Practical Ownership Control in Programming Languages

Alex Potanin
`alex@mcs.vuw.ac.nz`

Victoria University of Wellington, New Zealand

Abstract. This research abstract outlines the work I plan to do as my PhD study. In particular, I propose to devise a practical way of integrating ownership control into existing programming languages in a way that will help with adoption of ownership in the general programming community.

1 Introduction

An object is *aliased* whenever there is more than one pointer referring to that object [10]. Aliasing can cause a range of difficult problems within object-oriented programs, because one referring object can change the state of the aliased object, implicitly affecting all the other referring objects [14,2]. Aliasing is endemic and unavoidable in object-oriented programming languages, as any assignment statement may cause an extra alias to be created. To deal with such problems, object instance encapsulation has been widely studied in literature.

Islands, confinement, and ownership are all essentially forms of object encapsulation [13]. All these schemes are attempts to establish an encapsulation *boundary* that protects some objects *inside* the boundary from direct access by other objects *outside* that boundary. Where these proposals differ from earlier programming language encapsulation and module systems is that they restrict access to objects at runtime: that is, they constrain values of pointers or references to objects in object-oriented systems, rather than merely accesses to field and method names.

My PhD started off with analysing object graphs to estimate the role object encapsulation plays in modern real world programming. After conducting a number of studies using a snapshot query-based debugger developed as part of my Honours project [15], we were able to demonstrate that object-oriented programs do in fact exhibit symptoms of encapsulation in practice, and that proposed models of uniqueness, ownership, and confinement can usefully describe the aliasing structures of object-oriented programs [15,20,16,17].

2 Ownership Generic Java (OGJ)

After reviewing the literature for the work on object instance encapsulation, two main approaches to dealing with aliasing emerged: significantly modifying a language to allow ownership support [9,2,7], or enforcing coding conventions within an existing programming language [6,12,1,3]. Unfortunately, the former approaches do not take the introduction of generics in modern programming languages into account [4,11], and the latter add it in an ad-hoc fashion.

Investigating this further, led us to attempt the introduction of ownership control as an integral part of the already existing generic mechanism in the upcoming releases of languages such as Java and C#. We found this approach to work and we developed a compiler extension, called Ownership Generic Java (OGJ), that supports generics and ownership that we hope will provide a novel and practical way of making it possible for ownership support in the programming languages to be adopted by programmers [18,19].

Why would we want ownership and generic types combined? Consider for example a *box* as a kind of object. In any object-oriented language we are allowed to say: "this is a box" (meaning any box). In a language with generics, we are allowed to say: "this is a box of books" (denoting a box of books, but not containing birds). In a language with ownership parameterisation, we are allowed to say: "this is my box" or "these are library books". Combining ownership and generics naturally allows us to say: "this is my box of library books", not birds and not my personal books. Ownership works exceptionally well with genericity, both in theory, practice, and implementation, as we describe elsewhere [19].

3 Future Work

After setting a good foundation for my PhD work in my first year, I plan to spend the coming year or more expanding upon OGJ. While we have a complete proof of OGJ providing confinement, we are still developing a model to support the proof of the ownership containment invariant which will formally demonstrate that OGJ supports ownership.

Additional extensions to both formal and practical model developed around OGJ include support for deep ownership, external uniqueness and possibly more [5,8,3,6].

After the development of the formal and practical side of OGJ is completed, whether with a successful support for the extensions planned or with a demonstration that generics are not sufficient without major changes to a programming language to fit these schemes, we plan to go back to using our snapshot query-based debugger. We plan to conduct a second extensive study of a corpus of heap snapshots, now with programs developed using OGJ to see if our changes did make it possible for aliasing to be controlled. It would require development of a number of large programs using OGJ, which we may acheive by offerring OGJ to programmers willing to integrate it into their projects. Due to backwards compatibility of OGJ syntax (in fact - its syntax is exactly the same as Generic Java, this is one of the major advantages of our approach), we don't expect this to cause much disruptions to the coding tasks facing the developers.

