Flexible Ownership Domain Annotations for Expressing and Visualizing Design Intent

Marwan Abi-Antoun and Jonathan Aldrich (Advisor)

School of Computer Science
Carnegie Mellon University
{marwan.abi-antoun,jonathan.aldrich}@cs.cmu.edu

Abstract. Flexible ownership domain annotations can express and enforce design intent related to encapsulation and communication in real-world object-oriented programs.

Ownership domain annotations also provide an intuitive and appealing mechanism to obtain a sound visualization of a system's execution structure at compile time. The visualization provides design intent, is hierarchical, and thus more scalable than existing approaches that produce mostly non-hierarchical raw object graphs.

The research proposal is to make the ownership domains type system more flexible and develop the theory and the tools to produce a sound visualization of the execution structure from an annotated program and infer many of these annotations semi-automatically at compile time.

1 Problem Description

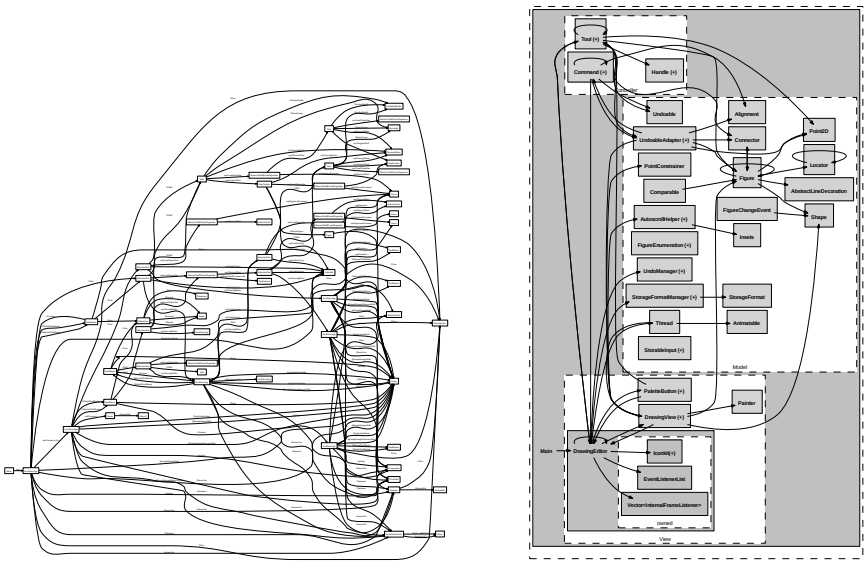
To correctly modify an object-oriented program, a developer often needs to understand both the code structure (static hierarchies of classes) and the execution structure (dynamic networks of communicating objects). Several tools can extract class diagrams of the static structure from code. To address the problem of extracting the execution structure of an object-oriented program, several static and dynamic analyses have been proposed.

Existing dynamic analyses, e.g., [1, 2] suffer from several problems. First, runtime heap information does not convey design intent. Second, a dynamic analysis may not be repeatable, i.e., changing the inputs or exercising different use cases might produce different results. Third, a dynamic analysis cannot be used on an incomplete program, e.g., to analyze a framework separately from its instantiation. Finally, some dynamic analyses carry a significant runtime overhead — a 10x-50x slowdown in one case [2], which must be incurred each time the analysis is run. Existing compile-time approaches visualize the execution structure using heavyweight and thus unscalable analyses [3] or produce non-hierarchical views that do not scale or provide design intent [3, 4].

Example: JHotDraw. JHotDraw [5] has 15,000 lines of Java code rich with design patterns. However, existing compile-time tools that extract class diagrams or raw object graphs of the execution structure from the implementation do not convey its Model-View-Controller (MVC) design [4] (See Figure 1(a)).

2 Thesis Statement

Hypothesis #1: Flexible ownership domain annotations can express and enforce design intent related to encapsulation and communication in real object-oriented programs. Ownership domains [6] divide objects into *domains* — i.e., conceptual groups, with explicit policies that govern references between



(a) Visualization of JHotDraw (without annotations) using Womble [4].

(b) The Ownership Object Graph for JHotDraw [16].

