

# Heterogeneous Proof-Carrying Components

Final Report

Peter Müller and Bertrand Meyer

ETH Zurich, Switzerland

{Peter.Mueller, Bertrand.Meyer}@inf.ethz.ch

May 2009

## Abstract

The goal of the Heterogeneous Proof-Carrying Components (HPCC) project was to develop the technology for a particular form of trusted component: Proof-Carrying Components. The results of the project show that it is feasible to produce a Proof-Carrying Component by translating a proof in the source language into a bytecode proof. This report presents the results of the HPCC project.

## 1 Introduction

The goal of the Heterogeneous Proof-Carrying Components (HPCC) project was to develop the technology for a particular form of trusted component: Proof-Carrying Components. The guarantee of quality of these components is a mathematical proof, machine-checkable, that the component satisfies specific properties.

The main contributions of the project were:

- A bytecode logic for .NET CIL, which handles unstructured control flow with jumps. This logic has been published at the BYTECODE workshop [2].
- A Hoare-style logic for Eiffel. The novelties of this logic are the treatment of exception handling (using the `rescue` mechanism), once routines, and multiple inheritance. These results have been published at the TOOLS 2009 conference [11].
- A logic for C# and Java, which includes abrupt termination using `break`, `try-catch`, and `try-finally` instructions.
- A proof-transforming compiler (PTC) that translates proofs from a common subset of C# and Java to CIL. The contributions of this PTC are translation functions that handles `break`, `try-catch`, and `try-finally` instructions. The results of the PTC and the logic for C# and Java have been published at the SAVCBS workshop [8].
- A proof-transforming compiler for Eiffel that handles exception handling and maps Eiffel multiple inheritance to CIL single inheritance. The results have been published at the TOOLS 2008 conference [13].
- An implementation of the platform, which includes a proof checker and a proof-transforming compiler.

**Project Plan.** We had initially proposed three PhD topics of three years each:

- *Doctoral Project A - Common Proof-Carrying Component Platform*
- *Doctoral Project B - Proof-Transforming Compiler*
- *Doctoral Project C - Integrated Validation*

We had to deviate from that plan for three reasons. First, we got funding for only two PhD students for a duration of three years each. Following the recommendation of the reviewers, we focused on the foundations of Proof-Carrying Components and dropped Doctoral Project C. Second, one of the hired PhD students, Xiaohui Jin, got severe health problems and had to leave to get treatment abroad. To compensate for this loss, we extended the remaining PhD project by one year in order to develop a working infrastructure. Nevertheless, we had to drop work package *A.3 Deployment Format*, and instead of developing a dedicated contract language, we used first order logic. Third, it turned out that the Java logic we had planned to use was unsound, and there was no suitable logic for Eiffel. Therefore, we replaced work package *B.3: Optimizing Compilers* by a new work package *B.3a: Source Logics*. However, the European *Mobius* project, in which we participate, covered optimizing Proof-Transforming Compilers [3, 14]. In summary, our work packages were:

1. WP A.1: Program Logic (Bytecode)
2. WP A.4: Proof Checker
3. WP B.1: Specification Transformation
4. WP B.2: Proof Transformation
5. WP B.3a: Source Logics
6. WP B.4: Implementation

## 2 Results

### 2.1 Common Proof-Carrying Component Platform

#### 2.1.1 Program Logic

The bytecode logic we developed is a Hoare-style program logic. The main contribution of this logic is the treatment of unstructured control flow with jumps. The logic handles object-oriented features such as inheritance, dynamic method binding, and object structures with destructive updates. The logic is sound and complete. The results of the program logic work package are described by Müller et al. [2, 1].

#### 2.1.2 Proof Checker

The proof checker was developed in Isabelle. We implemented the proof checker as a verification condition generator (VCGen). Given a bytecode proof, this VCGen generates a list of conditions (proof obligations) that have to be proven to show that the proof is correct. Given a proof script, and the proof obligations, the proof checker applies the script. If all proof obligations are proven, the proof is correct, and the component can be executed. Otherwise the proof is incorrect, and the component is rejected. More details are described in a Master's thesis [5]

### 2.2 Source Logic

#### 2.2.1 Logic for C# and Java.

The logic for C# and Java is a Hoare-style logic. The novelty in this logic was the formalization of `break` instructions as well as `try-catch` and `try finally`. The logic handles object-oriented features such as inheritance, dynamic method binding, and abrupt termination. For more details about the logic for C# and Java see [9, 8].

### 2.2.2 Logic for Eiffel.

We have developed an operational and axiomatics for Eiffel. We have formalized the Eiffel exception handling mechanism. This formalization discovered problems in the design of this mechanism. We have improved it, and the changes will be incorporated in the next version of Eiffel. The semantics is sound and complete. For more details about the semantics for Eiffel see [11, 10].