References

1. J. Aldrich, V. Kostadinov, and C. Chambers. Alias annotations for program understanding. In *ACM Conference on Object-Oriented Programming Languages, Applications, Languages, and Systems (OOPSLA)*, November 2002.
2. Boris Bokowski and Jan Vitek. Confined types. In *Proceedings of Conference on Object-Oriented Programming, Languages, and Applications*. ACM Press, 1999.
3. Chandrasekhar Boyapati, Barbara Liskov, and Liuba Shrira. Ownership types for object encapsulation. In *ACM Symposium on Principles of Programming Languages (POPL)*, January 2003.
4. Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. Making the future safe for the past: Adding Genericity to the Java programming language. In *ACM Conference on Object-Oriented Programming Languages, Applications, Languages, and Systems (OOPSLA)*, October 1998.
5. Dave Clarke. *Object ownership and containment*. PhD thesis, School of Computer Science and Engineering, University of New South Wales, Australia, 2002.
6. Dave Clarke and Sophia Drossopoulou. Ownership, encapsulation, and the disjointness of type and effect. In *ACM Conference on Object-Oriented Programming Languages, Applications, Languages, and Systems (OOPSLA)*, 2002.
7. Dave Clarke, Michael Richmond, and James Noble. Saving the world from bad beans: Deployment-time confinement checking. In *ACM Conference on Object-Oriented Programming Languages, Applications, Languages, and Systems (OOPSLA)*, October 2003.
8. David Clarke and Tobias Wrigstad. External uniqueness is unique enough. In Luca Cardelli, editor, *European Conference on Object-Oriented Programming (ECOOP)*, volume 2473 of *Lecture Notes In Computer Science*, pages 176–200, Darmstadt, Germany, July 2003. Springer-Verlag.
9. John Hogg. Islands: Aliasing protection in object-oriented languages. In *ACM Conference on Object-Oriented Programming Languages, Applications, Languages, and Systems (OOPSLA)*, volume 26, pages 271–285, New York, November 1991. ACM Press.
10. John Hogg, Doug Lea, Alan Wills, Dennis de Champeaux, and Richard Holt. The Geneva convention of the treatment of object aliasing. *OOPS Messenger*, 3(2):11–16, April 1992.
11. Andrew Kennedy and Don Syme. The design and implementation of Generics for the .NET Common Language Runtime. In *Programming Language Design and Implementation*, 2001.
12. P. Müller and A. Poetzsch-Heffter. *Programming Languages and Fundamentals of Programming*, chapter Universes: a type system for controlling representation exposure. Fernuniversität Hagen, 1999.
13. James Noble, Robert Biddle, Ewan Tempero, Alex Potanin, and Dave Clarke. Towards a model of encapsulation, 2003. Presented at the ECOOP 2003 IWACO Workshop on Aliasing, Confinement, and Ownership. Available at: http://www.mcs.vuw.ac.nz/comp/Publications.
14. James Noble, Jan Vitek, and John Potter. Flexible alias protection. In *European Conference on Object-Oriented Programming (ECOOP)*, Lecture Notes in Computer Science. Springer-Verlag, 1998.
15. Alex Potanin. The fox - a tool for java object graph analysis. Technical Report 02/28, School of Matematical and Computing Sciences, Victoria University of Wellington, http://www.mcs.vuw.ac.nz/comp/Publications/, November 2002.

16. Alex Potanin, James Noble, and Robert Biddle. Checking ownership and confinement. *Concurrency and Computation: Practice and Experience*, 2004. Accepted for Publication.
17. Alex Potanin, James Noble, and Robert Biddle. Snapshot query-based debugging. In *Australian Software Engineering Conference (ASWEC)*, 2004.
18. Alex Potanin, James Noble, Dave Clarke, and Robert Biddle. Featherweight generic confinement. In *Foundations of Object-oriented Programming (FOOL11)*, Venice, Italy, January 2004.
19. Alex Potanin, James Noble, Dave Clarke, and Robert Biddle. Generic ownership. In *European Conference on Object-Oriented Programming (ECOOP)*, 2004. Submitted for publication.
20. Alex Potanin, James Noble, Marcus Frean, and Robert Biddle. Scale-free geometry in object-oriented programs. *Communications of the ACM*, 2004. Accepted for Publication.