Intermediate Scientific Report
SNF Project no. 200020-113342
"Analyzing, capturing and taming software change"

November 7, 2007

a) Summary of results

This project focuses on the design and implementation of programming language mechanisms and concepts to enable and control extensibility of complex software systems. Some of the most significant results include:

- Changeboxes— we implemented a prototype to encapsulate and express changes to complex software systems. A Changebox is a programming language construct that supports the developer when making changes and supports the coexistence of multiple versions of a running system.
- Scoped Reflection— we implemented a reflection framework that supports a flexible and controlled instrumentation of a software system at runtime. Using our framework, we built flexible dynamic analysis tools and a pluggable type system for a dynamically typed language.
- Object Flow Analysis— we developed a novel object-centric dynamic analysis technique that tracks how objects are passed around a system at runtime. This type of information provides a new perspective of what how object-oriented systems behave at runtime. We have applied this analysis technique to detect runtime dependencies between software features and to support debugging activities.
- Evolution Analysis— we defined analysis techniques and extensions to our reverse engineering platform to model the developers and investigate their role in the life-cycle of a software system. We also implemented extensions to a development environment to integrate reverse engineering results directly in the environment where the developer works on a system's source code.

Results

We present the results obtained during the period from 2006-10-01 to 2007-09-30.

Changeboxes

Software systems need to constantly adapt to new and changing requirements to remain useful. We developed a programming language construct called a Changebox to encapsulate, manage, analyze and exploit changes to software systems. Changeboxes make change explicit, thus enabling a software developer to manipulate change more effectively than is currently possible. Changeboxes package incremental modifications to complex software systems. They are used to express low-level (syntactic) and high-level (semantic) changes. They represent units of modification that can be replayed, or selectively applied to yield different versions of software entities that may coexist in a single system.

A prototype of Changeboxes has been developed as part of Pascal Zumkehr's masters thesis [Zum07]. This work builds on our Reflection framework in order to control the scope of visibility of changes.

As a case study, we have used Changeboxes in the context of web-applications to support continuous development in a running system without disturbing currently active user sessions. We presented our research on Changeboxes at ASWEC 2007 (18th Australian Conference on Software Engineering — Melbourne).

Scoped Reflection

Structural and behavioural reflection are well-known techniques to enable run-time change, but they can also break a running system in catastrophic ways if they are applied without discipline. Scoped Reflection provides a degree of control over which reflection mechanisms are available at what time and to which clients.

To support our work on Scoped Reflection, we have implemented a Reflection framework which extends structural reflection to support sub-method elements [DDLM07]. We used sub-method reflection as a basis to for a second, improved implementation of partial behavioral reflection. We showed the usefulness of sub-method reflection to support pluggable type systems [HDN07]. As part of his masters thesis, Niklaus Haldiman based his implementation of TypePlug, a pluggable type system for Smalltalk, on sub-method reflection [Hal07].

We also outlined the role of behavioral reflection in a general context of implementing tools for dynamic analysis. The developers of these tools are faced with choosing from the many approaches to gathering runtime information. Currently the task of building dynamic analysis tools require detailed knowledge of the target programming language or virtual machine. We addressed this problem by outlining how behavioral reflection provides the basis for defining a higher level of abstraction to support building of dynamic analysis tools. [DGL06].

Object Flow Analysis

In contast to traditional dynamic analysis techniques, which characterize a system's behavior purely by capturing message sends, we have developed an object-centric dynamic analysis technique which we call Object Flow Analysis. As the basis of our work, we have implemented an Object Flow Analysis framework to capture the flow of object references through the running system. So far we have applied Object Flow Analysis to two different problem domains: (1) debugging and (2) feature dependency analysis.

Traditional debuggers support the analysis of an execution context when a error is detected in a system. However, the defect which is responsible for the error may have occured at an earlier point in the execution time and thus may no longer visible in the current execution context. We addressed this problem by applying our Object Flow Analysis technique to track runtime object references. Our approach provides a more object-centric (rather than stack-based) navigation of a system's behavior.

