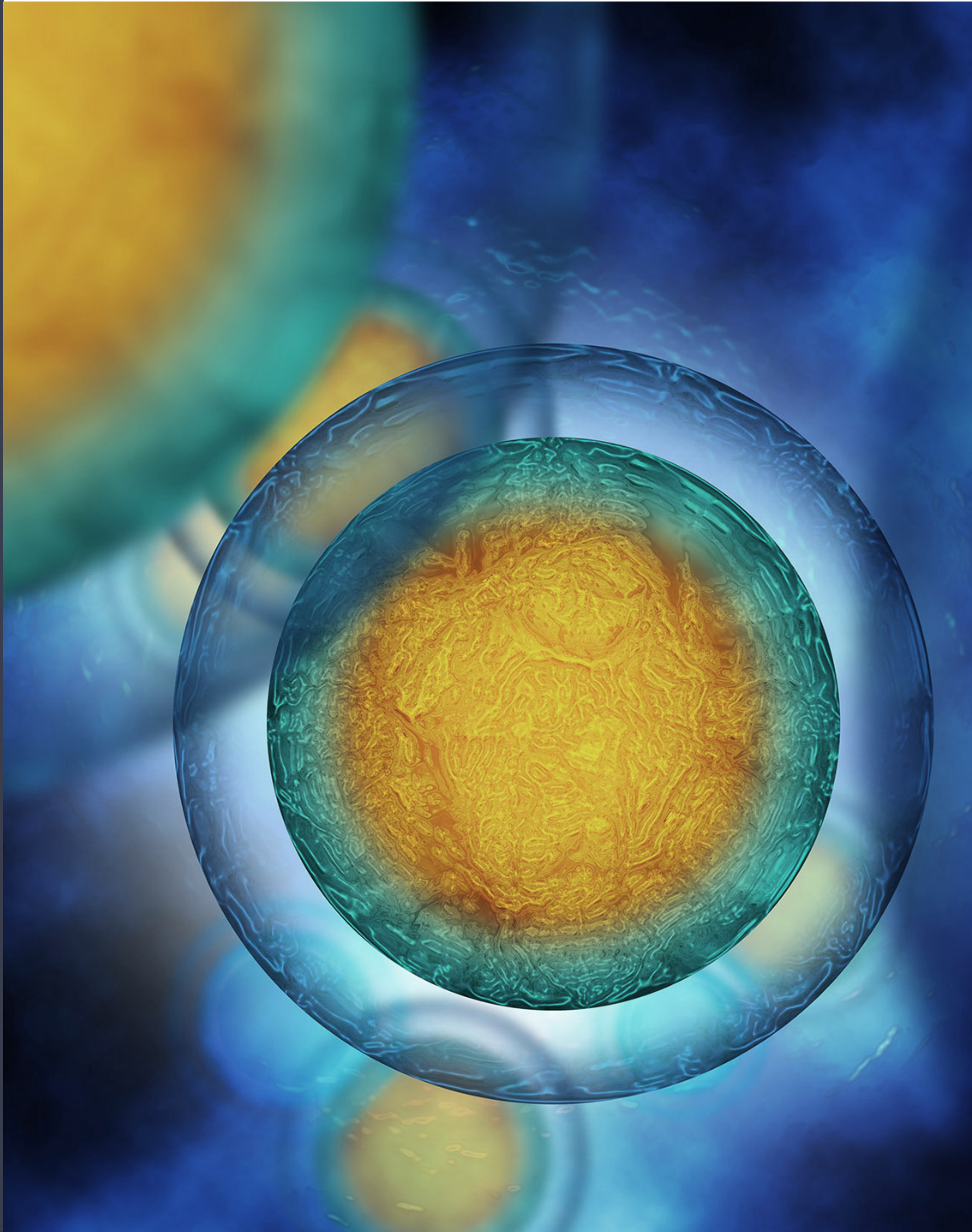
FE3CWS

CEFP 2019 SUMMER SCHOOL TEACHING MATERIAL

Intellectual output 4 of the ERASMUS+ project 2017-1-SK01-KA203-035402



Some words about the

CONTENTS

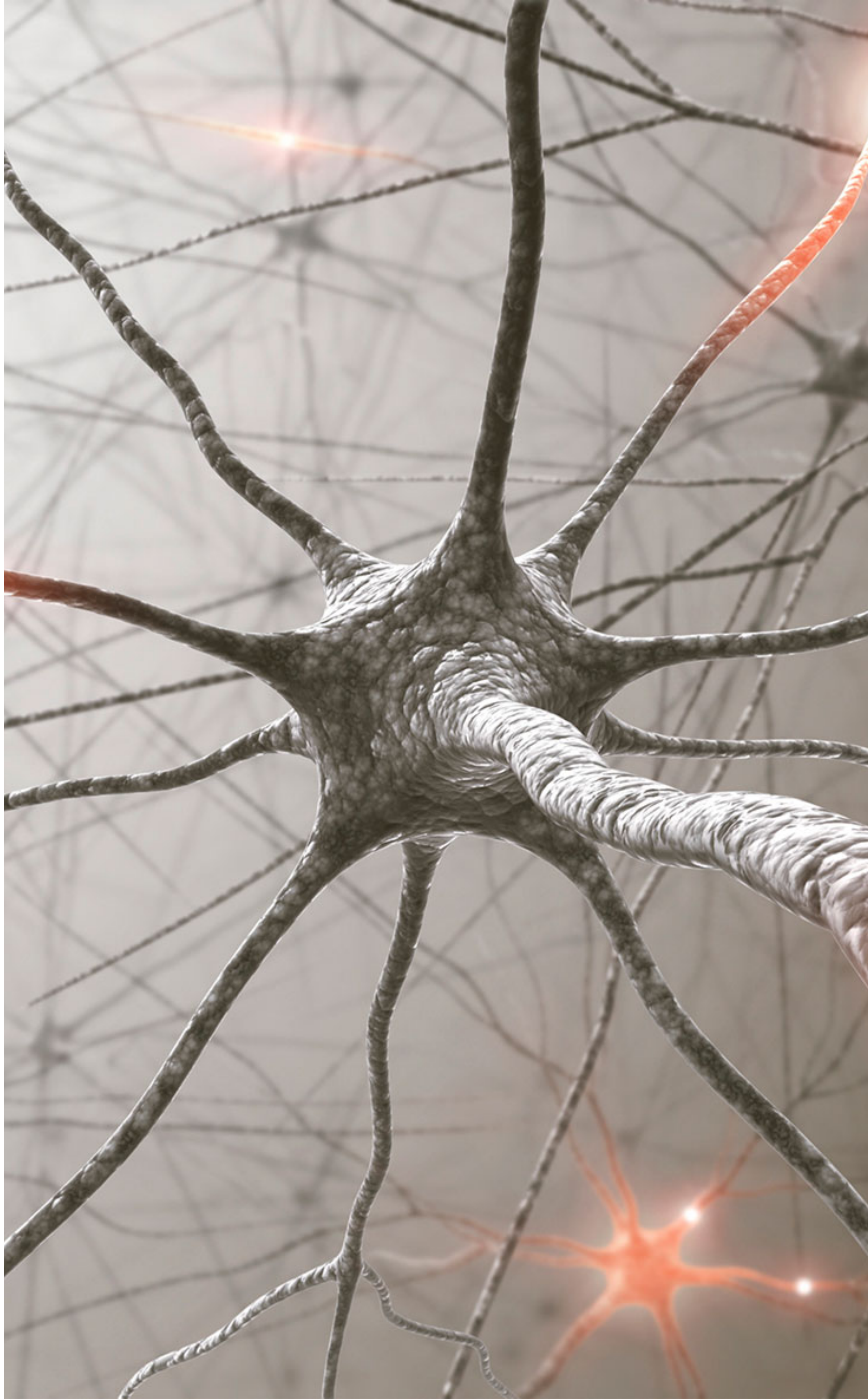
- 10 topics related to software composition, comprehension and correctness
- 20 authors from 7 European universities from Croatia, Hungary, Netherlands, Portugal and Slovakia
- Available in 7 languages: English, Hungarian, Slovak, Croatian, Romanian, Bulgarian and Portuguese

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

The Central European Functional Programming (CEFP) Summer School is the second intensive programme for higher education learners and teaching staff extending the community of the Central European Functional Programming (CEFP) summer school in the frame of the ERASMUS+ project No. 2017-1-SK01-KA203-035402 “Focusing Education on Composability, Comprehensibility and Correctness of Working Software” that was held between 17 and 21 June 2019.