## 2.3 Proof-Transforming Compiler

### 2.3.1 Specification Transformations

The specification translator translates contracts into first-order logic. Translating C# and Java contracts was straightforward since the translation does not change the type structure of the program. In the case of Eiffel, the impedance mismatch between Eiffel and CIL, which does not directly support important Eiffel mechanisms such as multiple inheritance made the translation challenging. To be able to translate contracts, we have used deeply-embedded expressions in the contracts. The main contribution in this work package was a contract translation function from Eiffel to CIL that handles multiple inheritance. The translation has been proved sound, and the proof has been formalized in Isabelle. For more details about the specification transformation see [12, 13].

### 2.3.2 Proof Translation

**C# and Java Proof-Transforming Compiler.** The challenge in the translation of a common subset of C# and Java to CIL was produced by the inclusion of `break`, `try-catch`, and `try-finally` instructions, especially because we needed to handle Java's complex exception tables. The main problem to solve was the formalization of the translation, and the proof of soundness. For more details about this proof-transforming compiler see [9, 8].

**Eiffel Proof-Transforming Compiler.** The translation of the Eiffel PTC maps Eiffel exception handling using rescue blocks to CIL exception handling using `try-catch` instructions. Furthermore, the Eiffel PTC translates once routines using static fields. The translation is sound; for more details see [12, 13].

### 2.3.3 Implementation

To show the feasibility of Proof-Carrying Components, we implemented a PTC for a subset of Eiffel. The compiler takes a proof in an XML format, and produces the bytecode proof. The compiler is integrated into EiffelStudio. The result produced by the compiler is embedded into Isabelle. Using the proof checker (Section 2.1.2) one can check whether the component is correct or not. This implementation is part of the result of several Master's theses at ETH [5, 15, 6, 7, 4].

## 3 Future Work

The HPCC project has shown that proofs in object-oriented programs can be translated to bytecode proofs. During this project, new tools such as Spec# have been developed to verifying object-oriented programs. The goal of these tools is to automatically verify object-oriented programs.

As future work, we plan to contribute to the development of these tools and integrate them with a proof-transforming compiler tool. Furthermore, we plan to investigate how to produce certificates from the proof obligations produced by such tools. Also, we will develop a case study to analyze the feasibility of the use of proof transformation in real applications.

## References

- [1] F. Y. Bannwart and P. Müller. A logic for bytecode. Technical Report 469, ETH Zurich, 2004.
- [2] F. Y. Bannwart and P. Müller. A Logic for Bytecode. In F. Spoto, editor, *Bytecode Semantics, Verification, Analysis and Transformation (BYTECODE)*, volume 141(1) of *ENTCS*, pages 255–273. Elsevier, 2005.
- [3] G. Barthe, T. Rezk, and A. Saabas. Proof obligations preserving compilation. In *Third International Workshop on Formal Aspects in Security and Trust, Newcastle, UK*, pages 112–126, 2005.
- [4] M. Guex. Implementing a Proof-Transforming Compiler from Eiffel to CIL. Technical report, ETH Zurich, 2006.
- [5] B. Hauser. Embedding Proof-Carrying Components into Isabelle. Master’s thesis, ETH Zurich, 2009.
- [6] M. Hess. Integrating Proof-Transforming Compilation into EiffelStudio. Master’s thesis, ETH Zurich, 2008.
- [7] H. Karahan. Proof-transforming compilation of Eiffel contracts. Technical report, ETH Zurich, 2008.
- [8] P. Müller and M. Nordio. Proof-Transforming Compilation of Programs with Abrupt Termination. Technical Report 565, ETH Zurich, 2007.
- [9] P. Müller and M. Nordio. Proof-transforming compilation of programs with abrupt termination. In *SAVCBS ’07: Proceedings of the 2007 conference on Specification and verification of component-based systems*, pages 39–46, 2007.
- [10] M. Nordio, C. Calcagno, B. Meyer, and P. Müller. Reasoning about Function Objects. Technical Report 615, ETH Zurich, 2009.
- [11] M. Nordio, C. Calcagno, P. Müller, and B. Meyer. A Sound and Complete Program Logic for Eiffel. In M. Oriol, editor, *TOOLS-EUROPE 2009*, Lecture Notes in Business and Information Processing, 2009. To appear.
- [12] M. Nordio, P. Müller, and B. Meyer. Formalizing Proof-Transforming Compilation of Eiffel programs. Technical Report 587, ETH Zurich, 2008.
- [13] M. Nordio, P. Müller, and B. Meyer. Proof-Transforming Compilation of Eiffel Programs. In R. Paige and B. Meyer, editors, *TOOLS-EUROPE*, Lecture Notes in Business and Information Processing. Springer-Verlag, 2008.
- [14] A. Saabas and T. Uustalu. Program and proof optimizations with type systems. *Journal of Logic and Algebraic Programming*, 77(1–2):131–154, 2008.
- [15] J. Tschannen. Automatic Verification of Eiffel Programs. Master’s thesis, ETH Zurich, 2009.