Transparent Object Proxies for JavaScript

Matthias Keil[1], Omer Farooq[1], Sankha Narayan Guria[2], Andreas Schlegel[1], Manuel Geffken[1], and Peter Thiemann[1]

Abstract:

This work appeared in the conference proceedings of the *European Conference on Object-Oriented Programming, ECOOP 2015*.

One important question in the design of a proxy API is whether a proxy object should inherit the identity of its target. Apparently proxies should have their own identity for security-related applications whereas other applications, in particular contract systems, require transparent proxies that compare equal to their target objects.

In this work we examine the issue with transparency in various use cases for proxies, discuss different approaches to obtain transparency, and propose two designs that require modest modifications in the JavaScript engine and cannot be bypassed by the programmer.

The JavaScript Proxy API embodies a design decision that reveals the presence of proxies in some important use cases. This decision concerns object equality. Proxies are *opaque*, which means that each proxy has its own identity, different from all other (proxy or non-proxy) objects.

Given opaque proxies, an equality test can be used to distinguish a proxy from its target as demonstrated in the following example:

```
1 var target = { /* some object */ };
2 var handler = { /* empty handler */ };
3 var proxy = new Proxy (target, handler);
4 proxy===target; // evaluates to false
```

Even though *target* and *proxy* behave identically, they are not considered equal. Thus, in a program that uses object equality, the introduction of a proxy along one execution path may change the meaning of the program without even invoking an operation on the proxy (which may behave differently from the same operation on the target).

Equality for opaque proxies works well under the assumption that proxies and their targets are never part of the same execution environment. But the assumption that proxies never share their execution environment with their targets is not always appropriate. One prominent use case is the implementation of a contract system.

Two examples for such systems are the contract framework of Racket [FFP14, Chapter 7] and TreatJS for JavaScript [KT15]. Both systems implement contracts on objects with spe-

[1] University of Freiburg, Freiburg, Germany, {keilr,schlegea,geffken,thiemann}@informatik.uni-freiburg.de
[2] Indian Institute of Technology Jodhpur, Jodhpur, India, sankha@iitj.ac.in

cific wrapper objects, Racket's chaperones or impersonators [St12] and JavaScript proxies, respectively. But this may chang the semantics of a program and thus it violates a ground rule for monitoring: a monitor should never interfere with a program conforming to the monitored property.

Our ECOOP paper [Ke15] shows that a significant number of object comparisons would fail when mixing opaque proxies and their target objects, e.g. when gradually adding contracts to a program. As neither the transparent nor the opaque implementation of proxies is appropriate for all use cases, we propose an alternative designs for *transparent proxies* that is better suited for use cases such as certain contract wrappers and access restricting membranes.

We use object capabilities to create proxies in a particular realm and to create an equality function that only reveals proxies for that realm. A new realm constructor returns a new transparency realm represented by an object that consists of a fresh constructor for transparent proxies (named *Constructor*) and an *equals* function revealing proxies of that realm.

```
var realm = TransparentProxy.createRealm();
var proxy == realm.Constructor(target, handler);
proxy===target; // true
realm.equals(proxy, target); // false
```

The proxy *proxy* is transparent with respect to equality unless someone uses the *realm.equals* method. The *realm.equals* method is a capability that represents the right to reveal proxies of that realm. In addition, the realm also contains a constructor for realm-aware weak maps and weak sets.

References

[Bo15] Boyland, John Tang, ed. 29th European Conference on Object-Oriented Programming, ECOOP 2015, July 5-10, 2015, Prague, Czech Republic, volume 37 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[FFP14] Flatt, Matthew; Findler, Robert Bruce; PLT: . The Racket Guide, v.6.0 edition, March 2014. `http://docs.racket-lang.org/guide/index.html`.

[Ke15] Keil, Matthias; Guria, Sankha Narayan; Schlegel, Andreas; Geffken, Manuel; Thiemann, Peter: Transparent Object Proxies in JavaScript. In: (Boyland) [Bo15], pp. 149–173.

[KT15] Keil, Matthias; Thiemann, Peter: TreatJS: Higher-Order Contracts for JavaScripts. In: (Boyland) [Bo15], pp. 28–51.

[St12] Strickland, T. Stephen; Tobin-Hochstadt, Sam; Findler, Robert Bruce; Flatt, Matthew: Chaperones and impersonators: run-time support for reasonable interposition. In (Leavens, Gary T.; Dwyer, Matthew B., eds): Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012, part of SPLASH 2012, Tucson, AZ, USA, October 21-25, 2012. ACM, pp. 943–962, 2012.