Fig. 1. Side-by-side comparison of compile-time visualizations.

ownership domains. Each object is in a single ownership domain and each object can in turn declare one or more public or private domains to hold its internal objects, thus supporting hierarchy. An ownership *domain* can convey design intent and represent an architectural tier — e.g., the `Model` tier in the MVC pattern [7] — and a *domain link* can abstract permissions of when objects in two domains are allowed to communicate [6] — e.g., objects in the `View` domain can communicate with objects in the `Model` domain, but not vice versa.

Preliminary Work. We added ownership domain annotations to two real 15,000-line Java programs, one developed by experts and one by novices [7]. In the process, we encountered expressiveness challenges in the type system that we plan on addressing. For instance, due to *single ownership*, an object can be in only one ownership domain which makes it difficult to annotate listeners [7] and existential ownership [8, 9] may increase the expressiveness [7].

Expected Contribution #1: We plan to make the ownership domains type system by Aldrich et al. more expressive and more flexible. We will evaluate the modified set of annotations using case studies on non-trivial programs.

Hypothesis #2: Imposing an ownership hierarchy on a program's runtime structure through ownership domain annotations provides an intuitive and appealing mechanism to obtain a sound visualization of a system's execution structure at compile time. Furthermore, the visualization is hierarchical and provides design intent.

By grouping objects into clusters called *domains*, ownership domain annotations provide a coarse-grained *abstraction* of the structure of an application — an important goal for a visualization [10, 11] and for scaling to larger programs.

Since the ownership domains type system guarantees that two objects in two different domains cannot be aliased, the analysis can distinguish between instances of the same class in different domains, which would be merged in a class diagram. This produces more precision than aliasing-unaware analyses [4] and more scalability than more precise but more heavyweight alias analyses [3].

Moreover, ownership domain names are specified by a developer, so they can convey abstract design intent more than arbitrary aliasing information obtained using

a static analysis that does not rely on annotations [12]. Finally, unlike approaches that require annotations just to obtain a visualization [13], ownership annotations are useful in their own right to enforce object encapsulation as illustrated by the existing research into ownership types [14, 8, 6, 15].

The novel contribution is the idea of visualizing the execution structure based on ownership domains. Compared with compile-time visualizations of code structure, ownership domains allow a visualization to include important information about the program's runtime object structures. Compared with dynamic ownership visualizations – which are descriptive and show the ownership structure in a single run of a program, a compile-time visualization is prescriptive and shows ownership relations that will be invariant over all program runs. Thus, this class of visualization is new and valuable.

Preliminary Work. We defined a visualization of the execution structure based on ownership domain annotations, the Ownership Object Graph [16]. We evaluated the visualization on two real 15,000-line Java programs that we previously annotated. In both cases, the automatically generated visualizations fit on one page, illustrated the design intent — e.g., JHotDraw's MVC design [16] — and gave us insights into possible design problems. The visualization is complementary to existing visualizations of the code structure and compares favorably with flat object graphs that do not fit on one readable page [16] (See Figure 1(b)).

Expected Contribution #2: The Ownership Object Graph would be most useful if it were *sound*, i.e., it should not fail to reveal relationships that may actually exist at runtime, up to a minimal set of assumptions regarding reflective code, unannotated external libraries, etc. Otherwise, the technique would not be an improvement over existing unsound heuristic approaches that do not require annotations and assume that the graph can be post-processed manually to become readable [4]. We plan to augment our formal definition of the Ownership Object Graph with a formal proof of soundness by defining the invariants imposed on the generated data structures, their well-formedness rules and relate the visualization objects abstractly to the program's runtime object graph.

Hypothesis #3: Once a developer initially adds a small number of annotations manually, it is possible to infer a large number of the remaining annotations semi-automatically. The visualization requires ownership domain annotations, but adding these annotations manually to a large code base is a significant burden. It is precisely such large systems where meaningful views of the execution structure would be most beneficial. In our experience, simple defaults can only produce between 30% and 40% of the annotations [7]. We plan to extend the earlier work on compile-time annotation inference by Aldrich et al. [17] and improve its precision and usability. We plan to develop an approach whereby a developer indicates the design intent by providing a small number of annotations manually; scalable algorithms and tools then infer the remaining annotations automatically. Finally, the Ownership Object Graph helps the developer visualize and confirm the source code annotations.

