRIC CLOUD COMPUTING IN EDUCATION

Cloud computing has become a key technology and therefore part of many computer science curricula. User-centric research focuses on strategies for estimating runtimes and costs for deploying real-world applications on the cloud.

Within the area of cloud computing an important aspect is assisting users in their decisions. Such decisions relate to the following questions:

– How does the application behave on the virtualized resources?

– How many virtual resources of what type from which cloud provider should be acquired for application deployment?

– For how long? How much will it cost?

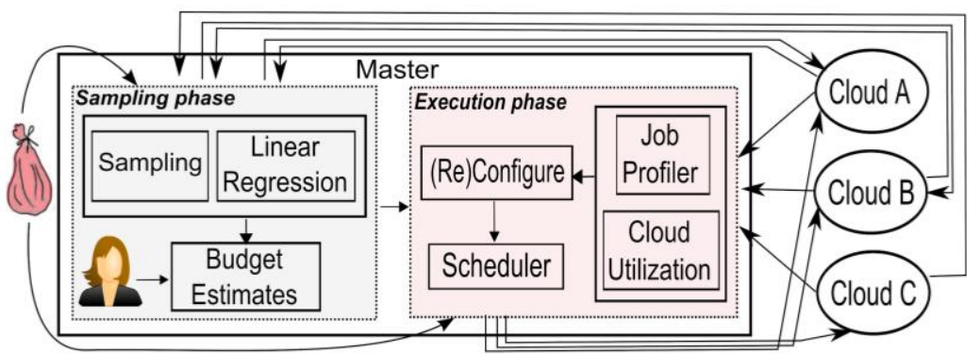
These questions are typically modeled as a scheduling problem, working under the assumption that there is no a-priori knowledge about the application. One typical, simple set of requirements is that the application is successfully deployed, and the costs are minimized.

THE ARCHITECTURE OF A SCHEDULER FOR A CLOUD APPLICATION

The BaTS scheduler [4] has been developed to assist users when deploying their applications to the cloud. It takes a self-scheduling approach to achieve this and it regularly checks the deployment progress.

Figure 1 depicts the architecture of BaTS. During the sampling phase, BaTS collects statistics on the runtimes of some of the application's tasks, using sampling with replacement. Here, only a small sample is needed (30-50 tasks) to compute the mean and the standard deviation of the tasks' runtime on various cloud offerings. Linear regression is used to optimize the computation of budget and makespan estimates.

During the execution phase, at regular time intervals, the current configuration is re-evaluated, to check whether the selected schedule is still feasible.



If a budget violation is expected, more profitable machines (better price/performance ratio) are acquired. If a makespan violation is expected, more faster machines are acquired.

USING THE BATS METHODOLOGY ON AWS RESOURCES

We introduce the problem of assisting application owners looking to select the best choices in terms of virtualized resources when deploying their application on Amazon EC2 (AWS) [7] resources.

The optimized sampling phase

The main idea is to use the average execution time for each virtualized resource type to compute budget and makespan estimates.

However, obtaining these statistics may incur significant costs given the many types of AWS EC2 offerings (currently 123 [8]). Figure 2 shows randomly selected tasks from an application where tasks' runtimes follow some distribution. The runtimes of these tasks are used to compute the statistics for one cloud offering. After collecting the statistics for every available cloud offering, we may compute the budget and makespan estimates. Assuming that we would like to evaluate every current AWS EC2 offering, that would mean executing 30 tasks on each of the 123 machine types. If we would simply execute different sets of randomly selected tasks (in total 3690), it would lead to a long (and costly) sampling phase, possibly rendering any user decision irrelevant for two reasons: a) there are too few tasks left to be executed, and b) the user budget might be already exceeded. If we would execute the same set of randomly selected on each machine type, it would still lead to a long (and costly) sampling phase.

We optimize this phase by using linear regression to reduce the total number of tasks that need to be executed before preparing statistics. We execute the same set of 7 randomly selected tasks on each machine type and collect runtimes [9].

Next, we execute 23 randomly selected tasks on the machines that first become available. Figure 3 illustrates our approach. Using the runtimes of the replicated 7 tasks we establish a linear relationship between execution times for the tasks of the application across all machine types. We use these linear relationships to then map the 23 runtimes to all other machine types. Once the mapping is finished, we have a set of 30 runtimes for each machine type, while only executing 884 tasks instead of 3690.

Different orders of magnitude in pricing

Once the runtime estimates are obtained, the budget and makespan estimates are computed using a modified Bounded Knapsack algorithm [11]. However, when considering AWS EC2 resources with a different pricing model, such as spot instances, this approach does not scale due to the different orders of magnitude.

To address this issue, we traded the determinism of the Bounded Knapsack approach for the scalability of a genetic algorithm approach [12].

Using a genetic algorithm, we could approximate the Pareto front (optimal set) of feasible schedules for a given application and a given set of machine types. Figure 4 shows the real Pareto front and two estimates for an application having a multi-modal distribution of task runtimes. Our solution generates accurate Pareto fronts within 1 second.

The key finding here was that to ensure a good coverage of the real Pareto front, the fitness function should also reward fastest/cheapest makespan.

