Enforcing Object-based Access in Newspeak

Ryan Macnak Gilad Bracha

September 1, 2015

Controlling access to functions is common in programming languages, for security and for software engineering purposes. Often access control is based on a static type discipline, but this does not work in dynamically typed languages, and can be problematic in the presence of dynamic loading and reflection.

We argue that dynamically enforcing object-based encapsulation as part of the method lookup semantics is a more attractive approach. There is a natural affinity between object-based encapsulation and capability-based security. The Newspeak programming language capitalizes on this affinity.

We have recently started enforcing access control in the Newspeak system. Here we report on our experience, from the perspectives of both implementation and usability.

Newspeak was designed as an object-capability language. Central to the object-capability approach is the notion of object-based encapsulation. While widely discussed in the literature, actual implementations of object-based encapsulation are rare. In Smalltalk, for example, instance variables are encapsulated on a per-object basis, but all methods are public. In Newspeak, all operations are method invocations and so it becomes even more critical to support encapsulation of methods. However, historically, Newspeak implementations have not enforced access control on methods. This situation has been corrected recently. We are now in a position to assess the implications for the implementation, and examine the impact on code structure and on usability of the live system.

We have implemented dynamic enforcement of access control in both the Newspeak virtual machine (nsvm) and in the Javascript-based implementation of Newspeak. The nsvm includes both an interpreter and a JIT. Naturally, most of the implementation complexity is in the JIT. So far we have found that access control imposes a very small performance penalty.

Access control poses difficulties for the programmer, both during interactive development and debugging and with respect to program architecture and design. A design pattern, whereby data that must be shared across objects is stored in weak tables in a common lexical scope, seems to be the best way of dealing with the architecural issue. Special dispensations in the IDE based on reflection may be needed to retain the pleasant development experience characteristic of the Smalltalk family of languages. We expect to report on our experiences in all these area at the workshop.