The included material was created and presented in the frame of the above mentioned project. This publication is the print-formatted version of the intellectual output O4 of the project.

TABLE OF CONTENTS



VISUAL PROTOTYPING USING TASK ORIENTED PROGRAMMING

In this course we will create applications using a visual assistant for Task Oriented Programming (TOP). TOP is a novel programming paradigm developers can use to quickly prototype multi-user web applications. The central way of modelling applications in TOP is by creating Tasks. Tasks represent pieces of real world work that can be performed by people or by systems. Using a handful of operations, they can be combined into bigger and more powerful Tasks.

We will explore the basic concepts of TOP by studying some example applications, while showing how to model them using Tasks in a visual development environment. The visual environment guides developers during the modelling process. The tool only presents sane ways to create and expand Tasks, and gives hints how to solve type and scoping errors. This results in correct and compilable program code.

Students are encouraged to extend the example applications in a hands on session. Our visual approach does only require basic knowledge on programming and data types. The introduction on TOP and its modelling principles are a prerequisite on the course on mTasks.

TASK ORIENTED PROGRAMMING FOR THE INTERNET OF THINGS

The Internet of Things (IoT) consists of devices that sense, act, and communicate with other systems on the internet. Typical requirements for IoT devices are that they must be cheap and consume little energy. This is achieved by driving the IoT devices by small microprocessors with tiny amounts of memory and processing power. Most of these systems have no proper operating system and just run a specific program to execute the intended task.

This makes programming of the IoT very challenging. The single program running on such a device must interleave all subtasks, like monitoring inputs, controlling the peripherals and communication. Various devices that cooperate have to agree on the protocol used and have to solve the notorious concurrent programming problems.

In this lecture we will give a hands-on introduction to Task Oriented Programming (TOP) for the IoT. In our TOP approach the communication between devices and their servers is handled transparently by the mTask system. The entire system is programmed in a single high-level functional program. For each subtask of the system we define a corresponding mTask. These subtasks can be composed by task combinators to more powerful tasks. These tasks can inspect intermediated values of other subtasks as well as communicate with any other task in the system via Shared Data Sources (SDS). Subtasks for an IoT device are dynamically shipped to the device and interpreted there. The strong type system prevents runtime type problems. This TOP approach greatly simplifies the development of software for the IoT.

Literature

Peter Achten. Clean for Haskell98 Programmers. 13th July 2007.

Peter Achten, Pieter Koopman and Rinus Plasmeijer. 'An Introduction to Task Oriented Programming'. In: Central European Functional Programming School. Springer, 2015, pp. 187– 245.

Douglas Adams. The Hitchhiker's Guide to the Galaxy Omnibus: A Trilogy in Four Parts. Vol. 6. Pan Macmillan, 2017.

Matheus Amazonas Cabral De Andrade. 'Developing Real Life, Task Oriented Applications for the Internet of Things'. Master's Thesis. Nijmegen: Radboud University, 2018. 60 pp.

Tom Brus et al. 'Clean – a language for functional graph rewriting'. In: Conference on Functional Programming Languages and Computer Architecture. Springer, 1987, pp. 364– 384.

Jacques Carette, Oleg Kiselyov and Chung-Chieh Shan. 'Finally tagless, partially evaluated: Tagless staged interpreters for simpler typed languages'. In: Journal of Functional Programming 19.5 (Sept. 2009), p. 509. issn: 0956-7968, 1469-7653. doi: 10.1017/S0956796809007205.

James Cheney and Ralf Hinze. First-class phantom types. Cornell University, 2003.

Li Da Xu, Wu He and Shancang Li. 'Internet of things in industries: a survey'. In: Industrial Informatics, IEEE Transactions on 10.4 (2014), pp. 2233–2243.

L. M. G. Feijs. 'Multi-tasking and Arduino : why and how?' In: Design and semantics of form and movement. 8th International Conference on Design and Semantics of Form and Movement (DeSForM 2013). Ed. by L. L. Chen et al. Wuxi, China, 2013, pp. 119–127. isbn: 978-90-386-3462-3.

Patrick C. Hickey et al. 'Building embedded systems with embedded DSLs'. In: ACM Press, 2014, pp. 3–9. isbn: 978-1-4503-2873-9. doi: 10.1145/2628136.2628146.

Pieter Koopman, Mart Lubbers and Rinus Plasmeijer. 'A Task-Based DSL for Microcomputers'. In: Proceedings of the Real World Domain Specific Languages Workshop 2018 on - RWDSL2018. Vienna, Austria: ACM Press, 2018, pp. 1–11. isbn: 978-1-4503-6355-6. doi: 10.1145/3183895.3183902.

Mart Lubbers. 'Task Oriented Programming and the Internet of Things'. Master's Thesis. Nijmegen: Radboud University, 2017. 69 pp.

Mart Lubbers, Pieter Koopman and Rinus Plasmeijer. 'Multitasking on Microcontrollers using Task Oriented Programming'. In: 2019 42st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). COnference on COmposability, COmprehensibility and COrrectness of Working Software. Opatija, Croatia: IEEE, 2019, pp. 1842–1846.

Mart Lubbers, Pieter Koopman and Rinus Plasmeijer. 'Task Oriented Programming and the Internet of Things'. In: Proceedings of the 30th Symposium on the Implementation and Application of Functional Programming Languages. International Symposium on Implementation and Application of Functional Languages. Lowell, MA: ACM, 2018, p. 12. isbn: 978-1-4503-7143-8. doi: 10.1145/3310232.3310239.

Rinus Plasmeijer, Peter Achten and Pieter Koopman. 'iTasks: executable specifications of interactive work flow systems for the web'. In: ACM SIGPLAN Notices 42.9 (2007), pp. 141–152.

Rinus Plasmeijer and Pieter Koopman. 'A Shallow Embedded Type Safe Extendable DSL for the Arduino'. In: Trends in Functional Programming. Vol. 9547. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016. isbn: 978-3-319-39109-0 978-3-319-39110-6. doi: 10.1007/978-3-319-39110-6.

Camil Staps and Mart Lubbers. The Clean language search engine. 2017. url: https: //cloogle.org.

Jurrien Stutterheim, Peter Achten and Rinus Plasmeijer. 'Maintaining Separation of Concerns Through Task Oriented Software Development'. In: Trends in Functional Programming. Ed. by Meng Wang and Scott Owens. Vol. 10788. Cham: Springer International Publishing, 2018, pp. 19–38. isbn: 978-3-319-89718-9 978-3-319-89719-6. doi: 10.1007/978- 3-319-89719-6_2.

PAINT YOUR PROGRAMS GREEN - ON THE ENERGY EFFICIENCY OF DATA STRUCTURE IMPLEMENTATIONS

Energy-efficiency has been a concern for both hardware and low-level software engineers for years [1], [2], [3]. However, the growing worldwide movement towards sustainability, including sustainability in software [4], combined with the systemic nature of energy efficiency as a quality attribute have motivated the study of the energy impact of application software in execution. This tendency has led researchers to evaluate existing techniques, tools, and languages for application development from an energy-centric perspective. Recent work has studied the effect that factors such as code obfuscation [5], Android API calls [6], object-oriented code refactorings [7], constructs for concurrent execution [8], and data types [9] have on energy efficiency. Analyzing the impact of different factors on energy is important for software developers and maintainers.