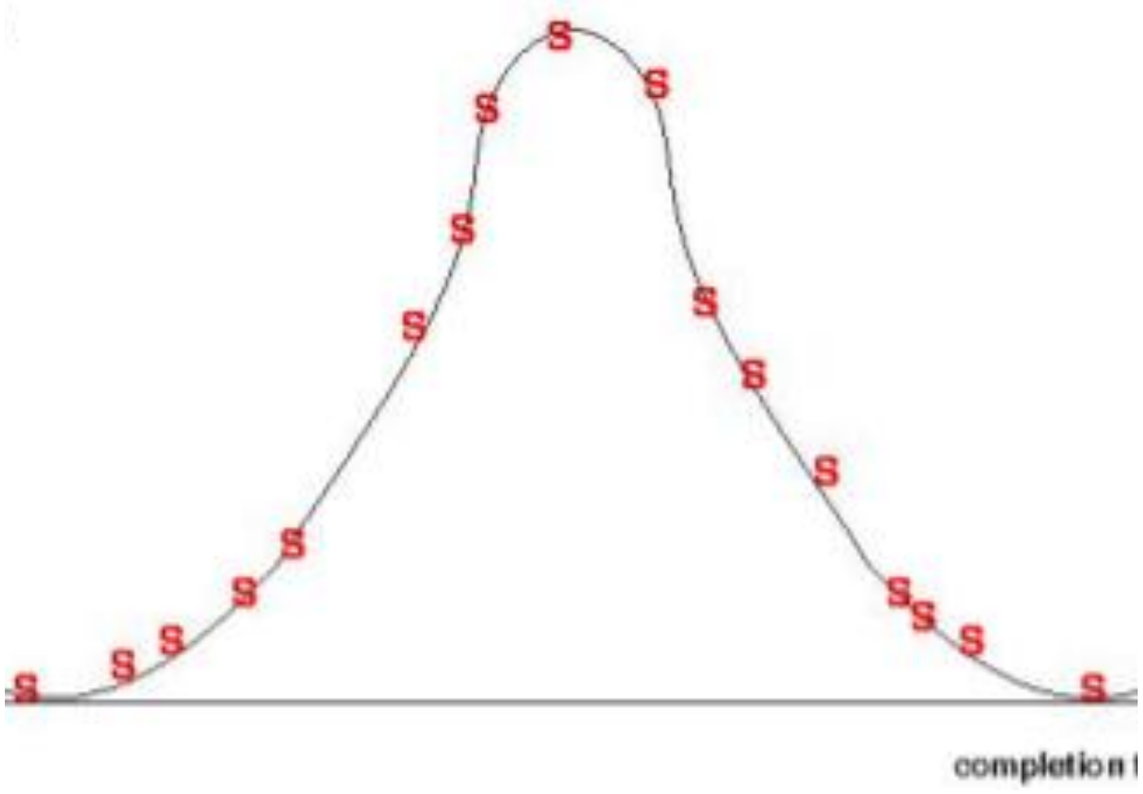
The tail phase of the computation

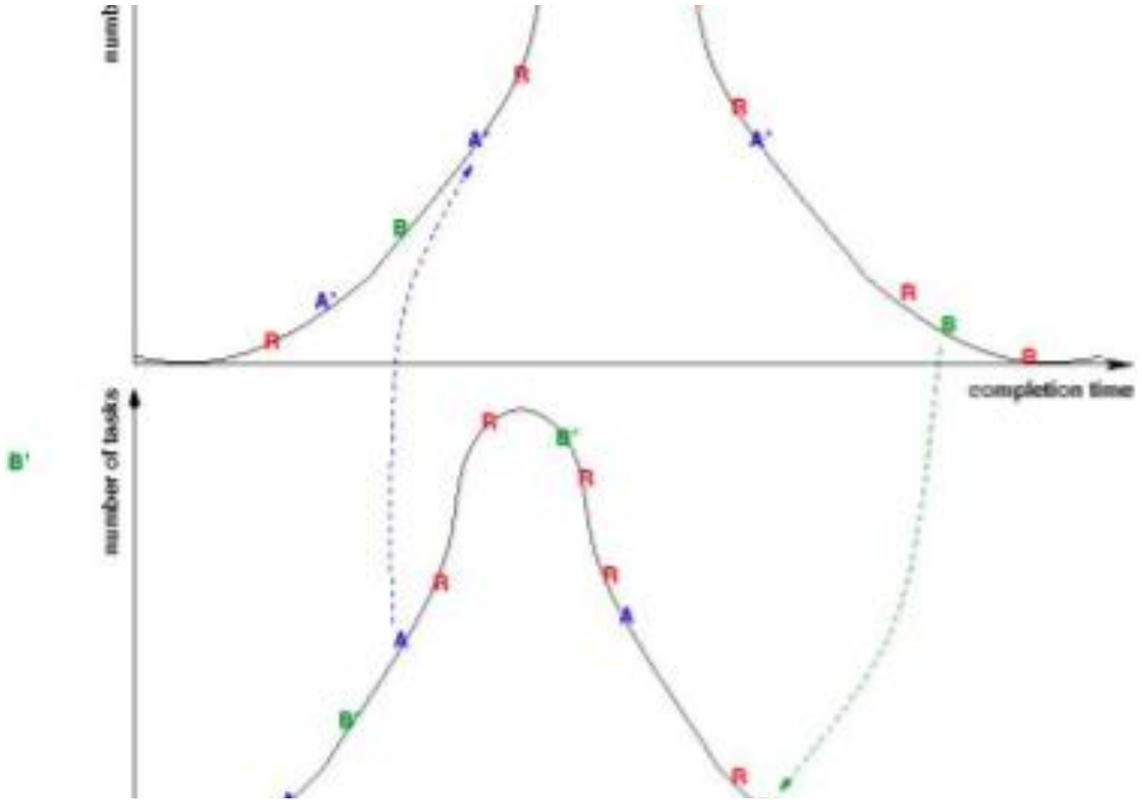
In the final phase of the computation, the underlying assumption that computing time is "fluid" needs to be addressed [10]. At this point, by definition, the application contains too few tasks for the assumption to hold. We analyzed several approaches to address this problem, focusing on better estimating the final computation needs of the application, such that the allocation of the final tasks to machines would be optimal. Our approaches ranged from perfect knowledge of remaining runtimes to zero-knowledge (random runtimes). The impact was insignificant. Therefore, the problem needed to be addressed at a different stage, namely at the budget and makespan estimation phase.

To that end, BaTS keeps track of the estimated unused final accountable time units fractions. BaTS provisions a cushion to address outliers running outside the final accountable time unit and adds virtualized resources and/or time to the schedule. However, outliers may still lead to violations.

Using the BaTS methodology in education

In the University of Amsterdam's Master of Computer Science curriculum the course "Web services and cloud-based systems" has been taught for several years. One important goal of this course is to familiarize students with the challenges of cloud-based systems. The practical work for this course entails developing a rudimentary scheduler for virtualized resources and analyzing its behaviour in an in-house setting versus a commercial cloud one. The in-house setting consists of an OpenNebula [5] deployment on DAS [6], the Dutch national computing cluster.

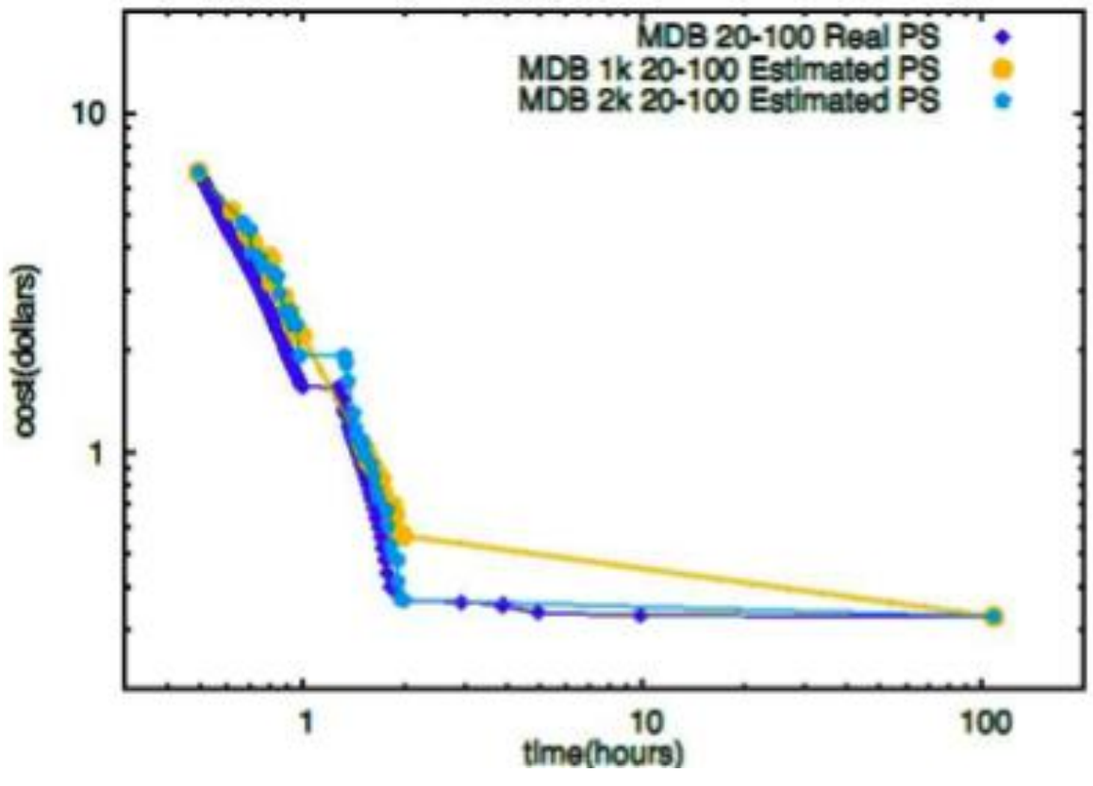




OpenNebula is an open source solution for Infrastructure-as-a-Service deployments: physical resources are managed and offered as virtual resources. Here, students have an upper limit on the number of virtual machines that can be fired up concurrently.

The commercial cloud setting consists of the Amazon EC2 [7] cloud offerings. Here, the students have a budget they can use for any lab-related resource acquisition. The assessment of their lab work takes into account any budget violations.

In general, the outcome of the lab work showed that students were able to understand the difference between best-effort and commercial virtual resource acquisition. They usually developed schedulers driven by profitability, while the best performing students developed more sophisticated schedulers where the users could select from several policies: fastest, cheapest, most profitable.



CONCLUSION & FUTURE WORK

Stochastic approaches for user-centric cloud scheduling are promising. Embedding research in education as soon as it reaches some stable state is very important.

As future work, we would like to support Haskell AWS Lambda functions deployed through the Haskell AWS API implementation.

References

[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.

FOCUSING EDUCATION ON ENERGY EFFICIENCY MEASUREMENTS DURING SOFTWARE TESTING

The mission of software engineering teachers is to prepare future software engineers who can master every problem during the whole software life cycle. Besides skills related to comprehension of stakeholders' needs and to composition of corresponding working software, the ability to check correctness of the results plays also a significant role.

In this paper, we focus on the latter set of skills. We focus on testing, where the level of automation is less important. We emphasize the measurement nature of testing, more precisely, we focus on software energy consumption measurement that can be provided during testing at different levels. All these aspects are presented from the point of view of a software engineering educator, with the aim to present how to set up a software engineering lab session that focuses on energy efficiency measurement during software testing and, with the authors' comments on this proposal.

PROBLEM DEFINITION

One of the challenges for battery manufacturers is how long the battery can operate without being continuously charged, and of course there are many other challenges such as the size that greatly affects the shape of the device, and the other factor which is an important feature of the battery is the weight of the identifier. The battery is considered to be somewhat lighter compared to the device that needs a battery to operate. The challenge here is how to make its size smaller and lighter weight and certainly a high efficiency in terms of operating time of the mobile device without being charged.

On the top of this hardware challenge, its software challenge brother exists, namely that the software itself should support energy savings. Doing that without limiting the user experience is considered nowadays as a silent but important goal of each software development that is targeting any kind of portable devices.

Remembering that the energy consumption of any mobile device is influenced starting with the running applications through the access level of basic services up to the actual mood of the user, to develop software for such devices is already a challenge [1]. One could say, the software development challenge is always the same, but we have to point out “mobility” as key system property here. Battery power status also determines the system performance due to operating system level configuration – well known as “energy saving preferences”.

It is even harder, if one (in our case the teacher) has to prepare students for such challenges. All the already known “best practices” and “energy saving tips” have to be presented in a context, which is easily comprehensible to the students.

This can be done by positioning the concepts into a known environment suchas software testing and test automation [2], in our case. This is what we aim with this paper, this is the content of the upcoming sections, starting with the proposal followed by an evaluation and closing with further tips on improvement.

SOLUTION PROPOSAL

As it was stated above, we have to find the best suitable environment to introduce energy consumption measurement [3] and energy efficiency evaluation practices.

It could be both initial software development and software evolution as well, as both development phases of the generic software life cycle offer opportunities to measure the product being developed/evolved [4].

With the choice of initial software development, the benefit is that all activities could get a focus on energy saving related issues, while choosing the software evolution alternative offers the possibility to evaluate the improvement in the product implementation. On the other hand-side, software evolution requires the existence of a working software at its beginning, while initial software development is the process that creates the product starting with the first requirement on the software.

