PhoneLab

Research Proposal (Second Round, 2011)
Microsoft Innovation Cluster for Embedded Software

Project Title:
PhoneLab: Cloud-Backed Development Environment for Smartphones

Project Duration: 2 years

Expected Amount of Funding: 120'000 CHF in total

The funding will support one EPFL doctoral student for two years.

Abstract:

We will develop a Scala-based language and a development environment to simplify the construction of cloud-backed smartphone applications, both by professionals and by end users. We will develop programming assistance tools that use cloud analysis (running on `http://ecocloud.ch` infrastructure) to suggest code fragments, and enable development and customization of applications both from the desktop and directly from smartphones.

Principal Investigators:

Viktor Kuncak
Assistant Professor
EPFL School of Computer and Communication Sciences

Martin Odersky
Full Professor
EPFL School of Computer and Communication Sciences

30 October 2011

1 Full Project Description

1.1 Motivation: Empowering users to Develop for Smartphones

The increasing adoption of smartphones has provided tens of millions of users with substantial resources for computation, communication, and sensing. The constant availability of these resources has a huge potential to positively transform our society. The potential for this transformation is limited by a new instance of software crisis, where the number of developers becomes a bottleneck. Namely, due to physical constraints (such as input device size and their energy consumption), the development for smartphones is more difficult than the development for desktops. At the same time, the true benefits of these platforms can only be realized through specialized applications for particular domains, which requires the involvement of domain experts that are not necessarily professional developers.

A promising approach to address the crisis is to empower users themselves to perform programming, blurring the traditional divide between professional software developers and end users. This direction is especially appealing as the number of people exposed to these computational devices continues to grow, the capabilities of these devices enable the adoption of advanced software development tools, and new algorithmic techniques enable more sophisticated code assistance functionality. Our project focuses precisely on such code assistance functionalities, as well as related application customization capabilities.

1.2 State of the Art

We propose to develop innovative technology for interactive synthesis and customization of applications. We will draw inspiration and insight from the **TouchStudio/TouchDevelop project** (with whose developers from Redmond we are already in contact), as well as the interactive synthesis of code snippets (`http://lara.epfl.ch/w/insynth`), which we have recently started developing as an opportunistic extension of the ongoing Proglab.NET ICES project. The TouchStudio/TouchDevelop project is related in the spirit to our proposed work, because it also uses a language whose that simplified the general programming model to make it easier for developers to use it, and because it has a completion facility for selecting methods to apply. The first advances we propose is reflected in the nature and deployment of the language we propose. We propose to deploy it as an embedding within a general purpose language (Scala), which will make them easier to extend and easier to benefit from the broader language ecosystem. The second advance we propose is user-oriented modification and customization of applications. This mode of changing applications will be based on developing new ways of linking the manifested application behavior with the application source code that caused this behavior. As a result, it will generalize the notion of macro recording and macro editing. The third advance we propose the algorithms used to assist the developers. Instead of suggesting individual methods, we propose to synthesize entire expressions based on type constraints, test cases, and verification results. Our algorithms will therefore enable developers to make much bigger steps than individual completion of methods, and support these steps by more sophisticated decision-making machinery.

Scala is currently being used to develop a number of DSLs, including the DSLs in the

context of the senior ERC grant awarded recently to Martin Odersky [4]. Other examples of languages for particular domains include compiler generators (for lexers, parsers, and code generators), nesC [6], Mace [12], and languages in the BOOM project [1]. The BOOM project focuses on leveraging cloud resources very efficiently, whereas our model makes use of cloud but focuses primarily on smartphone programming side.

Learning by demonstration and inductive learning is also related to one part of our proposal, but has previously been applied to different domains. Examples of systems in this area include Sheeodog [3] and IGOR II [10]. The notion of partial programs has been successfully explored in the sketching approach to synthesis [17]. A system for learning functions on string-valued spreadsheet columns has also been recently developed [8].