Collections	Associative Collections	Sequences
EnumSet StandardSet UnbalancedSet LazyPairingHeap LeftistHeap MinHeap SkewHeap SplayHeap	AssocList PatriciaLoMap StandardMap TernaryTrie	BankersQueue SimpleQueue BinaryRandList JoinList RandList BraunSeq FingerSeq ListSeq RevSeq SizedSeq MyersStack

iters	operation	base	aux
1	add	100000	100000
1000	addAll	100000	1000
1	clear	100000	n.a.
1000	contains	100000	1
5000	containsAll	100000	1000
1	iterator	100000	n.a.
10000	remove	100000	1
10	removeAll	100000	1000
5000	toArray	100000	n.a.
10	retainAll	100000	1000

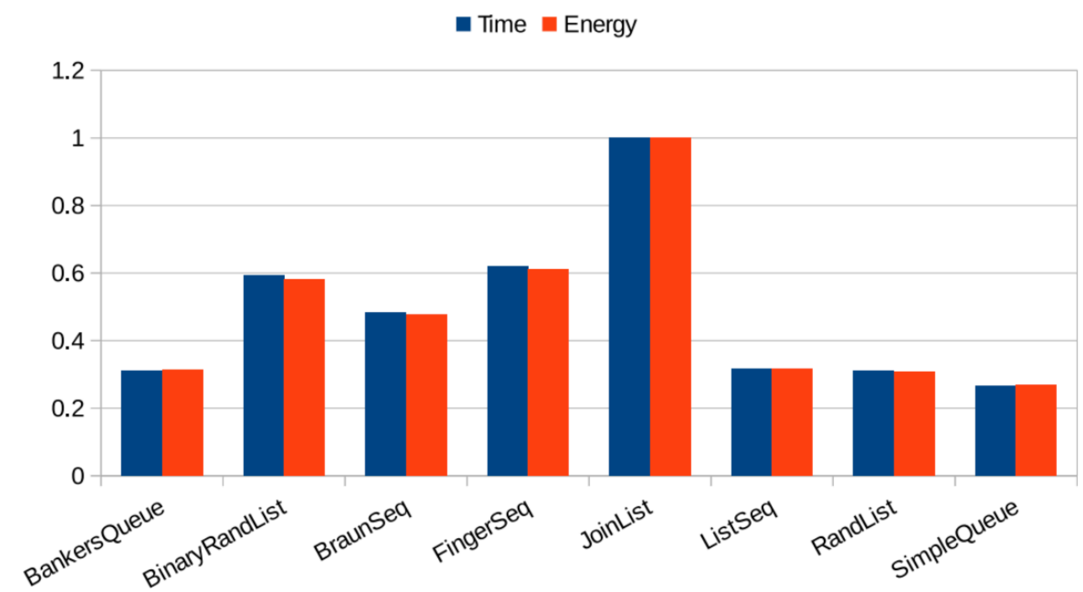
In this tutorial we analyze and compare the energy efficiency of different implementations for concrete data abstractions such as Sequences, Sets or Associative Collections. For each implementation, we inspect how operations such as adding, deleting or searching for elements handle different workloads. The subjects of our study are a functional programming language [10,11] and an object-oriented one [13,14].

In a functional setting, we have compared the implementations presented in Figure 1, namely using the operations presented in Figure 2.

In the object oriented realm, we have analyzed the implementations presented in Figure 3, namely using the operations presented in Figure 4.

Our goal is to provide developers actionable information that has already been integrated in supporting tools, and that can steer green software construction. We were able to show that the same operation made available in different implementations can differ significantly in terms of both runtime and energy consumption. As an example, in Figure 5 we depict the results of the remove operation for the Sequences abstraction implementations available in Haskell's Edison Library.

Sets	Lists	Maps
ConcurrentSkipListSet	*ArrayList*	*ConcurrentHashMap*
CopyOnWriteArraySet	*AttributeList*	*ConcurrentSkipListMap*
HashSet	*CopyOnWriteArrayList*	*HashMap*
LinkedHashSet	*LinkedList*	*Hashtable*
TreeSet	*RoleList*	*IdentityHashMap*
	RoleUnresolvedList	*LinkedHashMap*
	Stack	*Properties*
	Vector	*SimpleBindings*
		TreeMap
		UIDefaults
		WeakHashMap



Sets	Lists	Maps
add	*add*	*clear*
addAll	*addAll*	*containsKey*
clear	*add(index)*	*containsValue*
contains	*addAll(index)*	*entrySet*
containsAll	*clear*	*get*
iterateAll	*contains*	*iterateAll*
iterator	*containsAll*	*keySet*
remove	*get*	*put*
removeAll	*indexOf*	*putAll*
retainAll	*iterator*	*remove*
toArray	*lastIndexOf*	*values*
	listIterator	
	listIterator(index)	
	remove	
	removeAll	
	Continues...	

References

[1] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power cmos digital design," Solid-State Circuits, IEEE Journal of, vol. 27, no. 4, pp. 473– 484, Apr 1992.

[2] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 2, no. 4, pp. 437–445, Dec 1994.

[3] W. Yuan and K. Nahrstedt, “Energy-efficient soft real-time cpu scheduling for mobile multimedia systems,” in Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. ACM, 2003, pp. 149–163.

[4] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, M. Mahaux, B. Penzenstadler, G. Rodríguez-Navas, C. Salinesi, N. Seyff, C. C. Venters, C. Calero, S. A. Koçak, and S. Betz, “The karlskrona manifesto for sustainability design,” CoRR, vol. abs/1410.6968, 2014.

[5] C. Sahin, P. Tornquist, R. Mckenna, Z. Pearson, and J. Clause, “How does code obfuscation impact energy usage?” in 30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014, 2014, pp. 131–140.

[6] M. L. Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. D. Penta, and D. Poshyvanyk, “Mining energy-greedy API usage patterns in android apps: an empirical study,” in 11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India, 2014, pp. 2–11.

[7] C. Sahin, L. Pollock, and J. Clause, “How do code refactorings affect energy usage?” in Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2014, pp. 36:1–36:10.

[8] G. Pinto, F. Castor, and Y. D. Liu, “Understanding energy behaviors of thread management constructs,” in Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications, ser. OOPSLA ’14. New York, NY, USA: ACM, 2014, pp. 345–360. [Online]. Available: http://doi.acm.org/10.1145/2660193.2660235

[9] K. Liu, G. Pinto, and Y. Liu, “Data-oriented characterization of application-level energy optimization,” in Proceedings of the 18th International Conference on Fundamental Approaches to Software Engineering, ser. Lecture Notes in Computer Science, vol. 9033, 2015, pp. 316–331.

[10] Luis Gabriel Lima, Francisco Soares-Neto, Paulo Lieuthier, Fernando Castor, Gilberto Melfe, João Paulo Fernandes: Haskell in Green Land: Analyzing the Energy Behavior of a Purely Functional Language. SANER 2016: 517-528

[11] Luis Gabriel Lima, Francisco Soares-Neto, Paulo Lieuthier, Fernando Castor, Gilberto Melfe, João Paulo Fernandes, On Haskell and energy efficiency. Journal of Systems and Software 149: 554-580 (2019)

[12] Rui Pereira, Marco Couto, João Saraiva, Jácome Cunha, João Paulo Fernandes: The influence of the Java collection framework on overall energy consumption. GREENS@ICSE 2016: 15-21

[13] Rui Pereira, Pedro Simão, Jácome Cunha, João Saraiva: jStanley: placing a green thumb on Java collections. ASE 2018: 856-859

GREEN SOFTWARE IN AN ENGINEERING COURSE

Sustainable development has become an increasingly important theme not only in the world politics, but also an increasingly central theme for the engineering professions around the world. Software engineers are no exception as shown in various recent research studies. Despite the intensive research on green software, today's undergraduate computing education often fails to address our environmental responsibility. We present a module on green software that we introduced as part of an advanced course on software engineering. We introduce a catalogue of energy smells and green refactorings, which our preliminary results show that do help students in reason and optimizing the energy consumption of software systems.