Putting the two possible approaches into the teaching environment, the best option would be to use both. One semester for initial development of software and another one for the evolution of the same software. Usually the teacher does not have two semesters in a row to present the course contents in the way presented in the previous paragraph. This is the reason why we have to decide which kind of development to use to introduce the selected practices. From the point of view of the development process architecture, and by the fact that initial development could be also evolutionary, we decide for software evolution. It includes many activities of initial development (except early requirement gathering and analysis) and emphasizes the importance of testing and evaluation.

This choice allows the teacher to:

– Let students look back in their development history (past projects) to be critical to themselves.

– Let them evaluate their results using code metrics and energy consumption measurement (or estimation).

– Let them integrate the above activities into the standard verification & validation processes of software evolution.

Programming language of the development is not important, therefore the student can select any of their previous project for the evolution – or all of them, if they are competing in the number of evolved projects or programming languages used. But, programming language usually determines or limits the development environment and tools used. The selection of these tools and their plug-ins also offers a good support for code metrics evaluation. Energy consumption measurement and energy efficiency evaluation usually requires a different tool, as there are only few development environment that integrate energy consumption measurement or estimation by now. Regarding testing as measurement basis, we have to note that static code analysis is used to be a part of testing of software. Besides that, selected parts of the application code base are being executed during white box testing (mainly unit testing), which by a small extension of energy consumption measurement can present the energy consumption of the test case – an indirect look at energy consumption of the tested code. During black box testing, the complete application is being tested using test scenarios. These test scenarios are mainly comparable to intended all-day use cases of the software, others represent the borderline scenarios – including the ones when the user is in bad mood. Measuring energy consumption of the execution of these black box tests then gives an approximate (but direct) look at energy consumption of the product.

Here is the main benefit of evolution (when compared to initial software development). The teacher can prepare the starting version for the evolution, including code that can be improved, list of known bugs and the test base! The existence of test for retesting and regression testing is very important here as the productivity of evolution can be increased by this feature.

Assuming all above steps were made, the integration of energy efficiency measurements into the process of testing look from the student's perspective of activities as follows:

1. Select the product that could be your past project or from a repository.

2. Evaluate it using static code analysis, testing, energy measurement, usability survey, etc.

3. Improve it (different kinds of evolution such as add/change functionality, repair or adapt)

4. Retest to be sure you eliminated a defect or fault

5. Regression test (including re-evaluation)

6. Conclude results (make a final verdict across all available data, including energy efficiency).

DISCUSSION

As energy consumption measurement is relatively new compared to other techniques such as static code analysis, black/white box testing and debugging, it might be the point of failure. But if it gets combined with these elder principles building a composite score for each student, the critical property is much lower.

Looking at the possibilities of grading, we can find various "levels of freedom", which offer themselves to be used separately or as part of a grade composition:

1. number of different projects,

2. number of programming languages applied/used,

3. code quality of the final product,

4. energy efficiency of the final product,

5. code quality improvement during software evolution,

6. energy efficiency improvement during software evolution.

Considering all "levels of freedom" of task completing, many competitions could be defined for the competitive students, while the achievers will collect the "small victories" in number of projects, the perfectionists' goal will be to optimize all code metrics and minimize energy consumption. For the average student, improvement of energy efficiency and code comprehensibility could be the achievable goal.

Student competition can be even more supported by not letting them return to their past projects, but allowing them to choose from a selected repository.

Like with computer games, all game levels are then equally available to everyone.

To support equity as well, a common public forum of students and teachers could be also created.

Our future work in this area will focus on configuration of a portable integrated development, testing and energy consumption estimation environment.

This environment will be used in the frame of software evolution or initial development subjects to support education on energy efficiency measurement during software testing. It might limit students' creativity by offering a semi-closed sandbox, as hardware plays a very important role by the current architecture of energy consumption estimation. Some research aims to break this limitation – we are looking forward to those results to get them also integrated.

References

[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.

TOWARDS AN ENGINEERING DISCIPLINE FOR GREEN SOFTWARE

This technical report describes the research developed at the Green Software Laboratory at Coimbra and Minho Universities, which was presented at the first teachers training meeting of the Erasmus+ project “Focusing Education on Composability, Comprehensibility and Correctness of Working Software”. It presents both a green ranking for programming languages and data structures, and techniques to locate abnormal energy usage in software systems.

MOTIVATION

The current widespread use of non-wired but powerful computing devices, such as, smartphones, laptops, etc., is changing the way both computer manufacturers and software engineers develop their products. In fact, computer/software execution time, which was the primary goal in the last century, is no longer the only concern. Energy consumption is becoming an increasing bottleneck for both hardware and software systems. As a consequence, research on green software is a relevant and active area of research. This report briefly describes the research that is being developed in green software in the Green Software Laboratory (GSL). GSL consists of various Portuguese research groups, including two sites of the project “Focusing Education on Composability, Comprehensibility and Correctness of Working Software”. GSL is an initiative to develop techniques and tools aiming at reducing energy consumption across various computing systems (mobile, programs, databases, etc.).

GSL specifically focus on the software side, where it applies (source code) analysis and transformation techniques to detect anomalies in energy consumption and to define optimizations to reduce such consumption.

In the last century efficiency of a software system was mainly focused on execution time and memory consumption efficiency. Nowadays, software developers often ask the question “is a faster program also a greener program?”. There are many aspects of a software system that influences its energy performance: the programming language and its execution model (compiled to binary code or to a virtual machine, interpreted code, lazy versus strict evaluation, use of runtime partial evaluation, etc). The efficiency of the memory model and language libraries also influence performance. The complexity of the algorithm used to implement the desired computer problem, also influences performance: if the implemented algorithm has to do more work than what is strictly needed, then, more CPU and energy will be used.

In this document we briefly report the research results achieved in the GSL, namely in analyzing the energy efficiency of programming languages (Section 2), data structure libraries (Section 3), and of software’s source code (Section 4).

GREENNESS IN PROGRAMMING LANGUAGES

An interesting question that arises when discussing energy in programming languages is whether a faster language is also an energy efficient language, or not. Comparing software languages, however, is an extremely complex task, since the performance of a language is influenced by the quality of its compiler, virtual machine, garbage collector, etc. In the Green Software Laboratory we studied, assessed and compared the performance of (a total of) 27 of the most widely used software languages. We used two different computer problem repositories: Computer Language Bench- mark Game (CLBG)3 and the Rosetta Code4 repositories [1–3]. Both repositories define a set of computer tasks and provide implementations in a large group of programming languages. While CLBG was tailored to analyze execution time performance of languages, Rosetta Code was defined with more program comprehension purposes.

We compiled/executed such programs using the state-of-the-art compilers, virtual machines, interpreters, and libraries for each language. Then, we monitored the execution time, peak and overall memory consumption, and CPU/- DRAM/GPU energy consumption. We produced a energy ranking of the 27 languages and we also analyzed those results according to the languages' execution type (compiled, virtual machine and interpreted), and programming paradigm (imperative, functional, object oriented, scripting) used. For each of the execution types and programming paradigms, we compiled a software language ranking according to each objective individually considered (e.g., time or energy consumption). Our first experiments show expected results, like the C language being both the faster and greener language, however, it also show slower languages that are more energy efficient than others [2, 3].

GREENNESS IN DATA STRUCTURES

Programming language/paradigm, and its powerful compiler optimizations, is not the only aspect that influences the energy consumption of a software system. In fact, a program may also become more efficient by "just" optimizing its libraries [4,5]. Most languages offer powerful libraries to manipulate data structures. In GSL we studied the energy performance of two advanced data structures widely used in the Java and Haskell programming languages.

In Java, we conducted a detailed study in terms of energy consumption of the Java Collections Framework (JCF) library 5. We considered the usual three different groups of data structures, namely Sets, Lists, and Maps, and for each of these groups, we studied the energy consumption of each of its different implementations and methods [4]. This JCF energy-awareness can not only be used to steer software developers in writing greener Java software, but also in optimizing legacy Java code. We have developed a Java data structure refactoring tool, named jStanley, which refactors Java source code when a greener collection is available [6]. We have also executed an initial evaluation with 7 publicly available Java projects where we were able to improve the energy consumption between 2% and 17%.

In Haskell, we studied the energy consumption of Edison6, a fully mature and well documented library of purely functional data structures [7]. Edison provides different functional data structures for implementing three types of abstractions: Sequences (lists, queues and staks), Collections (sets and heaps) and Associative Collections (maps and finite relations). We analyzed 16 implementations of such data structures while measuring detailed energy and time metrics [5]. We further investigated the energy consumption impact of using different compilation optimizations. We have concluded that energy consumption is directly proportional to execution time and that the energy consumption of DRAM representing be- tween 15 and 31% of the total energy consumption. Finally, we also concluded that optimizations can have both positive or negative impact on energy consumption.

GREENNESS IN SOURCE CODE

Not only languages and data structure libraries do influence energy consumption, algorithms and programming practices also play a key role on the efficiency of programs. In GSL we have adapted well-know fault localization techniques to statically locate “energy leaks” (seen as energy inefficiency, thus, energy faults) in the source code of applications [8–11]. We defined SPELL - SPectrum-based Energy Leak Localization to determine red (energy inefficient) areas in software. A first experimental study shows that expert programmers, with access to the energy leaks detects by SPELL, were able to better optimize the energy consumption of the programs (between 15% and 74%), than experts with no information or the information provided by a standard programs (runtime) profiler. We have also studied the energy behaviour of C/C++ programs [12].