We next review state of the art in code fragment synthesis functionality. Several tools, including Prospector [15], XSnippet [16], Strathcona [11], PARSEWeb [20] and SNIFF [5] generate or search for the relevant code examples. In contrast to all these tools we aim to support polymorphic types and expressions with higher order functions. Additionally, we aim to synthesize snippets using all visible methods in a context, where they build or present them only if they exist in a corpus. Prospector, Strathcona and PARSEWeb do not incorporate the extracted examples into the current program context; this requires additional effort on part of the programmer. Moreover, Prospector does not solve queries with multiple argument methods unless a user initiate multiple queries. Type-based completion functionality is also emerging in proof assistants, such as Agda [2]. By type checking the unfinished program, the programmer can get useful information on how to fill in the missing parts. However, the existing approaches based on expressive types are less scalable than the algorithms we propose.

1.3 Proposed Advances and Research Directions

1.3.1 Language SL for Programming Smartphones

Smartphones backed by cloud services will be a major programming platform of the future; our first proposed advance is to develop a programming model and a domain-specific language (DSL) for this platform. We call our proposed language SL (for Smartphone Language).

The platform combining smartphones and cloud has a huge potential to improve the quality of human lives. Whereas smartphones provide sensing, interface, and ubiquitous access, cloud provides the capability to handle bursty compute-intensive jobs that rely on large background knowledge. Handling such compute-intensive jobs efficiently is essential for interactive applications that perform sophisticated decision making. Cloud also supports reliable storage and collaboration facilities that are essential for many applications. However, developing applications for this platform using current tools is difficult.

The first challenge is that these two physically very different platforms need to be viewed as one entity from the perspective of developers and users. SL will provide a unified model for this platform that enables both informal reasoning and rigorous tool-supported verification for the absence of program errors. The SL implementation will enable transparent communication with servers, including the use of cloud storage and invocation of compute jobs in the cloud. It will support persistent caching to improve performance and reduce communication bandwidth.

The next challenge is to develop a language that makes it easy to build applications that process and interpret complex noisy inputs such as video camera, GPS, motion sensors, as well as the touch screen with on-screen keyboard. To help with such tasks the language will support easy access to existing data processing services, such as the OCR service of the **Hawaii project**, image processing, geo-location, as well as language translation services.

To support composition of these sophisticated processing services and enable the building of adaptive applications, SL will use declarative descriptions of programs whenever possible. It will support under-specification and probabilistic computation with distributions. As an example of multi-modal input, suppose that a user takes a photo with the phone camera and starts using the on-screen keyboard to more narrowly indicate the object of interest. SL will support high-level composition operations that can integrate an answer from an image-recognition service with the results from the on-screen keyboard or handwriting module. Note that for such tasks to work well it is necessary that program modules return a distribution over possible values instead of just a "locally best" guess. To build such execution mechanisms for SL we will use our experience with a constraint programming in Scala [14] and with non-deterministic languages [7].

Developing a language that meets these requirements is an ambitious research direction. To make this exploration feasible within the bounds of this proposal we will develop our DSL by focusing on common application tasks in this domain, favoring programming simplicity at the expense of generality. Furthermore, the DSL will not be implemented from scratch but will be embedded within the Scala programming language. This will dramatically simplify our development efforts by leveraging the existing Scala infrastructure. This will allow us to focus on the key novel aspects, such as the efficient implementation of the unified programming model and automated combination of services providing probabilistic answers.

1.3.2 On-The-Fly Customization of SL Applications

To enable the development from smartphones themselves and enable end users to participate in creating applications, we will develop notions of programs and interfaces that blend application execution, customization, debugging, and programming. To achieve this, we will introduce the ability to modify an existing application by introducing alternate execution paths, specified through a combination of programming by demonstration (macro recording) and assisted text editing of high-level programs.

We plan to explore such customization activities at several levels. Our runtime will provide an active, editable interpretation of information provided to the user. For example, if a text message is shown to a user, an application customization mode can be used to edit this text. The challenge is reflecting such dynamic change as a static change of the application. Such functionality will enable very easy translation of applications to particular spoken languages, but is not limited to this aspect. It also has the potential to eliminate the need for programming customization logic, which typically arises in larger applications but is often arbitrarily limited to a few selected options.