Expected Contribution #3: We will develop a semi-automated interactive inference tool to help a developer add annotations to a code base without running the program. We will evaluate the tool by taking the programs that we previously annotated manually, removing the annotations and then using the tool to infer the annotations for that code. We chose this methodology since adding ownership domain annotations is often accompanied by refactoring to reduce coupling, to program to an interface instead of a class or to encapsulate fields [7]. Finally, we will compare the visualization obtained from the manually annotated program to the one obtained from the program with the tool-generated annotations.

3 Conclusion

The thesis of this research proposal revolves around adding ownership domain annotations to a program because: a) the annotations can be flexible yet express and enforce the design intent directly in code; b) the annotations can help produce a sound visualization of the execution structure which complements existing visualizations of the code structure; and c) many of these annotations can be inferred semi-automatically.

The goal is to make the ownership domains type system flexible and expressive enough to handle real-world complex object-oriented code. The research will then produce the theory and the tools to obtain a sound visualization of the execution structure from an annotated program and infer many of these annotations semi-automatically at compile time. The type system, visualization and inference will be evaluated in several case studies on real code.

Acknowledgements

This work was supported in part by NSF grant CCF-0546550, DARPA contract HR00110710019, the Department of Defense, and the Software Industry Center at CMU and its sponsors, especially the Alfred P. Sloan Foundation.

References

1. Rayside, D., Mendel, L., Jackson, D.: A Dynamic Analysis for Revealing Object Ownership and Sharing. In: Workshop on Dynamic Analysis. (2006) 57–64
2. Flanagan, C., Freund, S.N.: Dynamic Architecture Extraction. In: Workshop on Formal Approaches to Testing and Runtime Verification. (2006)
3. O'Callahan, R.W.: Generalized Aliasing as a Basis for Program Analysis Tools. PhD thesis, Carnegie Mellon University (2001)
4. Jackson, D., Waingold, A.: Lightweight Extraction of Object Models from Bytecode. IEEE Transactions on Software Engineering **27** (2001) 156–169
5. Gamma, E. et al.: JHotDraw. `http://www.jhotdraw.org/` (1996)
6. Aldrich, J., Chambers, C.: Ownership Domains: Separating Aliasing Policy from Mechanism. In: ECOOP. (2004) 1–25
7. Abi-Antoun, M., Aldrich, J.: Ownership Domains in the Real World. In: Intl. Workshop on Aliasing, Confinement and Ownership. (2007)
8. Clarke, D.: Object Ownership & Containment. PhD thesis, University of New South Wales (2001)
9. Lu, Y., Potter, J.: Protecting Representation With Effect Encapsulation. In: POPL. (2006) 359–371
10. Sefika, M., Sane, A., Campbell, R.H.: Architecture-Oriented Visualization. In: OOPSLA. (1996) 389–405
11. Lange, D.B., Nakamura, Y.: Interactive Visualization of Design Patterns Can Help in Framework Understanding. In: OOPSLA. (1995) 342–357
12. Rayside, D., Mendel, L., Seater, R., Jackson, D.: An Analysis and Visualization for Revealing Object Sharing. In: Eclipse Technology eXchange (ETX). (2005) 11–15
13. Lam, P., Rinard, M.: A Type System and Analysis for the Automatic Extraction and Enforcement of Design Information. In: ECOOP. (2003) 275–302
14. Noble, J., Vitek, J., Potter, J.: Flexible Alias Protection. In: ECOOP. (1998)
15. Dietl, W., Müller, P.: Universes: Lightweight Ownership for JML. Journal of Object Technology **4** (2005) 5–32
16. Abi-Antoun, M., Aldrich, J.: Compile-Time Views of Execution Structure Based on Ownership. In: Intl. Workshop on Aliasing, Confinement and Ownership. (2007)
17. Aldrich, J., Kostadinov, V., Chambers, C.: Alias Annotations for Program Understanding. In: OOPSLA. (2002) 311 – 330