The widespread use of non-wired devices and the advent of the internet-of- things, is changing the way software engineers develop their software. Software has to run on a variety of mobile devices and energy consumption is a main concern when developing software. Software Product Lines (SPL) have emerged as an important software engineering discipline allowing the development of soft- ware that shares a common set of features. In GSL we have defined static analysis techniques to reason about energy consumption in SPLs based on conditional compilation.

Such techniques allow software developers to identify (non) green products and/or features in a SPL [13].

Android is a widely used ecosystem for non-wired devices, and software energy analysis and optimization is an active area of research. The GSL team has developed several techniques [14,15] and tools to analyze and optimize energy consumption in the source code of Android applications [16, 17].

Nowadays, most of the data stored in our mobile devices (files, photos, videos) is also stored in the cloud provided by the ecosystem of the device's operating system. Such cloud systems are data centers that daily run a large amount of data querying processes, monitored and controlled by highly sophisticated database management systems, which are responsible to establish efficient query processing plans to support them. Database systems usually rely on plans that optimize response time. We designed and developed an alternative method to define energy consumption plans for database queries [18, 19]. Our first experimental results show that the use of optimization heuristics allows for significant gains, both in terms of energy consumption and the time spent with the execution of queries.

CONCLUSIONS

This technical report described the research developed at the Green Software Laboratory, namely a green ranking of programming languages and data structures, techniques to detect energy inefficiency in a software system's source code, and an energy-aware query execution plan for database systems.

References

[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

TEACHING TASK ORIENTED PROGRAMMING

On the Composability, Comprehensibility, Correctness winter school there will be two tutorials about Task Oriented Programming and concrete systems based on this paradigm. The iTask system offers a web-based interface for humans to see their tasks and share their progress with these tasks. The mTask system applies the same concepts to specify the tasks executed by microprocessors. In this contribution, we justify the decisions made at the teacher training in Amsterdam about what and how these topics will be presented at the winter school. Due to the limited time and diverse background of the audience we will focus on the practical usage of the paradigm. There is barely time to address the challenges and beauty of the construction of these systems.

INTRODUCTION

Task Oriented Programming, TOP, is a style of programming centered around the concept of tasks as executed by humans and machines. These tasks are specified by ordinary functions in a functional programming language. In all our examples we will use Clean [6]. The semantics of the tasks is quite different from plain function evaluation. A task is evaluated over and over again until it produces a stable value, or its result is not needed anymore. Intermediate task results can be observed by other tasks. Tasks can be composed by task combinators.

At the Composability, Comprehensibility, Correctness winter school in Kosice, there will be two sessions on TOP. These are entitled Why "Task Oriented Programming" matters and Functional Programming of Devices. Both sessions will consist of a lecture and practical work of the attendees. In this paper, we motivate the decisions about content and organization of these sessions after the discussions at the teacher training.

AUDIENCE

The Composability, Comprehensibility, Correctness winter school is for BSc, MSc, PhD students as well as for teachers. Discussions at the teacher training revealed that the experience in functional programming is diverse, both in the amount of experience as language-wise. The languages used range from pure and lazy languages like Haskell and Clean to Erlang, Scheme and Scala.

This implies that we cannot assume a solid common functional programming experience. Only a part of the audience will be familiar with concepts like strong typing, higher order functions, type constructor classes, Monads and generics. Although these topics are the building blocks of TOP, we cannot assume that they are known by all participants.

TASK ORIENTED PROGRAMMING

Task Oriented Programming is based on a small number of domain specific prim- itive tasks. These primitive tasks typically interact with the environment, e.g., humans executing part of the tasks, or hardware interacting with the physical world. Task combinators are used to compose task from smaller tasks.

Tasks can communicate via their results as well as via Shared Data Sources, SDSs. Such an SDS contains typed data that can be accessed via primitives like get and set. These primitives act on the task state to ensure referential transparency.

In order to reuse datatypes and computations of an existing language, a TOP system is often constructed as a Domain Specific Language, DSL, embedded in an existing (functional) programming language.

The iTask System

The iTask system was the first implementation of TOP [5]. It is a DSL embedded in the functional programming language Clean. The iTask system facilitates in- teraction with human worker by the type driven generation of web-pages. These pages are displayed in one of the existing browsers. The page provides informa- tion of the current tasks for a user. The user can interact with the iTask system by filling out forms and pushing buttons.

USED TECHNIQUES

The iTask system passes a state around very similar to a state monad. The operators to return, bind (>>=), and sequence (>>|) are very similar to the well known monadic versions [4,7]. This requires higher-order functions and user- defined infix operators. To reuse the operator symbol for different Monads they are defined as type constructor classes.

The task combinators are all higher order functions, typically user-defined infix operators, manipulating task results and the global task state. Specific for TOP is that the tasks produce intermediate results that can be observed while the tasks are repeated over and over again until they produce a stable result or their result is no longer used. This requires higher order functions, laziness and automatic garbage collection.

The iTask system generates a web-server that is used by the human workers to find their task. Like all web-servers, this requires serialization and deserialization of the state to store and retrieve the current state. By filling out web-forms for arbitrary algebraic data types users indicate their progress with the tasks. All of these features are implemented using generic programming.

To implement the tasks monitoring the value of an SDS efficiently there is a hidden publish-subscribe system for each SDS that activates tasks using this SDS when its value is updated.

Apart from these properties, the iTask system uses many additional techniques. For instance the execution of task parts in an interpreter running in the browser to ensure a fast response of the system for highly interactive tasks like a drawing.

TEACHING AT THE WINTER SCHOOL

Any teaching of programming requires practical programming experience using the educated techniques to master them. This holds also for TOP. As a consequence, we divide the four hours available for Why ”Task Oriented Programming” matters into two parts of nearly identical size.

In the first part, we will outline the concept of TOP using the iTask system. Given the background of the majority of the students, we have to skip nearly all details about the implementation of the system and we have to focus on the use of the library. This library is actually a shallow embedded DSL for TOP. In the lecture, we use this a set of primitives without spending much time to explain its architecture and implementation.

For the practical work, we will split the existing basic example project into a set of small independent TOP projects. The assignments will consist of small variations of these projects to experience the flavor of task oriented programming.

The more experienced functional programmers can skip most of the basic exercises and jump directly to more advanced assignment. This way, we will be able to adapt to the individual skills of each participant.

THE MTASK SYSTEM

Microprocessors are computer systems with very limited computing capabilities. They have typically a rather low clock rate and severe memory restrictions, like a few KB of memory to store the data of a running program. These cheap processors are the driving force of many elements in the Internet of Things, IoT. In such microprocessor systems one typically has to monitor several input ports as well as to control some outputs based on these observations. Due to the hardware restrictions, there is typically no operating system offering support.

The TOP paradigm provides lightweight threads that are very well suited to monitor and coordinate the progress of such well defined simple tasks. These tasks can run at their own speed while combinators and shared data sources are used to coordinate them. Running the iTask system on the IoT devices would enable us to construct programs that are partially executed on a web-server as well as on the IoT devices. The limitations of microprocessors make it impossible to run a full-fledged iTask program on IoT devices.

To approximate the ideal solution we have developed the mTask system [3, 2]. This is a multiview shallowly embedded DSL that can be used as part of the iTask system. It supports the TOP paradigm including the same task results as the iTask system, task combinators and shared data sources.

By construction this DSL has no higher order functions and no recursive datatypes. Due to the restrictions imposed in this DSL, mTask programs can be compiled to code that executes on microprocessors. Despite these restrictions, the DSL is very suitable to specify the tasks to be executed on IoT devices easily and very concisely.

USED TECHNIQUES

The mTask system is a multiview shallowly embedded DSL based on type con- structor classes [1]. Each instance of these classes defines an interpretation, called a view, of a program constructed from these primitives. Typical views implement pretty printing, code generation for microprocessors, and simulation of the mTask programs as an iTask program.

The DSL is extendable by construction in order to reuse existing libraries for peripherals like temperature sensors, displays and servo motors. It is simple to add such a library as a language primitive to the mTask system by introducing a new type constructor class and the required instances.

To make the mTask implementation portable to many different microproces- sors and to make the reuse of existing C++ libraries easily, the code generation view produces C++ code for the Arduino platform instead of native machine code for some specific processor. The avr-gcc compiler inside the Arduino plat- form can translate the generated C++ code and the libraries used to native code for many different microprocessors.

TEACHING AT THE WINTER SCHOOL

Being able to execute high-level TOP programs on a tiny microprocessor inter- acting with peripherals is appealing for a tutorial in the winter school. However, executing a mTask program on an actual microprocessor requires much of the students; they must compose an mTask program inside the iTask system, execute the iTask program to obtain C++ code, feed this C++ code to the Arduino IDE, connect the Arduino IDE to the microprocessor and select the right options, up- load the compiled program to the microprocessor, and finally run it. All of these steps are quite easy, but the entire process produces only a result when every step is done correctly.

Since the generated program will run on a microprocessor without operating system and very limited input/output peripherals debugging such a program is challenging. After ample discussion, this sequence of steps was deemed to be too ambitious for the given time and audience.