More generally, we will use the fact that SL program is probabilistic to enable the changes to its behavior. The execution of an SL program is based on the initial model of the world and user's expectations. Through interaction with users, we will allow users

to adjust these expectations and therefore adapt the programs. Whereas adaptation in existing applications is fairly limited to certain rare training functionality (as in speech recognition systems), every SL program will be subject to modification by interactive learning. If a program goes through a sequence of steps ending in an answer that the user views as less than ideal, the user can obtain a high-level view of the experienced execution sequence and modify key points, changing the underlying probabilistic model and the future control behavior of the program.

Furthermore, access to social networks on smartphones opens up the possibility of users sharing their application modifications, and creating custom views of applications that are best adapted to the expectations of a social group.

1.3.3 Interactive Type-Driven Synthesis for SL

Given the domain-specific language SL and its runtime, our main goal is to further improve developer productivity using tools that construct parts of SL programs automatically. Our starting point is a novel approach to synthesize code fragments consisting from library (API) function calls. The idea is to show a number of suggested code fragments from which the user can select one [9]. To determine candidate code fragments, our approach starts by leveraging type constraints. It encodes the code synthesis problem using a set of formulas and uses a sound theorem proving algorithm to check whether there exists an expression of a given type in a given type environment. While theorem proving may sound expensive, we have found that it becomes practical given appropriate cost functions that guide the search. We plan to develop this technology further in the context of the SL language. Because SL is domain-specific, we expect that the search will encounter a smaller number of possibilities and produce higher-quality results than for general Scala programs.

Crucial to making the proposed approach practical are search and ranking functions that select among many possible code fragments that satisfy syntax, scoping, and type constraints. We believe that proximity heuristics (ranking lexically closer functions higher) are helpful for programming within one application, but deeper information must be derived to prioritize methods from multiple APIs. We will deploy machine learning algorithm implementations to learn the distribution of API function calls. Our approach will examine existing source code bases to compute method invocations that are likely to be useful in a given context. The appropriate notion of features for predicting code fragments will be a key part of the design of this machine learning approach.

As the key algorithmic foundation for our approach, we will systematically study the new notion of *quantitative type inhabitation*, in which type declarations contain weights and we are interested in computing expressions that minimize the combined weight of synthesized expressions. We will explore first the cases of monomorphic type declarations, investigating practical restrictions of the full inhabitation problem that was proven PSPACE-complete by Statman [18]. We will then explore more expressive type systems that include generic types. In this situation the general problem quickly becomes undecidable. However, we are interested in code fragments of bounded length. This gives back the decidability of the problem, and suggests natural and predictable algorithms that explore the space using weight functions that grow as the size expressions increases.

Note also that static and dynamic assistance can be combined as follows. SL supports

non-determinism, which means that if the developer considers multiple suggested code fragments acceptable, the system can generate a non-deterministic choice between these alternatives, and leave it to the previously described run-time customization to determine which alternative works better during the execution.

As described so far, the synthesized code fragments are only guaranteed to confirm to *type* constraints. To ensure their *semantic* appropriateness we will apply automated testing and verification to the resulting code, building on the recent development of the Leon verifier for a purely functional subset of Scala and Scala effect analysis efforts in LAMP and LARA.

1.4 Software Stack and Leveraged Technologies

1.4.1 Scala Compiler Infrastructure

The software foundation for this project is the Scala language and tools. The language and tools are under continuous development by our groups. The capabilities of Scala on .NET have been extended substantially in the Proglab.NET project. Of particular importance for this project are techniques for developing domain-specific embedded languages [4], as well as the extensible plugin architecture of the Scala compiler.

1.4.2 Machine Learning using INFER.NET

The semantics of SL will involve probabilistic computation. As one of the execution mechanisms we will therefore explore the use of INFER.NET library, which allows both inference and learning of probabilistic models. Furthermore, in the code fragment synthesis IDE, we plan to use tools such as INFER.NET to compute the likelihood of suitability of different function calls in a given context.

1.4.3 Validation using Testing and Analysis

To perform deeper semantic analysis of possible alternatives (before and after user's selection), we will deploy cloud-based service for performing parallel testing as well as static program analysis and verification of applications. Our groups have significant expertise in static analysis for a number of languages including Scala [19] and PHP [13], as well as a range of analysis techniques. These tools, together with the high-level programming model will increase the automation of code development while simultaneously improving the reliability of resulting applications.