While the problem of locating features in object-oriented programs has been widely studied, runtime dependencies between features are less well understood. We applied our Object Flow Analysis technique to detect runtime dependencies between features of a software system. Exposing dependencies between features is essential for maintenance and evolution of a software system as changes to the source code that contributes to the behavior of one feature may inadvertently break the behavior of other features [LGN07].

To support Object Flow Analysis and dynamic analysis in general, we extended our existing Moose reengineering environment. We defined a meta-model for dynamic information called Dynamix which we described in [Gre07]. We consequently developed Dynamoose, a tool that extends the Moose environment to enable analysis of fine-grained dynamic information of Object Flow Analysis [LDGN06] and different feature analyses [Gre07]. We have also reported on our experience of fast prototyping tools based on the Mondrian visualization engine [LKG07].

We are currently investigating various approaches to supporting unit testing and debugging. Furthermore, we are studying how to implement Object Flow tracing directly in the virtual machine.

Evolution Analysis

Our work on evolution analysis focuses on three major tracks: (1) the role of the developer in software development and system evolution, (2) the development environment and (3) facilitating reverse engineering by investigating ways to ease the task of parsing the wide range of programming languages in use today so that we can model them and apply our reverse engineering tools on them.

We emphasize the role of developers when analyzing a software system. Knowledge of how the developers collaborate, and how their responsibilities are distributed over the software artifacts is a valuable source of information when reverse engineering a system. We also investigate the role of developers with respect to the development of features (*i.e.* dynamic units of behavior) to complement the static perspective of a system [GGD07].

To extend a software system, a developer requires in-depth knowledge of the inner structure of the system. There are many program comprehension techniques based on reverse-engineered information about software system. This information is usually manipulated by reverse engineering tools that are distinct from a software developer's working environment, namely the IDE. We have built extensions to the IDE so that we can exploit the results of reverse engineering analysis directly in the IDE. Our IDE enhancements support program comprehension by making reverse engineering analysis results readily available to the developer where and when she actually works. The implementation of our IDE enhancements is built using our Reflection framework [RN07].

Before we can use any reengineering tool to analyze the evolution of a software system we must reverse engineer that system. Thus we need to build a model of the system. To model any system we need a parser for the programming language it is implemented in so hat we can translate source code into a model. Writing parsers is a difficult and time-consuming task and there are so many languages and versions of languages today that it is not possible to support all of them.

We built an application that generates parsers based on mapping examples. A mapping example is a section in the source code to which we assign an element in our target model. Based on these examples, our application builds grammars and generates a parser. This approach is flexible enough to work with a software system written in an arbitrary programming language [NKG+07].

Staff contributions

- Lukas Renggli is in the second year of the PhD. He has reported on his experience of developing the Seaside web application framework that features an enhanced component model and is based on continuations [DLR07]. He worked on building a dynamic meta-described environment called Magritte [RDK07], and he used this environment to construct a highly expressive and configurable content management system that received the 3rd prize at ESUG 2007 Technology Innovation Awards [Ren07]. He recently started to distill these experiences and investigate how to build support for developing domain specific languages.

- Adrian Lienhard is in the third year of his PhD. He is investigating runtime information flows in object-oriented systems. He matured his Object Flow Analysis through which he can identify how objects are referenced and passed through the system at runtime [LDGN06]. He used Object Flow Analysis to develop an application for detecting runtime dependencies between features [LGN07].

- Adrian Kuhn is in the second year of his PhD. He is investigating extensions to programming languages to express the notion of collective behavior. Collective behavior associates custom behavior with collection instances, based on the type of its elements. He has implemented a proof-of-concept implementation of collective behavior [Kuh07a].

- David Roethlisberger is in the second year of his PhD. His focus is to improve development environments to ease software maintenance and evolution. He used reflective techniques, such as sub-method reflection, to integrate dynamic information about a program under development into the development environment [RN07, RGN07]. He built a tool suite to demonstrate his approach [RGL07].

Changes to the research plan

No major changes have occurred in the research plan.