Fortunately, the simulator view of the mTask system offers an alternative that is much easier to use. This view transforms the mTask program to an ordinary iTask program. The simulator offers a step-by-step execution of the mTask pro- gram. It displays a trace of the last step of the last executed task and the state of all peripherals and shared data sources. The clock, the value of the SDSs, as well as the state of the peripherals can be changed interactively to control the execution and investigate various scenarios. This makes the practical work of this tutorial a direct successor of the practical work of the previous iTask tutorial.

It was decided to schedule these tutorials on one day with the iTask lecture and associated practical work in the morning and the mTask tutorial in the afternoon. This way, the mTask session can directly build on the knowledge and skills gained in the iTask session. The understanding of TOP that stated in the morning will be deepened in the afternoon.

CONCLUSION

For both tutorials on TOP there are many more interesting topics than can be covered in the given time for the audience of the winter school. In the sessions, we will focus on understanding and using the TOP paradigm by relative simple examples. Slightly advanced examples will be used to illustrate the capabilities of this approach. In the practical work, we will focus on illustrative exercises that are mostly variants of examples used in the tutorial. For the advanced participants there will be some challenging assignments as well as the opportunity to discuss aspects of the systems in depth.

References

[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

AN INTERACTIVE APPROACH TO COLOURED PETRI NETS TEACHING

Formal methods belong to the techniques that, when used appropriately, can significantly contribute to the correctness of a software or hardware system under development. One of the suitable methods for systems with concurrent or non-deterministic behaviour is the Coloured Petri Nets modelling language. In this paper a teaching activity aimed at an explanation of the basic principles of the language and some of its functional programming-related features is described. The activity duration has been two and half hours and it involved an interactive model building with active audience participation.

INTRODUCTION

Considering the increasing dependency of the contemporary human society on computer systems, their correctness should be of utmost importance. And one of the approaches that can significantly contribute to the correctness is a uti- lization of formal methods during the software and hardware development. A formal method is a mathematically-based technique, which provides a formal language with unambiguously defined syntax and semantics and an apparatus, which allows performing verification, development and simulation tasks with sys- tem specifications, written in the language. One of the significant members of the formal methods family is the Coloured Petri Nets (CPN) modelling language. CPN [4, 3] combine the Petri nets formalism [1] with a functional language to handle data manipulation and decision procedures. The functional language is called CPN ML and it is a slightly modified version of Standard ML [2, 5]. The CPN language and corresponding specification, verification and simulation tasks are supported by the CPN Tools [6] software.

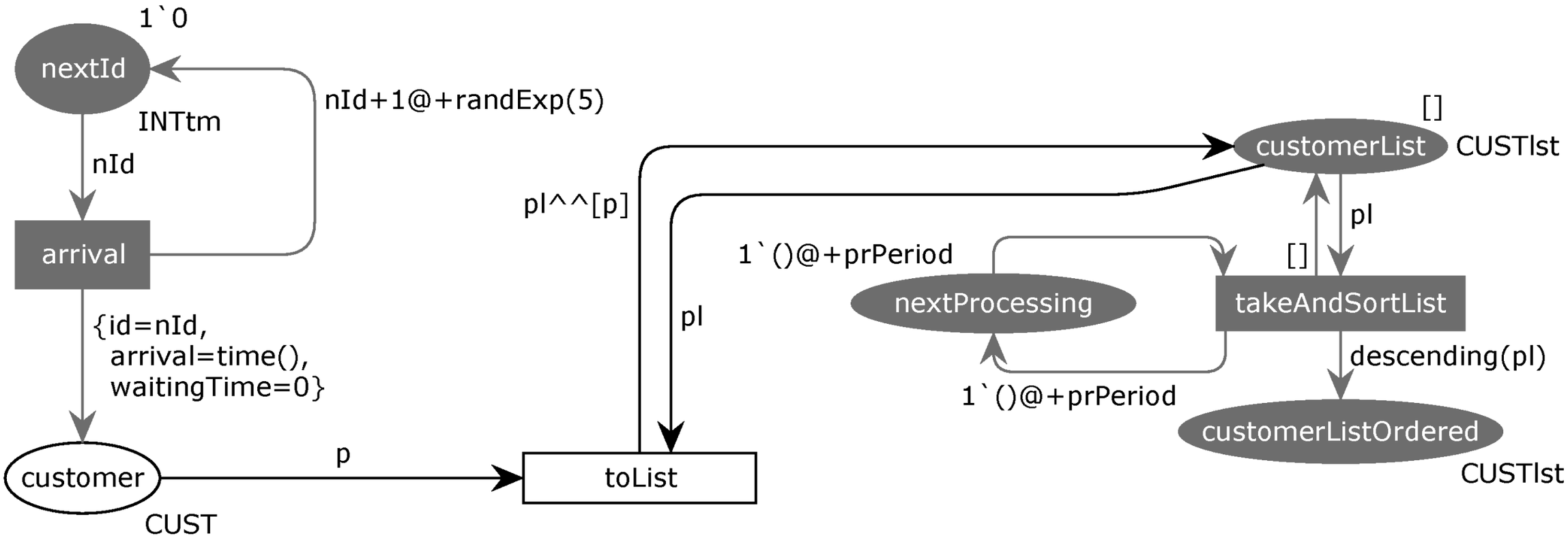
For more than a decade, CPN are a part of undergraduate courses related to formal methods, modeling and simulation at the home institution of the author. One of the methods, applied by the author when explaining CPN concepts is an interactive approach with an active audience participation. Here, the audience picks out the domain and process for which a CPN model will be designed and helps to create its selected parts. The experience from a particular implementation of this approach in a training activity for university teachers is described in the rest of this paper.

TRAINING ACTIVITY WITH INTERACTIVE CPN MODEL CREATION

The training activity was organized for about 10 participants, who were university teachers with certain functional languages background. The participants had limited to no previous knowledge of CPN. The total duration of the activity was about 2.5 hours, excluding breaks, and it was split into three phases.

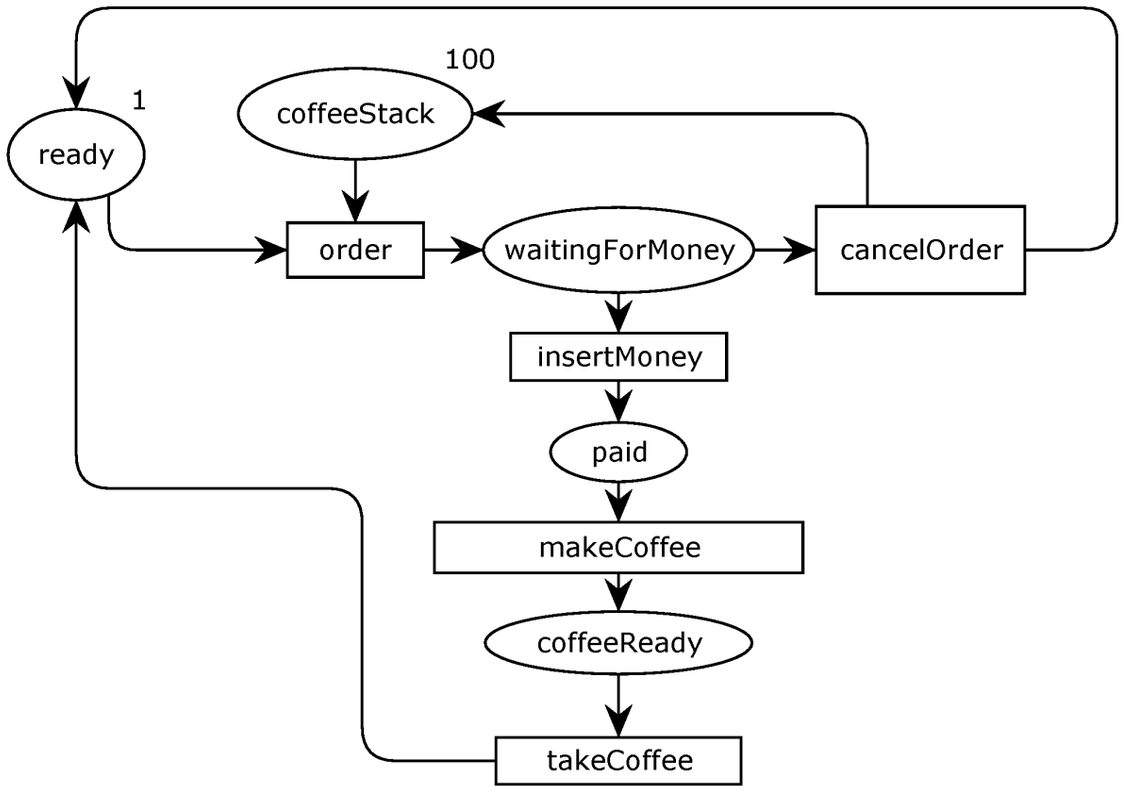
The first phase took about 30 minutes and explained the basic principles of CPN. Namely, that CPN have a graphical form, a bipartite graph with two types of vertices: places, drawn as ellipses and transitions, drawn as rectangles. The places hold tokens, which represent a state of the net and the transitions can be understood as events, which change the state by consuming existing tokens and creating new ones.