Activity	Months 1-6	Months 7-12	Months 13-18	Months 19-24
Language	DSL design	compilation	value customization	behavior customization
Types	monorphic	generic	subtyping	
Search	dynamic programming	weighted resolution	theoretical analysis	evaluation
Validation		testing	verification	

Figure 1: Gannt chart of activities of the student

2 Activities, Milestones, Deliverables

Figure 1 shows a tentative Gannt chart of the proposed activities described in previous sections. We are describing activities of one doctoral student, so the breakdown is fairly detailed, and the activities are closely interconnected.

Our final deliverables will be:

- a domain-specific language SL for developing common classes of Smartphone applications, embedded into Scala;
- an integrated development environment for SL, including in particular functionality for synthesizing entire code fragments based on partial information such as types and test cases;
- a run-time that allows user customization of SL applications after they are already running on the smartphone.

We will release the developed tools under the BSD license.

An important output of our work will be publications and presentations describing the underlying algorithms. We have a track record in publishing in leading programming language and software engineering conferences; we plan to continue this practice with the outputs of this research.

2.1 Synergistic Activities Funded from Other Sources

In LARA research group led by Viktor Kuncak, Etienne Kneuss works on the related direction of analyzing Scala programs. He is currently funded by the first-year doctoral fellowship of the EPFL School of Computer and Communication Sciences. He is also expected to continue to be affiliated with LARA and work on the closely related topics in analysis, synthesis, and domain-specific languages in Scala.

Further funding for synergistic acitivities is subject to approval in very competitive funding schemes, including NCCR grants in the areas of cloud computing and trustworthy computer systems. A junior ERC grant in the area of software synthesis has also been submitted by Viktor Kuncak.

Two students that are graduating soon from the LARA group, Ruzica Piskac, and Philippe Suter, will likely continue as collaborators on related activities. Ruzica Piskac will start as a tenure-track faculty member at the Max-Planck Institute for Software Systems in Saarbrücken, Germany, where she will lead an independent research group.

A number of members of the LAMP group are funded by a senior ERC grant awarded to Martin Odersky. Among the goals of this grant is enabling the development of embedded domain specific languages. These developments will make it easier to develop domain-specific language for cloud-backed smartphones. Of particular common interest is IDE support for domain-specific embedded languages. Whereas the proposed research focuses on one particular DSL, we expect that the techniques and lessons learned will lead to reusable mechanisms. These mechanisms will help future DSL developers that are not programming language experts to easily introduce not only DSL constructs, but also the corresponding IDE support for them.

3 Dissemination, Standardization, and Patents

Dissemination. Our research groups pursue active dissemination activities in the form of research publications, public presentations, and released software artifacts. We will continue these activities in the context of the proposed project. Following the ICES research program guidelines, we will release our software under the BSD license.

Standardization. With our collaborators we will work on establishing joint formats for exchanging information about software artifacts and establishing community standards for reliable software. We have already successfully started such standardization and dissemination activities through the Rich Model Toolkit Initiative (`http://richmodels.epfl.ch`), which gathers 50 research groups from 20 countries. The initiative was started by Viktor Kuncak in 2009 and is funded as an EU COST Action until 2013.

Patents. We do not expect to pursue any patent activities specific to the research in this proposal, focusing instead on disseminating the results through scientific publications.

4 Impact on Teaching and Curricula

If adopted, the proposed programming model will fundamentally change the nature of programming activities, making it more social and user-oriented. Just as social networks and blogs unleashed creative potentials of larger fragments of the population than ever before, the proposed project will help bring the potentials of large populations to the programming activity, making it more accessible, more productive, and more fun than ever before.

We expect immediate impact from project results in master's level courses that study the basic foundations of the underlying technology of this proposal. This includes the course Synthesis, Analysis, and Verification that incorporates synthesis algorithms. It may also impact the Foundations of Software, which introduces type constraints in programming languages, providing additional motivation for studying declarative approaches to type system description and implementation. The project ideas will be also explored and extended through individual semester projects as bachelor and master level.

Moreover, we plan to include the compilation for the developed language into the undergraduate compiler course that we teach. We expect this to make the course more appealing to broader populations of students, which have substantial user-side experience with smartphones.