References

[1] A. Chandrakasan, S. Sheng, and R. Brodersen, “Low-power cmos digital design,” Solid-State Circuits, IEEE Journal of, vol. 27, no. 4, pp. 473– 484, Apr 1992.

[2] V. Tiwari, S. Malik, and A. Wolfe, “Power analysis of embedded software: a first step towards software power minimization,” Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 2, no. 4, pp. 437–445, Dec 1994.

[3] M. L. Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. D. Penta, and D. Poshyvanyk, “Mining energy-greedy API usage patterns in android apps: an empirical study,” in 11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India, 2014, pp. 2–11.

[4] C. Sahin, L. Pollock, and J. Clause, “How do code refactorings affect energy usage?” in Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2014, pp. 36:1–36:10.

[5] G. Pinto, F. Castor, and Y. D. Liu, “Understanding energy behaviors of thread management constructs,” in Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications, ser. OOPSLA ’14. New York, NY, USA: ACM, 2014, pp. 345–360.

[6] K. Liu, G. Pinto, and Y. Liu, “Data-oriented characterization of application-level energy optimization,” in Proceedings of the 18th International Conference on Fundamental Approaches to Software Engineering, ser. Lecture Notes in Computer Science, vol. 9033, 2015, pp. 316–331.

[7] Luis Gabriel Lima, Francisco Soares-Neto, Paulo Lieuthier, Fernando Castor, Gilberto Melfe, João Paulo Fernandes: Haskell in Green Land: Analyzing the Energy Behavior of a Purely Functional Language. SANER 2016: 517-528

[8] Rui Pereira, Marco Couto, João Saraiva, Jácome Cunha, João Paulo Fernandes: The influence of the Java collection framework on overall energy consumption. GREENS@ICSE 2016: 15-21

[9] Rui Pereira, Pedro Simão, Jácome Cunha, João Saraiva: jStanley: placing a green thumb on Java collections. ASE 2018: 856-859

SOFTWARE APPLICATION ENERGY PROFILING FOR JAVA PROJECTS

This tutorial addresses the energy efficiency of software applications implemented in Java programming language. The first part describes the current state of the art in energy profiling as well as options for displaying energy consumption of segments of the source code. Next, a custom code analysis method for displaying information is presented addressing the processor, operating memory, and hard disk. After this analysis, a short introduction to the implemented Java application follows. In order to demonstrate the practical use of the application, several test solutions are created, where we measured the energy consumption. With each example, we put emphasis on solving one problem with at least two solutions to determine which implementation has lower energy intensity. The results of the examples are also part of this tutorial.

Boolean vs. boolean (billion times)

Type	AVG exec (s)	AVG CPU NRG (W)	AVG RAM NRG (W)	AVG HDD NRG (W)	Test count
boolean	0,49900	6,31915	0,00087	n/a	10000
Boolean	0,49879	6,26819	0,00071	n/a	10000

```
Data types boolean vs Boolean
boolean g = false;
 for (long i = 0 ; i<1000000000;i++){
    g = true;
}

Boolean h = false;
for (long i = 0 ; i<1000000000;i++){
    h = true;
}
```

Double vs. double (billion times)

Type	AVG exec (s)	AVG CPU NRG (W)	AVG RAM NRG (W)	AVG HDD NRG (W)	Test count
double	1,01489	6,18481	0,00096	n/a	9643
Double	5,66532	7,41853	0,01001	n/a	9294

```
STRING CREATOR  – StringBuilder vs. StringBuffer
StringBuilder test = new StringBuilder();
for(long i=0;i<=20000000;i++) { //100M Java heap space
    test.append(i);
}

StringBuffer stringBuffer = new StringBuffer();
for(int i=0;i<=20000000;i++) {
    stringBuffer.append(i);
}

STRING CREATOR  –– += vs. concat
String testString = "";
for(long i=0;i<=50000;i++){
    testString = testString.concat(String.valueOf(i));
    testString += String.valueOf(i);
}
```

```
sorting.bubbleSort();
sorting.selectionSort();
sorting.insertionSort();
sorting.quickSort();
sorting.mergeSort();
sorting.heapSort();
```

```
matrixMultiplication.strassenAlgMultiplication();
matrixMultiplication.classicalMultiplication();
```

```
hashGenerator.md5();
hashGenerator.sha1();
hashGenerator.sha256();
hashGenerator.sha384();
hashGenerator.sha512();
```

```
TreeMap vs HashMap vs LinkedHashMap 10;
TreeMap<Integer, Integer> treeMap = new TreeMap<>();
HashMap<Integer,Integer> hashMap = new HashMap<>();
LinkedHashMap<Integer, Integer> linkedHashMap = new LinkedHashMap<>();

for (int i = 0; i < 5000000; i++) {
    treeMap.put(i,  v: i + 1);
    linkedHashMap.put(i,  v: i + 1);
    hashMap.put(i,  v: i + 1);
}
```

References

[1] D. Li, W. G. J. Halfond, An investigation into energy-saving programming practices for android smartphone app development, in Proc. of the 3rd International Workshop on Green and Sustainable Software, GREENS 2014, ACM, 2014, pp. 46–53.

[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] 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

[5] Cs. Szabo, E.M.M. Alzeyani: Measuring Energy Efficiency of Selected Working Software, Studia Universitatis Babe¸s-Bolyai Informatica, Vol. 63, No. 1, 2018, pp. 5–16. https://doi.org/10.24193/subbi.2018.1.01

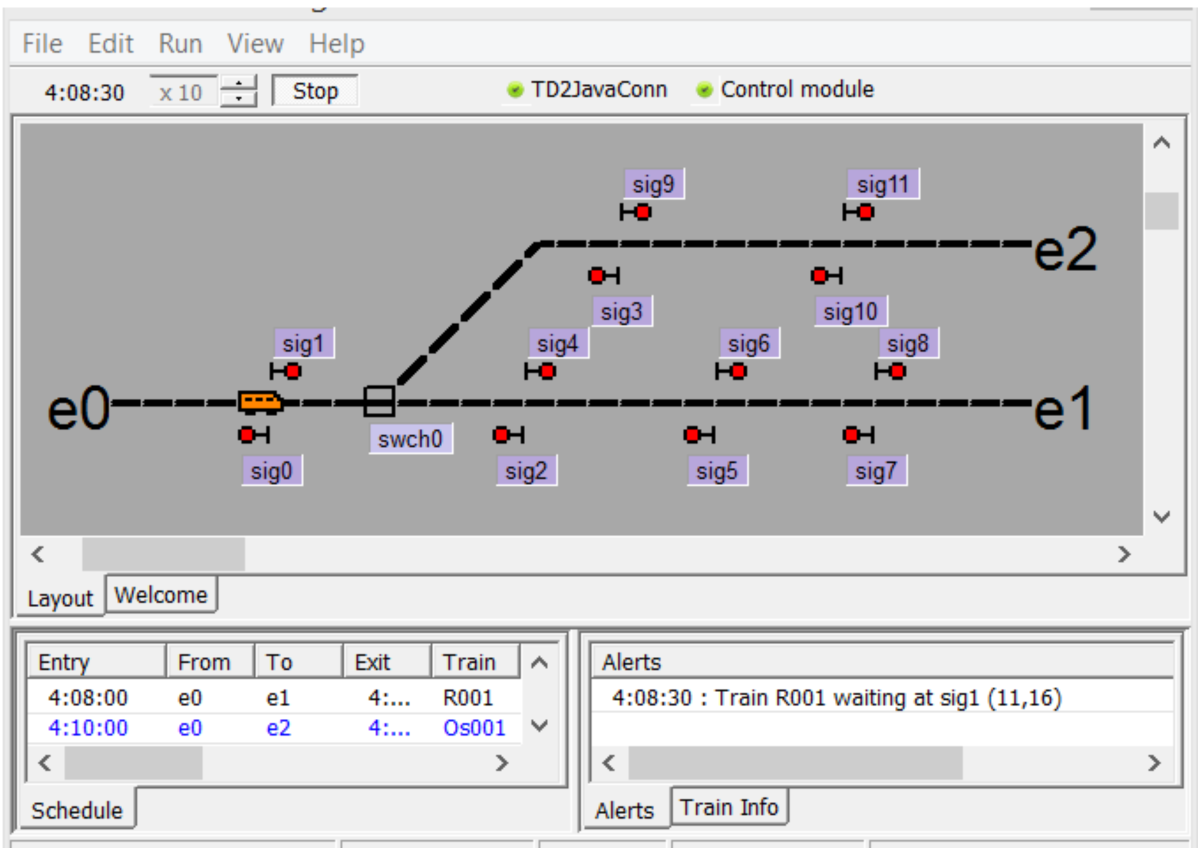
[6] Cs. Szabo, J. Saraiva: Focusing software engineering education on green application development, in Conference of Information Technology and Development of Education – ITRO 2017, Novi Sad, Serbia, pp. 165–169, ISBN 978-86-7672-302-7.