The second and third phase were devoted to a creation of a CPN model. As one of the goals of the activity was to show how some more advanced Stan- dard ML concepts, namely structures and functors, can be used in CPN models, the participants had been given a starter CPN model, which already used the concepts, before the second phase begun. The starter model is shown in Fig. 1. The part consisting of the nodes nextId, arrival and customer represents an arrival of customers, which arrive one by one to be served. The serving itself is not presented in the starter model. Instead, there is the transition toList, which takes a token from customer and adds its value to a list, held in the place customerList. The transition takeAndSortList is fired at regular inter- vals, defined by the value prPeriod.



Each firing of takeAndSortList empties the list in customerList, sorts its content and stores the ordered version in customerListOrdered. The place nextProcessing is auxiliary and ensures that takeAndSortList is fired only at the regular intervals. The sorting is provided by a function called descending, which implements the Quicksort algorithm. The function utilizes Standard ML structures and functors.

For the serving part, the activity participants decided to model a coffee vend- ing machine. During the second phase they participated on a creation of a CPN model

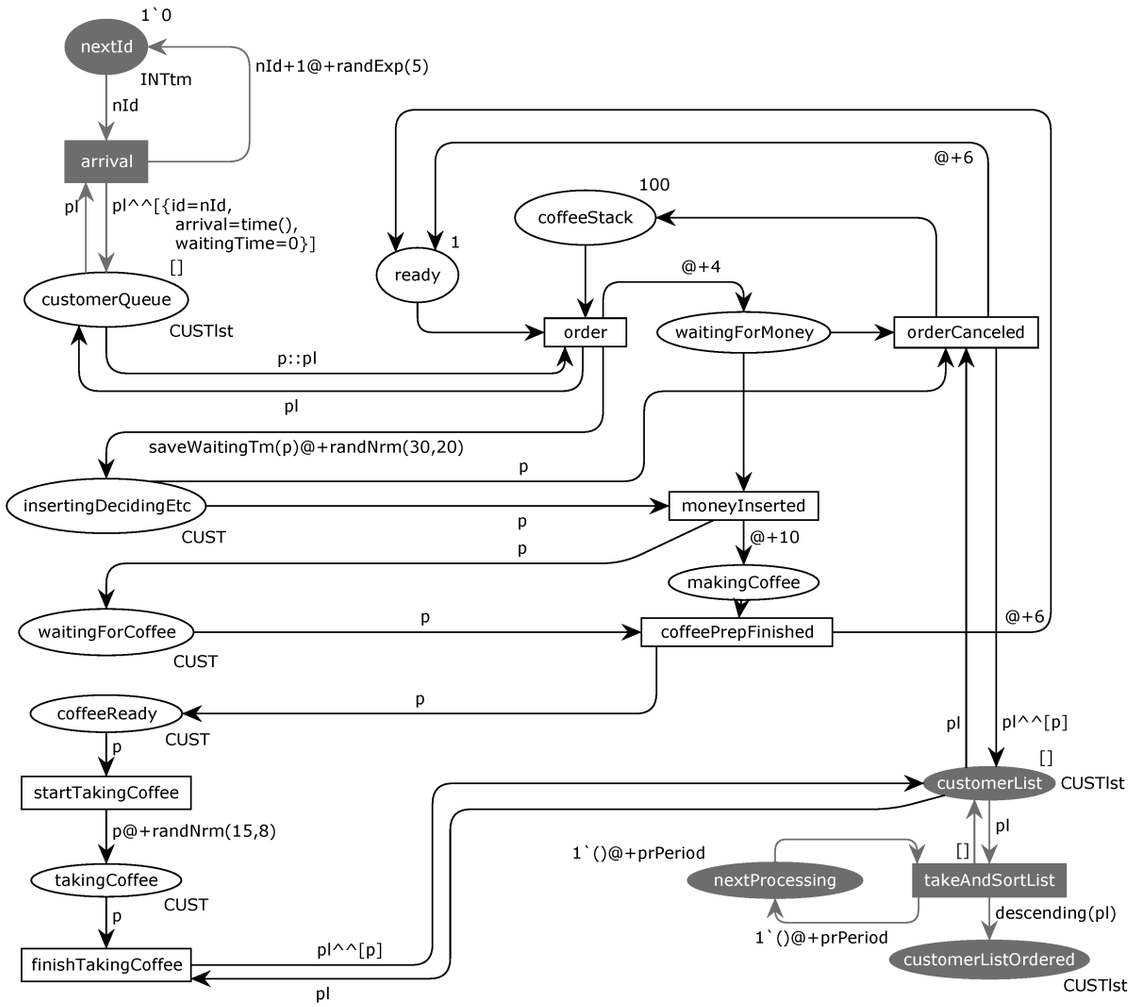


that captures the basic operation of the machine. The model is shown in Fig. 2. In its initial state the machine is ready to serve a customer (one token in the place ready) and is filled with 100 coffee doses (100 tokens in coffeeStack). The serving starts with a customer ordering a coffee by a firing of the transi- tion order. Then the machine waits for the customer's next step (a token in waitingForMoney). The customer can insert money (by firing insertMoney) or cancel the order (by firing cancelOrder). The cancellation returns the machine to the "ready" state. If the money is inserted, the machine prepares the coffee (by firing makeCoffee). Finally, by a firing of takeCoffee, the customer takes the prepared coffee and the machine returns to the "ready" state.

After the second phase there was about 70 minutes long break. During the break the lecturer connected the model from the phase 2 to the parts of the starter model and added vertices and arcs describing the customer behaviour. He also corrected some inconsistencies in the model, pointed out by one of the participants. The resulting, final, CPN model can be seen in Fig. 3. The vertices taken from the starter model (Fig. 1) without any change are rendered in grey.

The place customer is replaced by customerQueue, which holds a token with a list of values, representing a queue of customers waiting for the machine. Instead of the transition toList there is a serving part, created from the result of the second phase (Fig. 2). The serving part of the final model differs from Fig. 2 in three key aspects:

– The tokens carry information about the customer being served and arc expressions define durations of corresponding actions.



– Inconsistencies regarding the role of places and transitions are corrected. Now, all the transitions represent instantaneous events. For example, the transition makeCoffee from Fig. 2 is replaced by the place makingCoffee and the transition takeCoffee is replaced by the vertices startTakingCoffee, takingCoffee and finishTakingCoffee.

– Actions and states of the customers and the machine are modelled separately. The vertices customerQueue, insertingDecidingEtc, waitingForCoffee, coffeeReady, startTakingCoffee, takingCoffee and finishTakingCoffee belong to the customers while the rest of the serving part represents the machine or both parties.

The third phase of the training activity has been devoted to the explanation of the final model and a discussion about the place of such models in the development of correct computer systems. It took about 30 minutes.

CONCLUSION

The interactive training activity, presented here, is suitable for short, intensive courses, which often take place during summer schools or other similar teaching events. The described test run of the activity revealed that the original time donation, which was 2 hours, was not sufficient. Therefore the third phase has been needed, where the lecturer presented the final model. Considering the time needed to construct the final model by the lecturer, it will require another at least two hours to perform the whole model creation process interactively with the auditory. All the CPN models presented or mentioned here can be obtained by request from the author.

References

[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: AN EXTENSIBLE CODE COMPREHENSION FRAMEWORK

CodeCompass is an open source tool to help understanding large legacy software systems. Based on the LLVM/Clang compiler infrastructure, CodeCompass gives exact information on complex C/C++ language elements. The wide range of interactive visualizations includes class and function call diagrams; architectural, component and interface diagrams and “points to” diagrams and many others. CodeCompass also utilizes build information to explore the system architecture as well as version control information when available. Clang based static analysis results are also integrated. Although the tool focuses mainly on C and C++, it also supports Java and Python languages. Having a web-based, pluginable, extensible architecture, the CodeCompass framework can be an open platform to further code comprehension, static analysis and software metrics efforts.

INTRODUCTION

Bug fixing or new feature development requires a confident understanding of all details and consequences of the planned changes. Code comprehension tools can help to reveal the original intentions and implementation details by building a model from the source code and other available information. Although a number of such tools are available either as proprietary or free software, their feature set is limited.

CodeCompass was developed to eliminate these restrictions.

The CodeCom- pass project is a joint open source effort of Ericsson Ltd. and the Eötvös Loránd University, Budapest to help understanding large software systems. To provide exact information on complex C/C++ language elements like overloading, inheritance, the usage of variables and types, possible uses of function pointers and the virtual functions – features that various existing tools support only partially – CodeCompass is based on a real compiler, the LLVM/Clang infrastructure. Thus, it eliminates the weaknesses of the usual “light-weight” comprehension tools, like OpenGrok.

CodeCompass, however, is not restricted to the source code. It uses the build information of the system to reveal architectural connections. It also employs the version control information if available, so one can identify connections be- tween different source files “accidentally” modified in the same commit. To help fast and precise perception CodeCompass uses both textual and graphical representation of the software system to comprehend. A number of (interactive) diagrams are accessible from the usual function call graphs to the unique architectural diagrams. To provide easy access for the users, CodeCompass has a web-based architecture. The client can be a standard web browser, an editor plug-in or any 3rd party application. The communication is based on a REST API and scales well for parallel client requests.

In this paper we will compare CodeCompass to existing comprehension tools and describe its feature set. In Section 2 we overview the main archetypes of existing tools for code comprehension. We introduce the extendible architecture of CodeCompass in 3. The main features of the tool are discussed in Section 4. We summarize the paper in Section 5.