As the developed technology matures, we expect to incorporate it into development tools that students use when programming in Scala. We use Scala in all courses organized by the two research groups, at both master and bachelor level. The impact will be visible all the way down to the second year undergraduate course in advanced programming. More broadly, it will impact the entire Scala user community, as well as other programming languages and their development tools.

A Appendix

A.1 Biographies of Principal Investigators

Viktor Kuncak is an Assistant Professor at EPFL and the head of the Automated Reasoning and Analysis group. He is a member of the EPFL Thrust for Reliable Software Research (TRESOR) (`http://tresor.epfl.ch`) and the EPFL Ecocloud center (`http://ecocloud.ch/`). His recent work focuses on software synthesis, software verification, and decision procedures. He also works on static analysis, run-time checking, and constraint solving. He initiated and chairs the COST Action "Rich Model Toolkit – An Infrastructure for Reliable Computer Systems" which is funded by FP7 and involves 20 European countries. Before joining EPFL in 2007, Viktor Kuncak received his doctorate and master's degree from the Massachusetts Institute of Technology. He was a summer intern at Software Productivity Tools group in Microsoft Research, Redmond, USA, in 2002 and a visitor at the Max-Planck-Institute for Computer Science, Saarbrücken, Germany, in 2003 and 2005.

Homepage: `http://lara.epfl.ch/~kuncak`

Martin Odersky is a Professor at EPFL where he heads the programming research group. His research interests cover fundamental as well as applied aspects of programming languages. They include semantics, type systems, programming language design, and compiler construction. The main focus if his work lies in the integration of object-oriented and functional programming. His research thesis is that the two paradigms are just two sides of the same coin and should be unified as much as possible. To prove this he has experimented with a number of language designs, from Pizza to GJ to Functional Nets. He has also influenced the development of Java as a co-designer of Java generics and as the original author of the current javac reference compiler. His current work concentrates on the Scala programming language, which unifies FP and OOP while staying completely interoperable with Java and .NET.

Martin Odersky got his doctorate from ETH Zürich in 1989. He held research positions at the IBM T.J. Watson Research Center from 1989 and at Yale University from 1991. He was a professor at the Univerisity of Karlsruhe from 1993 and at the University of South Australia from 1997. He joined EPFL as full professor in 1999. He is associate editor of the Journal of Functional Programming and member of IFIP WG 2.8. He was conference chair for ICFP 2000, and program chair for ECOOP 2004 as well as ETAPS/CC 2007. He is a fellow of the ACM.

Homepage: `http://lamp.epfl.ch/~odersky`

A.2 Biography of the Student to be Funded

Thiomir Gvero is a PhD student at EPFL working with Viktor Kuncak. He works on program synthesis, software testing and their combination. His recent contributions are: interactive synthesis of the code snippets, InSynth, `http://lara.epfl.ch/w/insynth`, that resulted in a tool paper at the CAV 2011 conference. He also worked on repairing broken test cases, which was published at ICSE DEMO 2011 and ISSTA 2010. Finally, with collaborators he developed and implemented the UDITA language for test genera-

tion, http://mir.cs.illinois.edu/udita, published at the ICSE 2010 conference. He received ACM SIGSOFT Distinguished Paper Award for the ICSE 2010 publication. In 2010, he was a summer intern at the Microsoft, Redmond, where he worked with Nikolai Tillmann on the search strategy and on inferring object invariants to improve the Pex tool (http://research.microsoft.com/en-us/projects/pex). Before joining EPFL in 2009, he received Master (2007-2009) and Bachelor (2003-2007) degrees in Computer Science and Engineering from the School of Electrical Engineering of the University of Belgrade, Serbia.

Homepage: http://people.epfl.ch/gvero

A.3 Description of the Host Research Groups

The proposed research will be hosted in two collaborating laboratories: Laboratory for Automated Reasoning and Analysis (LARA), and Laboratory for Programming Methods (LAMP).