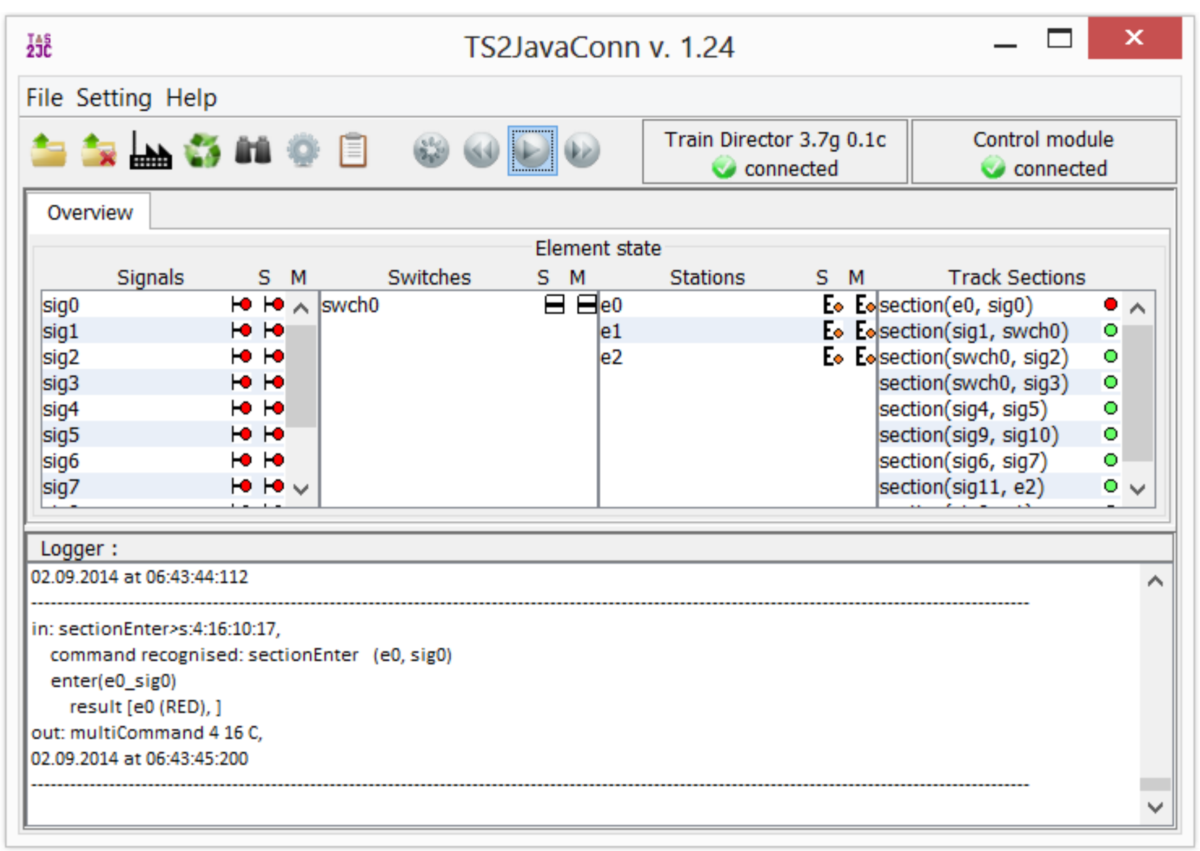
DEVELOPMENT OF CORRECT SOFTWARE WITH B-METHOD

One of the well-recognized approaches to the development of correct software systems is the utilization of formal methods (FM) for their specification and verification. FM are rigorous mathematically based techniques for the specification, analysis, development and verification of software and hardware. Rigorous means that a formal method provides a formal language with unambiguously defined syntax and semantics and mathematically based means that some mathematical apparatus (formal logic, set theory, etc.) is used to define the language.

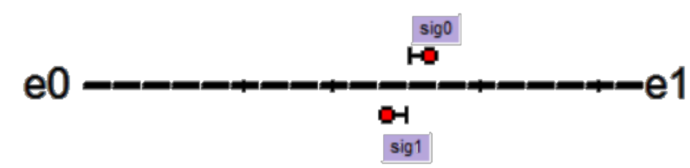
One of the FM used in industrial practice is B-Method[1,2,3,4], a state based, model-oriented formal method intended for software development. B-Method is primarily used in the railway sector, for the safety-critical software behind automated urban metro subway systems (including the one in Budapest). The strength of B-Method lies in a well-defined development process, which allows to specify a software system as a collection of components called B-machines and to refine such an abstract specification to a concrete one. The concrete specification can be automatically translated to ADA, C or another programming language. An internal consistency of the abstract specification and correctness of each refinement step are verified by proving a set of predicates called proof obligations (PObs). The whole development process, including proving, is supported by an industrial-strength software tool called Atelier B[5].

This tutorial serves as a gentle, practical, introduction to B-Method. During the tutorial, the participants will develop a simple software controller for a railway scenario. They will be able to run the scenario with the controller in a toolset containing corresponding simulation game.

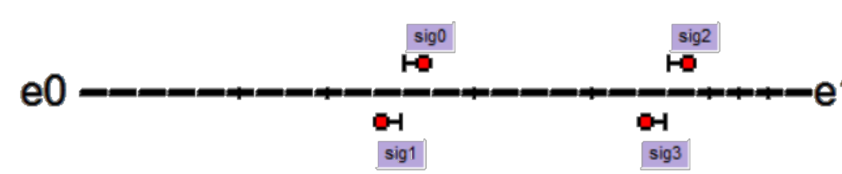




This tutorial serves as a gentle, practical, introduction to B-Method. During the tutorial, the participants will develop a simple software controller for a railway scenario. They will be able to run the scenario with the controller in the Train Director/TS2JavaConn (TD/TS2JC) toolset [6,7]. The toolset consists of a modified version of the Train Director [8] (Fig. 1) simulation game and an application called TS2JavaConn (Fig. 2), which allows using separately developed software controllers with the simulation game. With the ambition of using the toolset for controller prototyping it has been later [9] extended by a customized version of the Open Rails [10] 3D train simulator.



After an introduction to the B-Method, the participants of the course are given a controller for a single-track railway scenario with two sections (Fig.3). The controller is written in the language of B-Method. During the course, they develop and verify a controller for a single-track railway scenario with three sections (Fig. 4).