RELATED WORK

On the software market there are several tools which aim some kind of source code comprehension. Some of them uses static analysis, others examine also the dynamic behavior of the parsed program. These tools can be divided into dif- ferent archetypes based on their architectures and their main principles. On the one hand tools are having server-client architecture. Generally these tools parse the project and store all necessary information in a database. The (usually web- based) clients are served from the database. These tools can be integrated into the workflow as nightly CI runs.

This way the developers can always browse and analyze the whole, large, legacy codebase. Also there are client-heavy applications where smaller part of the code base is parsed. This is the use case for IDE editors where the frequent modification of the source requires quick update of the database about analyzed results. In this section we present some tools used in industrial environment from each categories.

Woboq [3] is a web-based code browser for C and C++. This tool has extensive features which aim for fast browsing of a software project. The user can quickly find the files and named entities by a search field which provides code completion for easy usability. The navigation in the code base is enabled through a web page consisting of static HTML files. These files are generated during a parsing process. The advantage of this approach is that the web client will be fast since no on-the-fly computation is needed on the server side while browsing.

Hovering the mouse on a specific function, class, variable, macro, etc. can show the properties of that element. For example, in case of functions one can see its signature, place of its definition and place of usages. For classes one can check the size of its objects, the class layout and offset of its members and the inheritance diagram. For variables one can inspect their type and locations where they are written or read.

In C and C++ macros form a sublanguage which is evaluated in a precompilation step. This evaluation is a textual substitution of macro tokens which means that the compilation phase works with another code than the original one. In Woboq, the final value of macro expansions can also be inspected.

A very handy feature of the tool is the semantic highlighting. By this feature the different language elements can easily be distinguished: the formatting of local, global or member variables, virtual functions, types, typedefs, classes, macros, etc. are all different.

Woboq can provide the aforementioned features because the information needed is collected in a real compilation phase. The examined project first has to be compiled and parsed by Woboq. The parsing is done by LLVM/Clang infrastructure which makes the whole abstract syntax tree available. This way all pieces of semantic information can be extracted with the same semantics the final program is to have. This also gives a disadvantage of the tool, namely Woboq can only be used for browsing C and C++ projects.

OpenGrok [4] is a fast source code search and cross reference engine. Op- posed to Woboq, this tool doesn't perform deep language analysis, therefore it is not able to provide semantic information about the particular entities. Instead, it uses Ctags [5] for parsing the source code only textually, and to determine the type of the specific elements. Simple syntactic analysis enables the distinguishing of function, variable or class names, etc. The search among these is highly optimized, and therefore very fast even on large code bases. The search can be accomplished via compound expressions (e.g. defs:target), containing even wild cards, furthermore, results can be restricted to subdirectories. In addition to text search there is opportunity to find symbols or definitions separately. The lack of semantic analysis allows Ctags to support several (41) programming languages. Also an advantage of this approach is that it is possible to incrementally update the index database. OpenGrok also gives opportunity to gather information from version control systems like Mercurial, SVN, CSV, etc.

Understand [6] is not only a code browsing tool, but a a complete IDE. Its great advantage is that the source code can be edited and the changes of the analysis can be seen immediately.

Besides code browsing functions already mentioned for previous tools, Understand provides a lot of metrics and reports. Some of these are the lines of code (total/average/maximum globally or per class), number of coupled/base/derived classes, lack of cohesion [2], McCabe complexity [1] and many others. Treemap is a common representation method for all metrics. It is a nested rectangular view where nesting represents the hierarchy of elements, and the color and size dimensions represent the metric chosen by the user.

For large code bases, the inspection of the architecture is necessary. Under- stand can show dependency diagrams based on various relations such as function call hierarchy, class inheritance, file dependency, file inclusion/import. The users can also create their custom diagram type via the API provided by the tool.

In programming, the core concepts are common across languages, but there are some concepts which are interpreted differently in a particular language. Understand can handle ~ 15 languages and can provide language specific information about the code e.g. function pointer analysis in C/C++ or package hierarchy diagrams in Ada.

Understand builds a database from the code base. All information can be gathered via a programmable API. This way the user can query all the necessary information which are not included in the user interface.

CodeSurfer [7] is similar to Understand in the sense that it is also a thick client, static analysis application. Its target is understanding C/C++ or x86 machine code projects. CodeSurfer accomplishes deep language analysis which provides detailed information about the software behavior. For example, it im- plements pointer analysis to check which pointers may point to a given variable, lists the statements which depend on a selected statement by impact analysis, and uses data flow analysis to pinpoint where a variable was assigned its value, etc.

THE CODECOMPASS ARCHITECTURE

In the previous section we have listed some aspects concerning the goals and architectures of code comprehension tools. Now we present where CodeCompass stands among these tools.

CodeCompass has a client-server architecture in which it presents the information gathered in a preceding parsing phase. The reason why this architecture was chosen comes from the goal of the tool. As opposed to code editors, Code- Compass has been planned to be a code comprehension tool. There are fundamental differences between these two use-cases. During code writing, programmers are manipulating only a few files at the same time. In code comprehension, however, it is needed to consider the sources of multiple modules through the code base. In editors code completion is one of the most useful features: the programmer doesn’t want to remember all methods and fields of a class, but requires the editor to list these. In code comprehension the wide range of visualizations is needed in order to overview the relations of code parts. While editing the source, the programmer focuses only to a relatively small fragment of the code, like a function or a class. In code comprehension it is not only the low-level behavior of the functions, but their dependencies and effects are considered in the context of high-level module system.

The main user interface of CodeCompass is web-based. All the aforementioned visualizations and functionalities can be queried via a public API which is assigned to a server application. The web interface handles the use-cases that aim fast and handy browsing, inspection and comprehension tasks. However, CodeCompass is more than just a code browsing tool. It is also a framework, i.e. an extensible collector and presenter of static analysis processes. That is why the intention was not to create a client-heavy application which stores the analysis results on the client side, but being able to serve the various needs of users. This way it is possible to implement a script for example which collects the set of functions that form a closure by function call relation, thus specifying a coherent slice of the software.

Another design requirement of CodeCompass was to handle large-scale code bases and still answering user requests very fast, i.e. in terms of seconds at most. This is accomplished by storing all the least amount of information in a database which are sufficient to answer the requests. Since we intended to give precise results for the queries, a preceding parsing process is required. In the first we stored the whole abstract syntax tree of the source, but this resulted a 1:1000 ratio between the source code and the database size. However, it turned out that most cases the users are interested in named entities only (function, variables, classes, macros, etc.), so it was unnecessary to store anything else, such as control structures or other statements. Nonetheless, there are some tasks which require more than the stored information, like a slicing algorithm. If the user wants to see the effects of changing the value of a variable when state modifying statements have to be taken into account too. This requires the reparsing of the code on the fly.

CODECOMPASS FEATURES

In this section we will give an overview about the features available through the standard GUI. When describing language specific features, such as listing callers of a method, we will always assume the project's language to be C++ as that has the most advanced support in CodeCompass, but similar features are available for Java and Python.

Search

Probably the most fundamental use-case of a code comprehension tool is search- ing. One may search either for a file or source code. For finding source code elements the tool provides 3 different search possibilities:

In full text search mode the search phrase is a group of words such as “returns an astnode*”. A query phrase matches a text block, if the searched words are next to each other in the source code in that particular order. Wildcards, such as *, or ? can be used, matching any multiple or single character. Logical operators such as AND, OR, NOT can be used to join multiple query phrases at the same time.

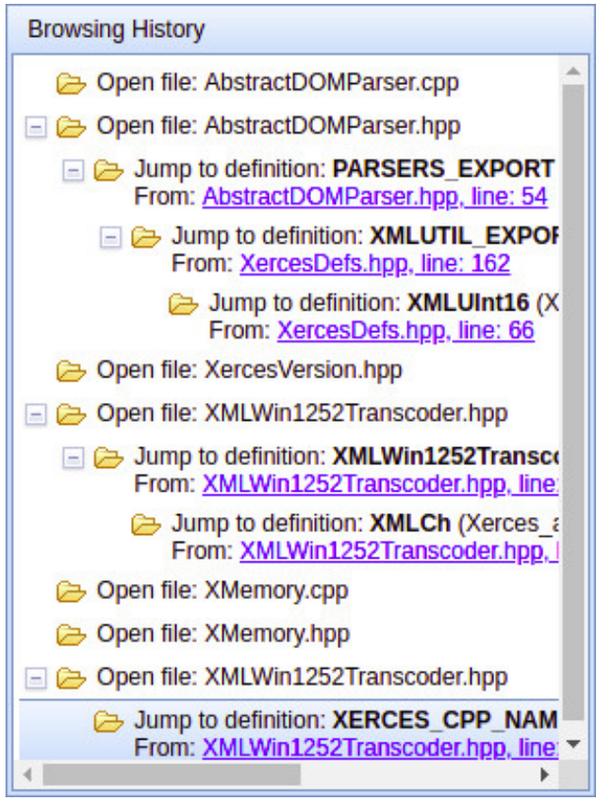
On a higher level it is possible to find symbols in source codes by definition search. Here we are using CTags for indexing the code base thus being able to find variables, functions, classes, macros, etc. It is important to know that this language entity search has nothing to do with deep language parsing.