LAMP research group (http://lamp.epfl.ch) is led by Martin Odersky. It includes eight doctoral students, four postdoctoral researchers, and a number of master's students. It is assisted by a secretary and a system manager. The group is the origin and the center of the design of the Scala programming language (http://www.scala-lang.org) and the Scala compiler. Group's members have introduced a number of programming language concepts and systems, including: local type inference, nominal objects with dependent types, Generic Java (GJ), and Pizza. The group teaches the undergraduate course "Advanced topics in programming", where students learn about advanced programming techniques found in modern languages, such as Scala, as well as master's level course "Advanced Compiler Construction" and the programming language course "Foundations of Software".

LARA research group (http://lara.epfl.ch) is led by Viktor Kuncak. It was founded in 2007 and includes six doctoral students, one postdoctoral researcher, and several master's students. It is assisted by a part-time secretary and shares a system manager with LAMP. The group teaches "Compiler Construction" undergraduate course as well as a graduate course "Synthesis, Analysis, and Verification". Key activities of the group include software synthesis and software verification, as well as the enabling technology of decision procedures.

Systems and prototypes developed by the group are presented at

http://lara.epfl.ch/w/software

Particularly relevant for this proposal are synthesis related tools, summarized at

http://lara.epfl.ch/w/impro

References

[1] P. Alvaro, T. Condie, N. Conway, K. Elmeleegy, J. M. Hellerstein, and R. Sears. Boom analytics: exploring data-centric, declarative programming for the cloud. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, 2010.

[2] A. Bove, P. Dybjer, and U. Norell. A brief overview of Agda - a functional language with dependent types. In *TPHOLs*, pages 73–78, 2009.

[3] V. Castelli, L. D. Bergman, T. Lau, and D. Oblinger. Sheepdog, parallel collaborative programming-by-demonstration. *Knowl.-Based Syst.*, 23(2), 2010.

[4] H. Chafi, Z. DeVito, A. Moors, T. Rompf, A. K. Sujeeth, P. Hanrahan, M. Odersky, and K. Olukotun. Language virtualization for heterogeneous parallel computing. In *OOPSLA*, 2010.

[5] S. Chatterjee, S. Juvekar, and K. Sen. SNIFF: A search engine for java using free-form queries. In *FASE '09*, pages 385–400, 2009.

[6] D. Gay, P. Levis, J. R. von Behren, M. Welsh, E. A. Brewer, and D. E. Culler. The nesC language: A holistic approach to networked embedded systems. In *PLDI*, 2003.

[7] M. Gligoric, T. Gvero, V. Jagannath, S. Khurshid, V. Kuncak, and D. Marinov. Test generation through programming in UDITA. In *International Conference on Software Engineering (ICSE)*, 2010.

[8] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *POPL*, 2011.

[9] T. Gvero, V. Kuncak, and R. Piskac. Interactive synthesis of code snippets. In *Computer Aided Verification (CAV) Tool Demo*, 2011.

[10] M. Hofmann. IGOR2 - an analytical inductive functional programming system: tool demo. In *Proceedings of the 2010 ACM SIGPLAN workshop on Partial evaluation and program manipulation*, PEPM '10, pages 29–32, 2010.

[11] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *ICSE '05*, pages 117–125, 2005.

[12] C. E. Killian, J. W. Anderson, R. Braud, R. Jhala, and A. Vahdat. Mace: language support for building distributed systems. In *PLDI*, 2007.

[13] E. Kneuss, P. Suter, and V. Kuncak. Runtime instrumentation for precise flow-sensitive type analysis. In *International Conference on Runtime Verification*, 2010.

[14] A. Köksal, V. Kuncak, and P. Suter. Constraints as control. In *POPL*, 2012.

[15] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman. Jungloid mining: helping to navigate the api jungle. In *PLDI*, 2005.

[16] N. Sahavechaphan and K. Claypool. Xsnippet: mining for sample code. In *OOPSLA*, 2006.

[17] A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, and V. A. Saraswat. Combinatorial sketching for finite programs. In *ASPLOS*, 2006.

[18] R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theor. Comput. Sci.*, 9:67–72, 1979.

[19] P. Suter, A. S. Köksal, and V. Kuncak. Satisfiability modulo recursive programs. In *Static Analysis Symposium (SAS)*, 2011.

[20] S. Thummalapenta and T. Xie. PARSEWeb: a programmer assistant for reusing open source code on the web. In *ASE '07*, pages 204–213, 2007.