Is the World Ready for Ownership Types? Is Ownership Types Ready for the World?

Tobias Wrigstad[1] and Dave Clarke[2]

[1)]Department of Information Technology Uppsala University

[2)]IBBT-DistriNet Dept. Computer Sciences K.U.Leuven

Abstract This position paper identifies several weaknesses in the state of reseach on ownership-based confinement and calls for a coordinated effort to tackle this problem with the ultimate goal of making ownership types ready for the world.

Seven years ago, Wrigstad started his two master students Cele and Stureborg [5] in an early experiment with ownership types in practical programming. The experiment uncovered substantial problems with refactoring, a process of piecemeal improvement of the structure of a piece of software by performing semantically preserving minor transformations, such as moving data from one class into another class whose objects most frequently access it. Similar problems are found when testing systems with strong encapsulation; writing code that is easily testable often goes against the encapsulation that one wants in the product builds of the software, or for applying program analysis. Maybe the answer lies in pluggability or gradual encapsulation [17]?

In 2011, there is still a significant lack of research on how systems for alias encapsulation interact with existing programming practices and most proposed systems use an "all-or-nothing approach" that requires an entire system to be heavily annotated if any encapsulation guarantees are to be given. We still know (surprisingly) little about how compatible actual encapsulation invariants and recurring patterns found in existing software are with the various flavours of ownership types[1]; how common programming idioms can be expressed in terms of their effect on object interconnection structures; how fundamental programming practises, such as refactoring, interact with systems that place rigid restrictions on object graphs or if there exists a corresponding set of refactoring principles for evolving alias encapsulation. Furthermore, we do not know what flexibility is required by idioms and practises that rely on privileged access to data, such as equivalence tests or software testing, and whether there is an underlying principle that can be extracted. And if so; how do these principles relate to strong encapsulation?

[1] Although see Nägeli's masters thesis [12] for a notable exception that investigates the compatibility of ownership types with design patterns and successfully applies ownership to two small examples.

Within the subfield of ownership-based object encapsulation, a few studies exist that investigate the state of real-world programs. In a rare and much welcomed exception, Potanin et al. [14] have studied the run-time structure of programs. The study was based on snapshots of a running system, and each snapshot showed that an overwhelming percentage of all objects were not aliased but uniquely referenced and that roughly one third of all objects are strongly encapsulated. Sadly, we must take these results with a grain of salt, since the method does not capture temporary violations of uniqueness occurring in-between the snapshots, etc. Abi-Antoun and Aldrich [1] perform static extraction of object graphs based on ownership domains [2], which is an interesting start, but their study is only on 68 KLOC in a single program, which is quite small to generalise from. While Abi-Antoun and Aldrich's efforts are commendable, they do not scale to the majority of ownership research which are based on much stricter confinement invariants such as owners-as-dominators [6,15] or owners-as-modifiers [10].

We argue that research on ownership types must incorporate larger practical experiments that involve ownership constructs in real-world programming in order to be taken seriously. Although excursions in formal territories are important and often enlightening, we believe there is a "disjointness between theory and practice" in the field of ownership types that may hide shortcomings of many proposals in their interaction with common programming idioms or practises. As an example, informal experiments by the authors with the Joe$_1$ [7] programming language shows that the circumstances necessary for disjointness seldom arise in practise in programming, rendering the proposal much less appealing in practise than it appears on paper. This does not by any means demean the results of Clarke and Drossopoulou, but also does not help in getting practitioners interested in the results of the field, and the practical weaknesses might propagate in derivative proposals [13,4] in silence. As an additional anecdotal evidence, it has been long said that ownership types systems do not need run-time ownership information since conformance to *e.g.,* owners-as-dominators is guaranteed at compile-time. However, in systems that do not support run-time ownership information (which is expensive [3]), downcasting in the presence of ownership types is complicated but necessary to encode frequently occurring idioms such as the common Java equals method for structural equivalence testing (one can also resort to downcasting to volatile existential owners [18]). Surely actual programming in our proposed systems, even with ocular type checking, would have made such observations early on, and reported on them faithfully? Proposals such as Joelle [9] and Loci [19] have some success in bringing ownership types close to real-world programming (the latter with at least 100 KLOC actual translated code), but only for flat ownership structures that give far less interesting guarantees than a fully-fledged ownership types system such as Jolene [8] or OGJ [15]. Surely, we must be able to do better when bringing ownership to the masses?

We call for a *coordinated effort* to make ownership-based confinement ready for the world. We identify the need for a common, pluggable compiler infrastructure in the public domain that facilitates easy extension and practical experiment with language constructs for object encapsulation. This compiler should not stand alone, but be accompanied by a well-accepted set of standard corpus programs (note the plural; we cannot rely solely on JHotDraw as the litmus test), idioms, refactorings, etc. to be translated, expressed, or carried out as minimal benchmarking and real-world testing of encapsulation mechanisms. Translating programs into using ownership types will require libraries and better integration with libraries is a long-standing problem. Key libraries for recurring data structures should not only be in the corpus, but be specifically studied in terms of their encapsulation properties and interconnection structure.