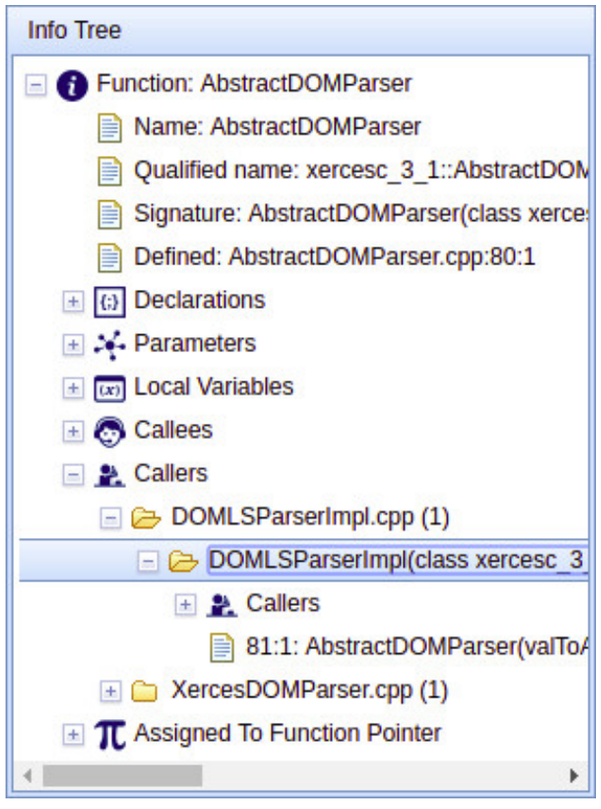
While debugging a program, sometimes the only information to start with is an output message in the console log emitted by our software. This is the only trace where one may start, e.g. "DEBUG INFO: TSTHan: sys_offset=-0.019821, drift_comp=-90.4996, sys_poll=5". Note that such a message can contains timestamps or other dynamically generated fragments, so it is impossible to find this message as a direct string. However, in CodeCompass a fuzzy search can be done by log search.

Information about language symbols

When the element has been found, the next step is gathering information about it. The user can choose “Info tree” from the pop-up menu after selecting a named entity. This tree contains all information that is provided by a language parser. In case of C/C++ we are using the LLVM/ Clang compiler in order to fetch information about the symbols.

For functions we can check their parameters, local variables, callers and callees. An interesting feature of the tree is that the callers are presented re- cursively i.e. the children of a node are the callers of a function. Their children nodes are the callers of these functions, and this goes on recursively, theoretically back to the main function.





However, function calls are not always direct, but can happen via function pointers. Even though this is a dynamic behavior, CodeCompass summons all the occurrences where a function was assigned to a function pointer and the invocation happens through this pointer.

In case of classes the collected information are the aliases (by typedef the class can have a synonym), inheritance relations (grouped by visibility), friends, methods/fields (direct or inherited) and usages (as local/global variable, function parameter/return type or field of another class).

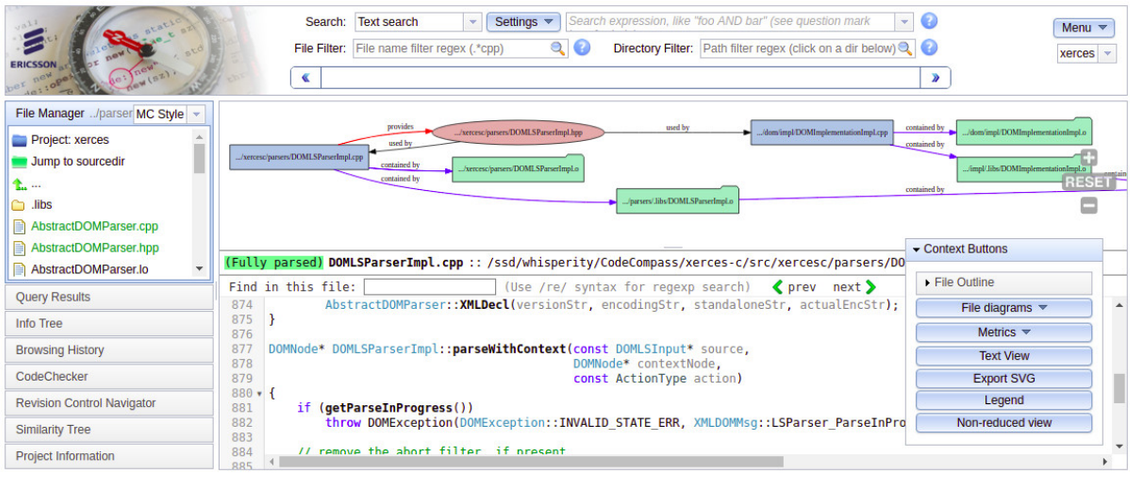
For variables it is useful to know the places in the code where it was written and read. For enumeration types the enumeration constants are listed with their integer values.

Diagrams

Visualizations are one of the most helpful representations for humans to overview a system. CodeCompass presents several symbol and file based diagrams. These diagrams are graph-based, i.e. they represent entities and their connections. These are also interactive diagrams: hovering the mouse over the nodes the represented entity is displayed in the text view.

By clicking them the selected entity becomes the center node showing its relations according to the diagram type.

Function call diagram shows all callers and callees of a function in a graph. UML class inheritance diagram shows the full inheritance chain up until the root base class and recursively for all derived classes. We have also implemented a pointer analysis diagram which shows the allocated objects and the pointers which possibly point to them. Of course this is a dynamic information which can only partly be collected in a static analysis.



An interface diagram called for a C/C++ source file shows which headers are “used only” or “implemented” by the given file. Usage means that a source file uses another file if there is a symbol usage in it which is declared in the other file. Implementation relationship means that a symbol is declared in a file (thus forming an interface) and defined in an other. These relations are also applicable for directories considering the contained files. In case of a compiled language there are also the output files like objects and executables. Based on linkage information we can present which sources make a binary file up.

CodeBites provides a different visualization of the inspected source code. In this view the nodes of the graph are the definitions of specific named symbols, like classes, functions, etc. The idea is that a programmer would like to discover this entity by understanding its behavior but without loosing the focus. So the parts of the code text in a node are clickable which triggers the addition of the selected element’s definition.

Version control visualizations

Visualization of version control information is an important aid to understand software evolution. Git blame view shows line-by-line the changes (commits) to a given file. Changes that happened recently are colored lighter green, while older changes are darker red. This view is excellent to review why certain lines were added to a source file. CodeCompass can also show Git commits in a filterable list ordered by the time of commit. This search facility can be used to list changes made by a person or to filter commits by relevant words in the commit message.

Metrics

CodeCompass can show the McCabe Cyclomatic Complexity [1], the lines of code and the number of bugs found by Clang Static Analyzer metrics for individual files and summarized over directory hierarchies. These metrics can be visualized on a tree map, where directories are indicated by boxes. The box size and its color shade is proportional to the chosen metric.

Browsing history

De Alwis and Murphy studied why programmers experience disorientation when using the Eclipse Java integrated development environment (IDE) [8]. They use visual momentum [9] technique to identify three factors that may lead to disorientation: i) the absence of connecting navigation context during program exploration, ii) thrashing between displays to view necessary pieces of code, and iii) the pursuit of sometimes unrelated subtasks. The first factor means that the programmer, during investigating a problem visits several files as follows a call chain, or explores usage of a variable. At the end of a long exploration session, it is hard to remember why the investigation ended up in a specific file. The second reason for disorientation is the frequent change of different views in Eclipse. The third contributor to the problem is that a developer, when solving a program change task, evaluates several hypotheses, which are all individual comprehension subtasks. Programmers tend to suspend a subtask (before finishing it) and switch to another. For example, the programmer investigates how a return value of a function is used, but then changes to a subtask understanding the implementation of the function itself. It was observed that, for a developer, it is hard to remind themselves about a suspended subtask [10].

CodeCompass implements a browsing history view which records (in a tree form) the path of navigation in the source code. A new subtask is represented by a new branch of the tree, while the nodes are navigation jumps in the code labeled by the connecting context (such as “jump to the definition of init”). So problem i) and ii) is addressed, by the labeled nodes in the browsing history, while problem iii) is handled by the branches assigned to subtasks.

CodeChecker - C/C++ Bug Reporting

Clang Static Analyzer implements an advanced symbolic execution engine to report programming faults. CodeCompass can visualize the bugs identified the Clang Static Analyzer and Clang Tidy by connecting it to a CodeChecker server [11]. CodeCompass shows the bug position, and the symbolic execution path that lead to a fault.

Namespace and type catalog

CodeCompass processes Doxygen documentation and stores them for the function, type, variable definitions. It also provides a type catalog view that lists types declared in the workspace organized by a hierarchical tree view of namespaces.

SUMMARY

We presented CodeCompass, a static analysis tool for comprehension of large- scale software. It was designed to avoid the various shortages of the existing comprehension tools which are either lightweight, easy to use but without the deep knowledge of a real compiler; or heavyweight, non-scalable installed on the client machine. Having a web-based, pluginable, extensible architecture, the framework can be an open platform to further code comprehension, static analysis and software metrics efforts. Initial user feedback and usage statistics suggests that the tool is useful for developers in comprehension activities and it is used besides traditional IDEs and other cross-reference tools.

References

[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.