Listing 1. The controller for the railway scenario with two sections, written in the language of B-Method

```
MACHINE  route2sec

SETS

   PROP_SIGNAL={green, red};

   PROP_SECTION={free,occup}

CONCRETE_VARIABLES

   e0, e1, sig0, sig1, e0_sig1, sig0_e1

INVARIANT

   e0:PROP_SIGNAL & e1:PROP_SIGNAL &  sig0:PROP_SIGNAL & sig1:PROP_SIGNAL
&

   e0_sig1:PROP_SECTION & sig0_e1:PROP_SECTION &

   (e0=green => sig1=red) & (sig1=green => e0=red)  &

   (e1=green => sig0=red) & (sig0=green => e1=red)  &

   (e0=green   => e0_sig1=free) & (sig1=green => e0_sig1=free) &

   (e1=green   => sig0_e1=free) & (sig0=green => sig0_e1=free)


INITIALISATION

   e0:=red || e1:=red ||    sig0:=red || sig1:=red ||    e0_sig1:= free || sig0_e1:= free

OPERATIONS

   ss <-- getSig_sig0 = BEGIN ss:=sig0 END;

   ss <-- getSig_sig1 = BEGIN ss:=sig1 END;

   ss <-- getEntry_e0 = BEGIN ss:=e0 END;

   ss <-- getEntry_e1 = BEGIN ss:=e1 END;

   reqGreen_e0 = IF sig1=red & e0_sig1=free THEN e0:=green END;

   reqGreen_e1 = IF sig0 = red & sig0_e1 = free THEN e1:=green END;

   reqGreen_sig0 = IF e1=red & sig0_e1= free THEN  sig0:=green END;

   reqGreen_sig1 = IF  e0 = red & e0_sig1 = free THEN  sig1:=green END;

   enterNI_e0_sig1 = BEGIN e0_sig1:=occup || e0:=red || sig1:=red END;

   enterIN_sig0_e1 = BEGIN sig0_e1:=occup || sig0:=red || e1:=red END;

   enterNI_e1_sig0 = BEGIN sig0_e1:=occup || sig0:=red || e1:=red END;

   enterIN_sig1_e0 = BEGIN e0_sig1:=occup || e0:=red || sig1:=red END;

   leaveNI_e0_sig1 = BEGIN e0_sig1:=free  END;

   leaveIN_sig0_e1 = BEGIN sig0_e1:=free  END;

   leaveNI_e1_sig0 = BEGIN sig0_e1:=free  END;

   leaveIN_sig1_e0 = BEGIN e0_sig1:=free  END

END
```

References

1. Abrial, J. R.: The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.
2. Abrial, J. R. : Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.
3. Lano, K.: The B Language and Method: A Guide to Practical Formal Development, Springer-Verlag, FACIT series, 1996.
4. Schneider, S.: The B-Method: An Introduction, Palgrave Macmillan, Cornerstones of Computing series, 2001.
5. https://www.atelierb.eu/
6. Korečko, Š., Sorád, J.: Using simulation games in teaching formal methods for software development, In: Innovative Teaching Strategies and New Learning Paradigms in Computer Programming, R. Queirós, Ed., pp. 106–130, IGI Global, 2015.
7. Korečko, Š., Sorád, J., Dudláková, Z., Sobota, B.: A Toolset for Support of Teaching Formal Software Development In: Lecture Notes in Computer Science : Software Engineering and Formal Methods : 12th Internatinal Conference, SEFM 2014, pp. 278-283, Springer, 2014.
8. https://www.backerstreet.com/traindir/en/trdireng.php
9. Korečko, Š., Sobota, B.: Computer Games as Virtual Environments for Safety-Critical Software Validation, in Journal of Information and Organizational Sciences, vol. 41, no. 2, 2017.
10. http://openrails.org

PROGRAMMING OF ADVANCED MANAGEMENT AND ORCHESTRATION OF VIRTUALISED NETWORK RESOURCES - SELECTION OF CASE STUDIES

New management and orchestration (MANO) functions are standardised for use in distributed and virtualised network environments. Their main role is to provide safe and reliable operation of applications using network functions. Therefore, as continuation of our previous lecture where we provide basic concepts, here in this lecture we will provide a selection of case studies where these functions are implemented and explain the advanced mechanisms behind and simplicity of their application.

References

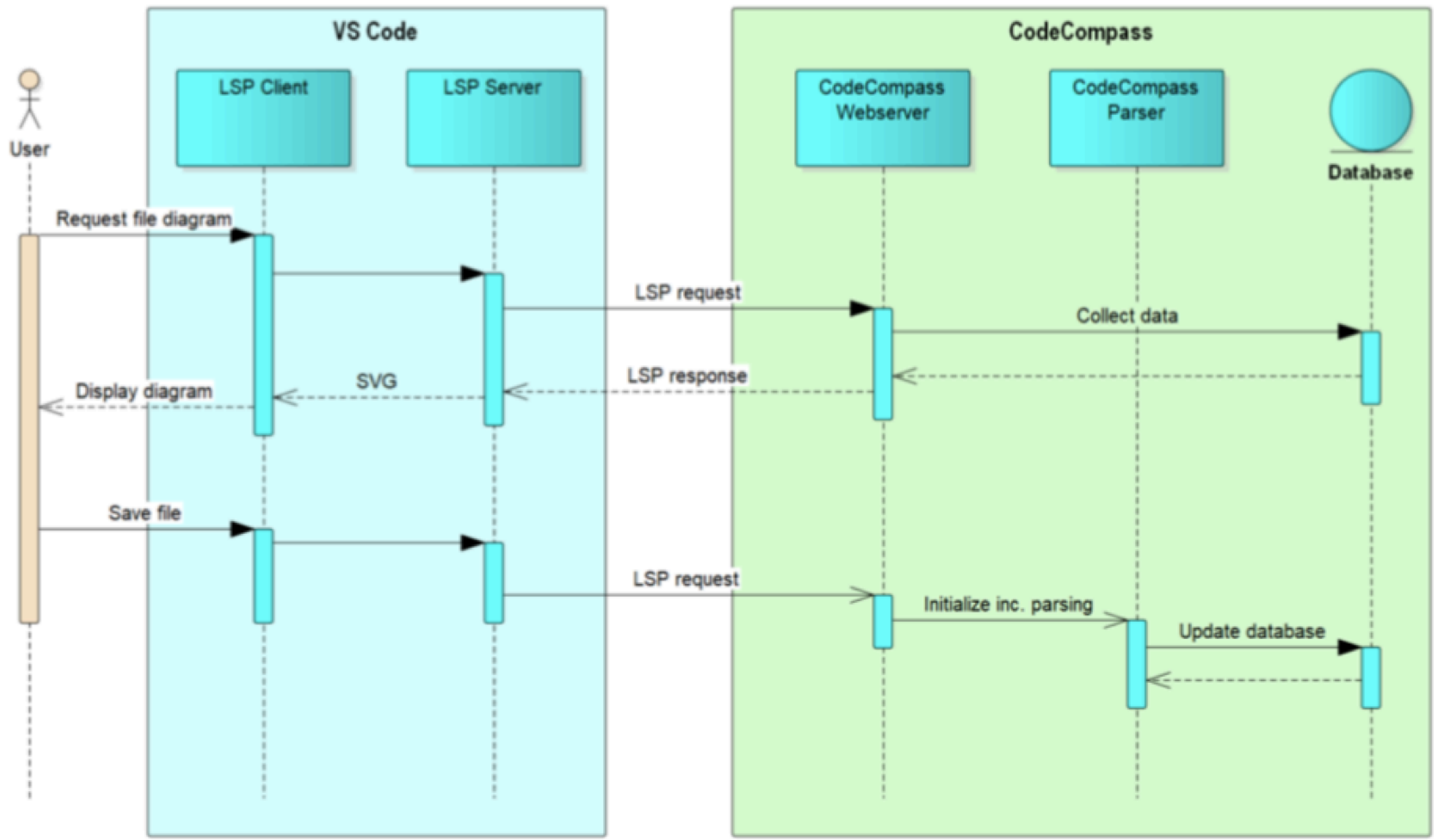
[1] ETSI Industry Specification Group (ISG) NFV: ETSI GS NFV- MAN 001 v1.1.1: Network Functions Virtualisation (NFV); Management and Orchestration European Telecommunications Standards Institute (ETSI), 2014, https://www.etsi.org/deliver/etsi_gs/NFV- MAN/001_099/001/01.01.01 60/gs_NFV-MAN001v010101p.pdf, accessed July 1, 2018