We believe that ubiquitous parallelism gives a window of opportunity to push ownership-related constructs into the mainstream, see for example [9,19,11,16] for encouraging examples moving in this direction. Thus, integration with concurrency libraries, developments of methods for program transformation for parallelism, determinism, etc. should be a key target for researchers in the field and this should further influence the compiler and corpus selection.

As the understanding of object-encapsulation in real-world program matures, this set of standard programs etc. can be extended to improve on the validity of future results. There is also a lack of a common website to promote and make available results from work in the field in an approachable fashion.

In conclusion, we believe that the iron for ownership is hot, but that much work is needed before it makes sense to strike. IWACO is a suitable venue to begin this discussion and solicit feedback and cooperation from active experts of the field.

Acknowledgements The authors wishes to thank the anonymous reviewer that reminded them of Nägeli's work. Tobias Wrigstad's work was supported in part by the Swedish Research Council within the UPMARC Linnaeus centre of Excellence.

References

1. M. Abi-Antoun and J. Aldrich. Static extraction and conformance analysis of hierarchical runtime architectural structure using annotations. *SIGPLAN Not.*, 44:321–340, October 2009.
2. J. Aldrich and C. Chambers. Ownership Domains: Separating Aliasing Policy from Mechanism. In *ECOOP*, volume 3086 of *LNCS*, pages 1–25. Springer, June 2004.
3. C. Boyapati, R. Lee, and M. Rinard. Safe runtime downcasts with ownership types. In *International Workshop on Aliasing, Confinement and Ownership in Object-oriented Programming*, July 2003. Submitted to IWACO'11.
4. N. R. Cameron, S. Drossopoulou, J. Noble, and M. J. Smith. Multiple Ownership. In *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2007.

5. G. Cele and S. Stureborg. Ownership Types in Practice. Master's thesis, Stockholm University, 2005. TR-02-02.
6. D. Clarke. *Object Ownership and Containment*. PhD thesis, School of Computer Science and Engineering, University of New South Wales, Sydney, Australia, 2001.
7. D. Clarke and S. Drossopoulou. Ownership, encapsulation and the disjointness of type and effect. In *Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '02, pages 292–310, New York, NY, USA, 2002. ACM.
8. D. Clarke and T. Wrigstad. External Uniqueness is Unique Enough. In *ECOOP 2003 – Object-Oriented Programming*, volume 2743/2003 of *Lecture Notes in Computer Science*, pages 59–67. Springer Berlin / Heidelberg, 2003.
9. D. Clarke, T. Wrigstad, J. Östlund, and E. B. Johnsen. Minimal Ownership for Active Objects. In *Programming Languages and Systems (Procedings of the 6th Asian Symposium on Programming Languages and Systems)*, volume 5356/2008 of *Lecture Notes in Computer Science*, pages 139–154. Springer Berlin / Heidelberg, 2008.
10. W. Dietl, S. Drossopoulou, and P. Müller. Generic universe types. In *In ECOOP*, pages 28–53. Springer, 2007.
11. P. Haller and M. Odersky. Capabilities for Uniqueness and Borrowing. In *Proceedings of the 24th European Conference on Object-Oriented Programming*, Lecture Notes in Computer Science. Springer, 2010.
12. S. Nägeli. Ownership in design patterns. Master's thesis, ETH Zurich, 2006.
13. J. Östlund, T. Wrigstad, D. Clarke, and B. Åkerblom. Ownership, Uniqueness, and Immutability. In *Objects, Components, Models and Patterns (Proceedings of 46th International Conference on Objects, Models, Components, Patterns)*, volume 11 of *Lecture Notes in Business Information Processing*, pages 178–197. Springer Berlin Heidelberg, 2008.
14. A. Potanin and J. Noble. Checking ownership and confinement properties. In *Formal Techniques for Java-like Programs*, 2002.
15. A. Potanin, J. Noble, D. Clarke, and R. Biddle. Generic Ownership for Generic Java. In *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, OOPSLA '06, pages 311–324, New York, NY, USA, 2006. ACM.
16. J. Schäfer and A. Poetzsch-Heffter. JCoBox: Generalizing Active Objects to Concurrent Components. In *24th European Conference on Object-Oriented Programming (ECOOP 2010)*, LNCS, pages 275–299. Springer, June 2010.
17. I. Sergey and D. Clarke. Towards gradual ownership types, 2011. To appear at IWACO'11.
18. T. Wrigstad and D. Clarke. Existential owners for ownership types. *Journal of Object Technology*, 6(4), May/June 2007.
19. T. Wrigstad, F. Pizlo, F. Meawad, L. Zhao, and J. Vitek. Loci: Simple thread-locality for java. In *ECOOP 2009 — Object-Oriented Programming*, volume 5653 of *Lecture Notes in Computer Science*, pages 445–469. Springer Berlin / Heidelberg, 2009.