Important events

- Marcus Denker presented a *tool demonstration* of our implementation of the Reflectivity System at at Dyla07 (3rd Workshop on Dynamic Languages and Applications) and ECOOP 2007 (21st European Conference on Object-Oriented Programming) in Berlin, Germany.
- Oscar Nierstrasz was an *invited speaker* at the following events:
  - Keynote Speaker on “Modeling Change as a First-Class Entity” at ASWEC 2007 (18th Australian Conference on Software Engineering — Melbourne, April 10-13, 2007)
  - JUGS event: “Object-oriented Reengineering Patterns — an Overview” (Technopark Zurich, Jan. 25, 2007).
- Tudor Gîrba and Adrian Kuhn *co-organized* FAMOOSr (Workshop on FAMIX and Moose in Reengineering - co-located with TOOLS 2007).
- Orla Greevy was *Program Chair* of PCODA 2007 (3rd Workshop on Program comprehension through dynamic analysis) colocated with WCRE in Vancouver, CA [ZHLG07].

b) Publications

Published papers are annexed to this report. They are all available electronically as PDF files at the following url:

`http://www.iam.unibe.ch/~scg/cgi-bin/scgbib.cgi?snf07`

Please note that theses and student projects are *not* included with this report, but are nevertheless available electronically from the above URL.

Papers published in the context of the RECAST project are also not included with this report. They have been previously submitted with the final report for RECAST. Electronic versions are available at:

`http://www.iam.unibe.ch/~scg/cgi-bin/scgbib.cgi?recast07`

Published papers

[DGL06] Marcus Denker, Orla Greevy, and Michele Lanza. Higher abstractions for dynamic analysis. In *2nd International Workshop on Program Comprehension through Dynamic Analysis (PCODA 2006)*, pages 32–38, 2006.

[DLR07] Stéphane Ducasse, Adrian Lienhard, and Lukas Renggli. Seaside: A flexible environment for building dynamic web applications. *IEEE Software*, 24(5):56–63, 2007.

[GGD07] Orla Greevy, Tudor Gîrba, and Stéphane Ducasse. How developers develop features. In *Proceedings of 11th European Conference on Software Maintenance and Reengineering (CSMR 2007)*, pages 256–274, Los Alamitos CA, 2007. IEEE Computer Society.

[Gre07] Orla Greevy. Dynamix — a meta-model to support feature-centric analysis. In *Proceedings of FAMOOSr 2007 (1st International Workshop on FAMIX and Moose in Reengineering)*, June 2007.

[GWN07] Markus Gaelli, Rafael Wampfler, and Oscar Nierstrasz. Composing tests from examples. *Journal of Object Technology*, 6(9):71–86, October 2007.

[HHD07] Michael Haupt, Robert Hirschfeld, and Marcus Denker. Type feedback for bytecode interpreters. In *Proceedings of the Second Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems (ICOOOLPS'2007)*, pages 17–22, July 2007.

[Kuh07a] Adrian Kuhn. Collective behavior. In *Proceedings of 3rd ECOOP Workshop on Dynamic Languages and Applications (DYLA 2007)*, August 2007.

[Kuh07b] Adrian Kuhn. Rbcrawler — a visual navigation system for Smalltalk's Refactoring Browser. European Smalltalk User Group Innovation Technology Award, August 2007.

[LDGN06] Adrian Lienhard, Stéphane Ducasse, Tudor Gîrba, and Oscar Nierstrasz. Capturing how objects flow at runtime. In *Proceedings International Workshop on Program Comprehension through Dynamic Analysis (PCODA 2006)*, pages 39–43, 2006.

[LGN07] Adrian Lienhard, Orla Greevy, and Oscar Nierstrasz. Tracking objects to detect feature dependencies. In *Proceedings International Conference on Program Comprehension (ICPC 2007)*, pages 59–68, Washington, DC, USA, June 2007. IEEE Computer Society.

[LKG07] Adrian Lienhard, Adrian Kuhn, and Orla Greevy. Rapid prototyping of visualizations using mondrian. In *Proceedings IEEE International Workshop on Visualizing Software for Understanding (Vissoft 2007)*, pages 67–70, June 2007.