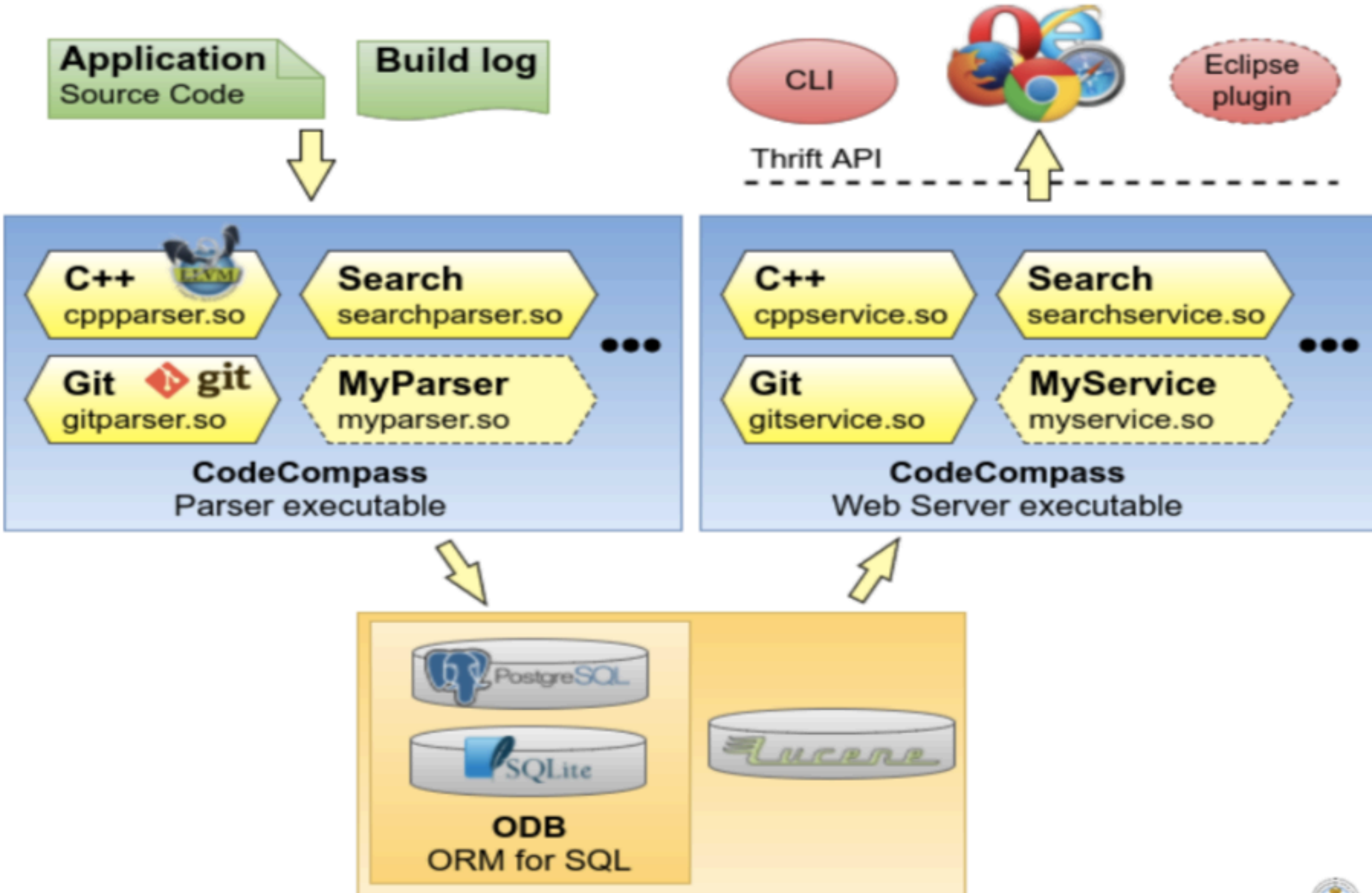
[2] Han, B., Gopalakrishnan, V., Ji, L., Lee, S.: Network function virtualization: Challenges and opportunities for innovations. IEEE Communications Magazine 53(2), 90–97 (2015)

[3] OpenStack Cloud Software. OpenStack Foundation (2018), www.openstack.org, accessed July 1, 2018

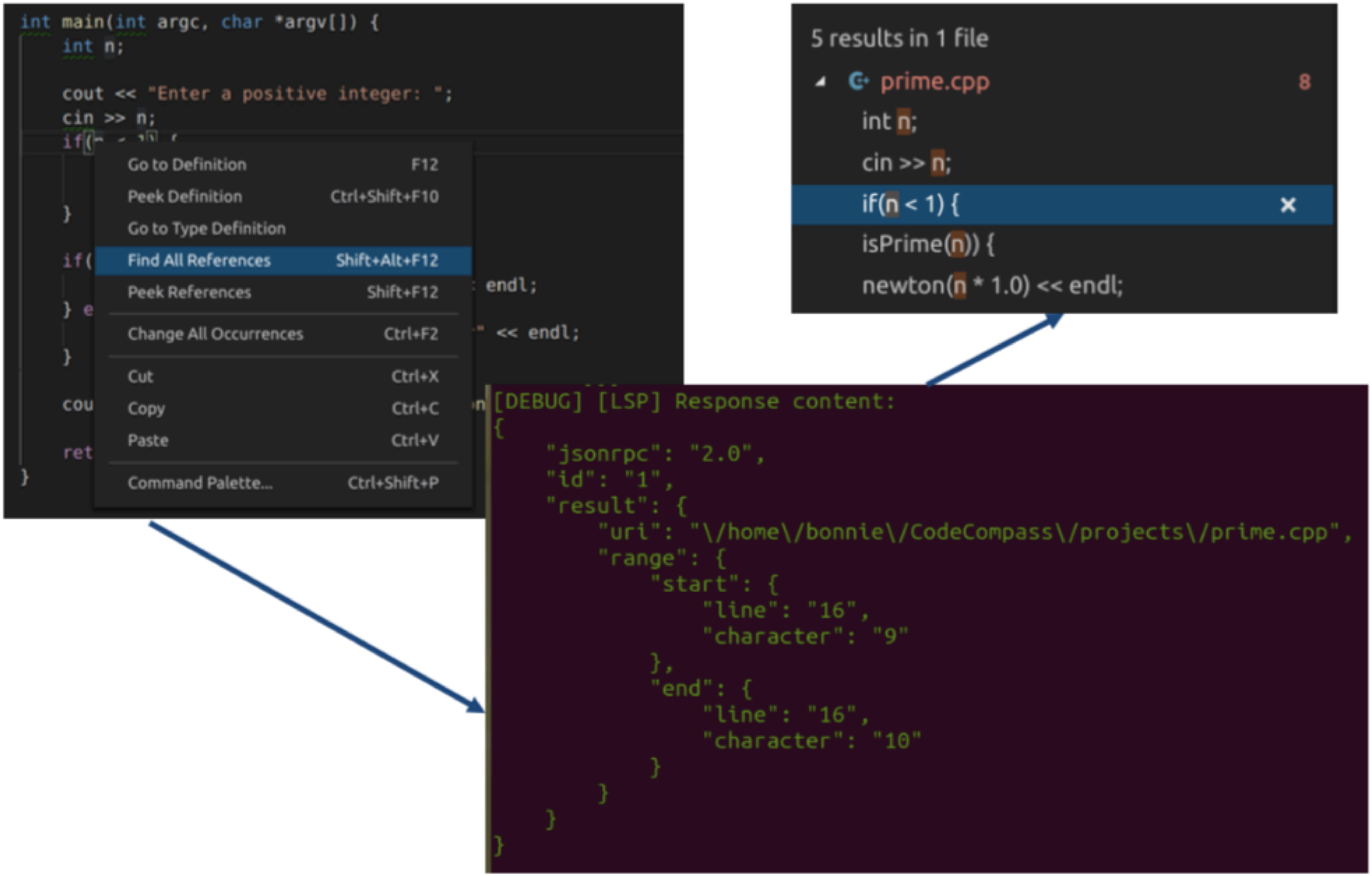
CODE COMPREHENSION WITH ADVANCED TOOL SUPPORT

