Run-time Manipulation of Programs in a Statically-Typed Language

Sašo Greiner
University of Maribor, Faculty of Electrical Engineering and Computer Science,
Smetanova 17, 2000 Maribor, Slovenia
E-mail: saso.greiner@uni-mb.si,
http://labraj.uni-mb.si/disertacija.pdf

Thesis Summary

Keywords: programming languages, metaprogramming, reflection

Received: March 31, 2009

This article is an extended abstract of a doctoral dissertation on metaprogramming and programming language design. A metaprogramming model is studied and implemented in a statically–typed pure object–oriented programming language Zero. The object model of language is based on closures which enables metaprogramming model to achieve a high degree of dynamic manipulation, normally only found in dynamically–typed languages. Metaprogramming in Zero is based on safely–typed structural and behavioural reflection.

Povzetek: Prispevek predstavlja doktorsko disertacijo s področja načrtovanja programskih jezikov in metaprogramiranja.

1 Introduction

Metaprogramming [6] is a key programming language feature in implementation of today's rapidly growing enterprise software systems. Metaprogramming allows manipulation of program behaviour and structure during program execution. This is vital for software that requires a high degree of availability and scalability. The concept is usually found in dynamic languages, such as Smalltalk [3], Lisp [7, 1], and Self [8]. In the dissertation we designed a metaprogramming model applicable for a statically–typed language. We developed the language Zero [4] which is a pure object–oriented programming language allowing structural and behavioural manipulation of programs at run–time.

2 The language Zero

The language Zero is built on top of language Z_0 [5]. Zero is a statically–typed language which makes execution more efficient than with dynamic languages and less fallible in terms of typing. Zero enables a high degree of application manipulation at run–time as it supports both behavioural and structural metaprogramming. In other words, changing the functionality of an application may be addressed by modifying its behaviour and structure when application is already running. The metaprogramming model of language Zero is based on pure object–orientation. That is, all values including control structures and methods in a program, are objects. Such pure object–oriented model enables efficient implementation of the metaprogramming core in the language. The most important aspect of such a representation is that all dynamic changes can be type–checked at run–time. This is vital as maintaining a program in a type–safe state is mandatory for statically–typed languages. The metaprogramming model of Zero is based on metaclasses. Metaclasses provide introspective features, such as obtaining information about classes, methods, and parameter types, as well as dynamic features for changing structural and behavioural properties of a running program. There are 3 main metaclasses in Zero: `Class`, `Method`, and `Closure`. Metaclass `Class` represents run-time class objects. A run–time method is represented my metaclass `Method`. Methods themselves are based on metaclass `Closure` which serves as the fundamental class of all control structures.

The metaprogramming model allows inspection of running programs as well as their manipulation. The latter includes decomposition of existing functionality and construction of a new one. By allowing changes to programs at run–time it becomes unnecessary for the programs to shut down and recompile. Behavioural reflection in Zero is realised by handlers, which are in fact method objects that may be attached to closures. Handlers resemble aspect–oriented programming (AOP) [2], where attached method objects may be viewed as advices. A join–point, a spot where program behaviour may be extended, is always a closure in Zero.

We demonstrate practical cases where manipulating program structure and behaviour may be used to achieve that running programs meet the new requirements. Often enough, certain parameters only become available at run–time. The metaprogramming model of Zero allows such programs to be dinamically restructured and reconfigured

taking these new parameters into account.
The Zero metaprogramming model works on instance and class levels. This means program structure and behaviour may address only a particular instance or a class and consequently all instances of this class.
Metaprogramming in Zero is used for fine–grain manipulation as well as for modifying large structures. Fine–grain manipulation works with closures which are basic building blocks of control structures, such as loops and selection statements, and method bodies. Modifying large structures such as replacing entire methods and superclasses is based on signature compatibility.

3 Conclusion

We have designed and developed a statically–typed object–oriented programming language Zero which allows dynamic changes of program structure and behaviour. Run–time changes of running programs are addressed with a metaprogramming model based on metaclasses which ensure that changes applied do not cause typing errors. Safe method and class replacements are based on signature compatibility. The metaprogramming model of language Zero allows fine–grain tuning of programs with the use of closures as building blocks of methods and all control structures. A more rigid tuning is achieved by replacing entire methods and classes in class hierarchies. As closures are basic blocks of programs, all parts of a program may be modified either by changing the behaviour or their entire structure.

Acknowledgement

The work on dissertation was supervised by prof. dr. Janez Brest and prof. dr. Viljem Žumer.

References

[1] Stanley Jefferson and Daniel P. Friedman. A simple reflective interpreter. *Lisp and Symbolic Computation*, 9(2-3):181–202, 1996.

[2] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. *Lecture Notes in Computer Science*, 2072:327–355, 2001.

[3] Wilf R. LaLonde and John R. Pugh. *Inside Smalltalk Volume I*. Prentice-Hall Internation, Inc., 1990.

[4] Sašo Greiner, Viljem Žumer, Janez Brest. Zero – a blend of static typing and dynamic metaprogramming. *Comput. Lang. Syst. Struct., 35(3):241–251*, 2009.

[5] Sašo Greiner, Damijan Rebernak, Janez Brest, and Viljem Žumer. Z_0 – a tiny experimental language. *SIGPLAN Not.*, 40(8):19–28, 2005.

[6] Diomidis Spinellis. Rational metaprogramming. *IEEE Softw.*, 25(1):78–79, 2008.

[7] Guy Steele. Common lisp: The language. *Digital Equipment Corporation*, 1984.

[8] David Ungar and Randall B. Smith. Self: The power of simplicity. *OOPSLA'87*, 4(8):227–242, 1987.