Generic Ownership: Practical Ownership Control in Programming Languages

Alex Potanin
PhD Advisors: James Noble and Robert Biddle
Victoria University of Wellington, New Zealand
{alex,kjx,robert}@mcs.vuw.ac.nz

ABSTRACT

This research abstract outlines the work I plan to do as part of my PhD. 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.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features

General Terms

Reliability,Security,Languages

1. INTRODUCTION

An object is *aliased* whenever there is more than one pointer referring to that object [12]. 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 [17, 3]. 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 [16]. 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. We conducted a number of studies using a snapshot query-based debugger called Fox [20] that demonstrated 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. More details can be found in my Honours report and related publications ([18, 24, 19, 20]).

2. OWNERSHIP GENERIC JAVA (OGJ)

Review of the literature for the work on object instance encapsulation exhibits two main approaches to dealing with aliasing: significantly modifying a language to allow ownership support [11, 3, 9], or enforcing coding conventions within an existing programming language [8, 15, 2, 4, 6]. Unfortunately, the former approaches do not take the introduction of generics in modern programming languages into account [5, 14], 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#. This approach worked exceptionally well and I developed a compiler extension, called Ownership Generic Java (OGJ), that supports generics and ownership. It is hoped that it will provide a novel and practical way of making it possible for ownership support in the programming languages to be adopted by programmers [23, 22, 21].

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 [21].

3. FUTURE THESIS WORK

Having set a good foundation for my PhD work in the first year, I am spending the second year finalising the formal side of OGJ. At this stage, with a lot of advice from Dr Dave Clarke (my unofficial PhD adviser), I have developed a formal model of OGJ based on Featherweight Generic Java with locations ([13, 7, 1]) and have a proof of types soundness, ownership and confinement guarantees.

My plan for the second half of my PhD is to expand on the proposed model of defaulting [22] any Generic Java program to have ownership support. Once the formal framework of specifying ownership information for any Generic Java program is developed and the advantages (as well as formal proof principles) of ownership that OGJ provides are stated, it would be possible to advance the major, and rather bold, claim of my thesis, that:

> Generic Ownership is the easiest way of having ownership in a language.

After the development of the formal and practical side of OGJ is completed, the plan is to go back to using a snapshot query-based debugger to conduct a second extensive study of a corpus of heap snapshots, now with programs developed using OGJ to see if the changes did make it possible for aliasing to be controlled. It would require development of a number of large programs using OGJ, which may be achieved 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 this approach), I don't expect this to cause much disruptions to the coding tasks facing the developers.

If the study comes up with positive experimental data, I can see the possibility of offering ownership as a Java Specification Request (JSR) for a future introduction of ownership in programming languages. This, of course, is going to be a collaborative effort among many ownership types researchers. Additional extensions to both formal and practical model developed around OGJ include support for deep ownership, external uniqueness, manifest ownership, ownership transfer, and possibly more [7, 10, 4, 8, 1].

4. REFERENCES

[1] Aldrich, J., and Chambers, C. Ownership domains: Separating aliasing policy from mechanism. In *ECOOP* (Oslo, Norway, 2004), Springer-Verlag.

[2] Aldrich, J., Kostadinov, V., and Chambers, C. Alias annotations for program understanding. In *OOPSLA* (Nov. 2002).

[3] Bokowski, B., and Vitek, J. Confined types. In *Proceedings of Conference on Object-Oriented Programming, Languages, and Applications* (1999), ACM Press.

[4] Boyapati, C., Liskov, B., and Shrira, L. Ownership types for object encapsulation. In *ACM Symposium on Principles of Programming Languages (POPL)* (Jan. 2003).

[5] Bracha, G., Odersky, M., Stoutamire, D., and Wadler, P. Making the future safe for the past: Adding Genericity to the Java programming language. In *OOPSLA* (Oct. 1998).

[6] Cargill, T. Localized ownership: managing dynamic objects in c++. *Pattern languages of program design 2* (1996), 5–18.

[7] Clarke, D. *Object ownership and containment*. PhD thesis, School of Computer Science and Engineering, University of New South Wales, Australia, 2002.

[8] Clarke, D., and Drossopoulou, S. Ownership, encapsulation, and the disjointness of type and effect. In *OOPSLA* (2002).

[9] Clarke, D., Richmond, M., and Noble, J. Saving the world from bad beans: Deployment-time confinement checking. In *OOPSLA* (October 2003).

[10] Clarke, D., and Wrigstad, T. External uniqueness is unique enough. In *ECOOP* (Darmstadt, Germany, July 2003), L. Cardelli, Ed., vol. 2473 of *Lecture Notes In Computer Science*, Springer-Verlag, pp. 176–200.

[11] Hogg, J. Islands: Aliasing protection in object-oriented languages. In *OOPSLA* (New York, Nov. 1991), vol. 26, ACM Press, pp. 271–285.

[12] Hogg, J., Lea, D., Wills, A., de Champeaux, D., and Holt, R. The Geneva convention of the treatment of object aliasing. *OOPS Messenger 3*, 2 (April 1992), 11–16.

[13] Igarashi, A., Pierce, B. C., and Wadler, P. Featherweight Java: a minimal core calculus for Java and GJ. *ACM Transactions on Programming Languages and Systems (TOPLAS) 23*, 3 (May 2001), 396 – 450.

[14] Kennedy, A., and Syme, D. The design and implementation of Generics for the .NET Common Language Runtime. In *Programming Language Design and Implementation* (2001).

[15] Müller, P., and Poetzsch-Heffter, A. *Programming Languages and Fundamentals of Programming*. Fernuniversität Hagen, 1999, ch. Universes: a type system for controlling representation exposure.

[16] Noble, J., Biddle, R., Tempero, E., Potanin, A., and Clarke, D. Towards a model of encapsulation. In *Proceedings of the IWACO Workshop in European Conference on Object-Oriented Programming* (July 2003).

[17] Noble, J., Vitek, J., and Potter, J. Flexible alias protection. In *ECOOP* (1998), Lecture Notes in Computer Science, Springer-Verlag.

[18] Potanin, A. The fox - a tool for java object graph analysis. Tech. Rep. 02/28, School of Matematical and Computing Sciences, Victoria University of Wellington, http://www.mcs.vuw.ac.nz/comp/Publications/, November 2002.

[19] Potanin, A., Noble, J., and Biddle, R. Checking ownership and confinement. *Concurrency and Computation: Practice and Experience 16*, 7 (2004), 671–687.

[20] Potanin, A., Noble, J., and Biddle, R. Snapshot query-based debugging. In *Australian Software Engineering Conference (ASWEC)* (2004).

[21] Potanin, A., Noble, J., Clarke, D., and Biddle, R. Generic ownership. Submitted for publication.

[22] Potanin, A., Noble, J., Clarke, D., and Biddle, R. Defaulting Generic Java to Ownership. In *Proceedings of the Workshop on Formal Techniques for Java-like Programs in European Conference on Object-Oriented Programming* (Oslo, Norway, June 2004), Springer-Verlag.

[23] Potanin, A., Noble, J., Clarke, D., and Biddle, R. Featherweight generic confinement. In *Foundations of Object-Oriented Programming (FOOL11)* (Venice, Italy, January 2004).

[24] Potanin, A., Noble, J., Frean, M., and Biddle, R. Scale-free geometry in object-oriented programs. *Communications of the ACM* (2004). Accepted for Publication.