In this tutorial we will introduce the state of the art code comprehension tools to the students. We will give a theoretical foundation for code comprehension, navigation and code visualisation methods and approaches to apply them in practical software development. In the practice session we will demonstrate how to set up a specific toolset: CodeCompass with incremental parsing, Visual Studio Code as front-end tool and the usage of the Language Server Protocol. We will parse an open source library and find and fix a specific bug in it using the toolset.


VS Code
CodeCompass
User
LSP Client
LSP Server
CodeCompass Webserver
CodeCompass Parser
Database
Request file diagram
LSP request
Collect data
SVG
LSP response
Display diagram
Save file
LSP request
Initialize inc. parsing
Update database



Application
Source Code
Build log
CLI
Eclipse plugin
Thrift API
C++
cppparser.so
Search
searchparser.so
Git
git
gitparser.so
MyParser
myparser.so
CodeCompass
Parser executable
C++
cppservice.so
Search
searchservice.so
Git
gitservice.so
MyService
myservice.so
CodeCompass
Web Server executable
PostgreSQL
SQLite
ODB
ORM for SQL
Lucene


int main(int argc, char *argv[]) {
int n;
cout << "Enter a positive integer: ";
cin >> n;
Go to Definition F12
Peek Definition Ctrl+Shift+F10
Go to Type Definition
Find All References Shift+Alt+F12
Peek References Shift+F12
Change All Occurrences Ctrl+F2
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Command Palette... Ctrl+Shift+P
5 results in 1 file
prime.cpp 8
int n;
cin >> n;
if(n < 1) {
isPrime(n)) {
newton(n * 1.0) << endl;
[DEBUG] [LSP] Response content:
{
"jsonrpc": "2.0",
"id": "1",
"result": {
"uri": "\/home\/bonnie\/CodeCompass\/projects\/prime.cpp",
"range": {
"start": {
"line": "16",
"character": "9"
},
"end": {
"line": "16",
"character": "10"
}
}
}
}

References

[1] Jonathan Sillito, Gail C. Murphy, Kris De Volder. (2008). Asking and Answering Questions during a Programming Change Task. IEEE Transactions on Software Engineering, VOL. 34, NO. 4, July/August 2008.

[2] Porkolab, Zoltan & Brunner, Tibor & Krupp, Daniel & Csordas, Marton. (2018). Codecompass: an open software comprehension framework for industrial usage. 361- 369. 10.1145/3196321.3197546.

[3] Nathan Hawes, Stuart Marshall, Craig Anslow. (2015). CodeSurveyor: Mapping LargeScale Software to Aid in Code Comprehension. 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT) , 27-28 Sept. 2015.

[4] Porkolab,Zoltan & Brunner,Tibor (2018). The codecompass comprehension framework. 393-396. 10.1145/3196321.3196352

[5] CodeCompass, https://github.com/Ericsson/CodeCompass. Last accessed 5 Nov 2018.

[6] Brunner, Tibor & Porkolab, Zoltan. (2017). Two Dimensional Visualization of Soft- ware Metrics. Proceedings of the Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications.

[7] B. De Alwis and G.C. Murphy. (1998). Using Visual Momentum to Explain Dis- orientation in the Eclipse IDE. Proc. IEEE Symp. Visual Languages and Human Centric Computing, pp. 51-54, 2006.

FUNCTIONAL ARRAY PROGRAMMING WITH SINGLE ASSIGNMENT C: OPPORTUNITIES AND CHALLENGES

SAC (Single Assignment C) is in several aspects a functional programming language out of the ordinary. As the name suggests, SAC combines a C-like syntax (with lots of curly brackets) with a state-free, purely functional semantics. Originally motivated to ease adoption by programmers with an imperative background, the choice offers surprising insights into what constitutes a "typical" functional or a "typical" imperative programming language construct. Again on the exotic side for a functional language, SAC emphasises multi-dimensional arrays, instead of lists and trees. Array programming treats multi-dimensional arrays in a holistic way: functions map potentially huge argument arrays to result arrays with a call-by-value semantics, and new array operations are defined by composition of existing ones. SAC is a high-productivity language for application domains that deal with large collections of data in a computationally intensive way.

At the same time SAC also is a high performance language competing with low-level imperative languages through compilation technology. The abstract view on arrays combined with the functional semantics support far-reaching program transformations. A highly optimised runtime system takes care of automatic memory management with a focus on immediate memory reuse. Last not least, the SAC compiler exploits the state-free semantics of SAC and the data-parallel nature of SAC programs for fully compiler-directed acceleration on a large variety of contemporary machine architectures, from multi-core servers to GPGPU accelerators and clusters of workstations.

The lectures motivate the language design of SAC and provide a hands-on introduction to array programming as a paradigm. We look into all aspects from the underlying array calculus to the concrete language design with imperative-looking functional code, discuss a multitude of examples, explore compilation challenges and eventually see some performance results on various parallel computing architectures.

BALANCED DISTRIBUTED COMPUTATION PATTERNS

The state-of-the-art concurrent software development made extensive usage of various methodologies and approaches to obtain high speed up. However, parallelism remains one of the most difficult domains especially in the case of pattern based programming approaches. The main purpose is to explore parallel computation schemes in a new environment, to illustrate the appropriateness and applicability in novel distributed computation setups. The amount of parallelism is explored based on many factors such as: applied computation pattern refined granularity, semantics of distributed nodes, data streaming, and especially load balancing.

References

[1] Zsok V.: D-Clean Semantics for Generating Distributed Computation Nodes, Work- shop on Generative Technologies, WGT 2010, Satellite workshop at ETAPS 2010, Paphos, Cyprus, March 27, 2010, pp. 77–84.

[2] Zsok V., Hernyak Z., and Horvath, Z.: Designing Distributed Computational Skeletons in D-Clean and D-Box. Central European Functional Programming School CEFP 2005, First Summer School, Budapest, Hungary, July 4-15, 2005, Revised Selected Lectures, LNCS Vol. 4164, Springer-Verlag, 2006, pp. 223–256.

[3] Zsok V., Koopman, P., Plasmeijer, R.: Generic Executable Semantics for D-Clean, Proceedings of the Third Workshop on Generative Technologies, WGT 2011, ETAPS 2011, Saarbrucken, Germany, March 27, 2011, ENTCS Vol. 279, Issue 3, Elsevier, December 2011, pp. 85–95.

[4] Zsok V., Porkolab Z.: Rapid Prototyping for Distributed D-Clean using C++ Tem- plates, Annales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae, Sectio Computatorica, Eotvos Lorand University, Budapest, Hungary, 2012, Vol. 37, pp. 19–46.

[5] Zsok V. et al.: Modeling CPS Systems using Functional Programming, Proc. of IFL17, Uni. of Bristol, pp. 168–174.