[NKG$^+$07] Oscar Nierstrasz, Markus Kobel, Tudor Gîrba, Michele Lanza, and Horst Bunke. Example-driven reconstruction of software models. In *Proceedings of Conference on Software Maintenance and Reengineering (CSMR 2007)*, pages 275–286, Los Alamitos CA, 2007. IEEE Computer Society Press.

[RDK07] Lukas Renggli, Stéphane Ducasse, and Adrian Kuhn. Magritte — a meta-driven approach to empower developers and end users. In Gregor Engels, Bill Opdyke, Douglas C. Schmidt, and Frank Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *LNCS*, pages 106–120. Springer-Verlag, September 2007.

[Ren07] Lukas Renggli. Pier — the meta-described content management system. European Smalltalk User Group Innovation Technology Award, August 2007.

[RGD07] Stefan Reichhart, Tudor Gîrba, and Stéphane Ducasse. Rule-based assessment of test quality. *Journal of Object Technology*, 6(9):231–251, October 2007.

[RGL07] David Röthlisberger, Orla Greevy, and Adrian Lienhard. Feature-centric environment. In *Proceedings IEEE International Workshop on Visualizing Software for Understanding (Vissoft 2007) (tool demonstration)*, 2007.

[RN07] David Röthlisberger and Oscar Nierstrasz. Combining development environments with reverse engineering. In *Proceedings of FAMOOSr 2007 (1st International Workshop on FAMIX and Moose in Reengineering)*, 2007.

Theses and Student projects

[Gre07] Orla Greevy. *Enriching Reverse Engineering with Feature Analysis*. PhD thesis, University of Berne, May 2007.

[Hal07] Niklaus Haldimann. TypePlug — pluggable type systems for Smalltalk. Master's thesis, University of Bern, April 2007.

[Mar06] Philippe Marschall. Persephone: Taking Smalltalk reflection to the sub-method level. Master's thesis, University of Bern, December 2006.

[Mey06] Michael Meyer. Scripting interactive visualizations. Master's thesis, University of Bern, November 2006.

[Rei07] Stefan Reichhart. Assessing test quality — TestLint. Master's thesis, University Bern, April 2007.

[Zum07] Pascal Zumkehr. Changeboxes — modeling change as a first-class entity. Master's thesis, University of Bern, February 2007.

Selected RECAST publications

[GDG06] Orla Greevy, Stéphane Ducasse, and Tudor Gîrba. Analyzing software evolution through feature views. *Journal of Software Maintenance and Evolution: Research and Practice (JSME)*, 18(6):425–456, 2006.

[KDG07] Adrian Kuhn, Stéphane Ducasse, and Tudor Gîrba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243, March 2007.

c) Publications in press

Publications to appear

[DDLM07] Marcus Denker, Stéphane Ducasse, Adrian Lienhard, and Philippe Marschall. Sub-method reflection. *Journal of Object Technology*, 6(9):231–251, October 2007.

[HDN07] Niklaus Haldiman, Marcus Denker, and Oscar Nierstrasz. Practical, pluggable types. In *International Conference on Dynamic Languages (2007)*, 2007. To appear.

[LDG07] Adrian Lienhard, Stéphane Ducasse, and Tudor Gîrba. Object flow analysis — taking an object-centric view on dynamic analysis. In *International Conference on Dynamic Languages (2007)*, 2007. To appear.

[RGN07] David Röthlisberger, Orla Greevy, and Oscar Nierstrasz. Feature driven browsing. In *International Conference on Dynamic Languages (2007)*, 2007. To appear.

[RN07] Lukas Renggli and Oscar Nierstrasz. Transactional memory for Smalltalk. In *International Conference on Dynamic Languages*, 2007. To appear.

[ZHLG07] Andy Zaidman, Abdelwahab Hamou-Lhadj, and Orla Greevy. Workshop on program comprehension through dynamic analysis (pcoda). In *Proceedings of IEEE 14th Working Conference on Software Maintenance and Reengineering (WCRE)*, pages 315–315, November 2007.