# Suicide Objects *

Jorge Ressia, Fabrizio Perin, Lukas Renggli
Software Composition Group, University of Bern, Switzerland
http://scg.unibe.ch/

## Categories and Subject Descriptors
D.1.5 [**Programming Techniques**]: Object-oriented Programming; D.3.4 [**Programming Languages**]: Memory management (garbage collection)

## General Terms
Languages, Design

## 1. INTRODUCTION
The way programming languages manage memory has been a hot topic for many years. Languages with garbage collection (GC) removed the burden of memory management from developers. Typically the garbage collection infrastructure is a black box that developers have no control over. This is particularly striking in object-oriented systems, where objects themselves have no way to control their life and death. Instead an external process (the GC) decides if an object is still needed or not.

In biology life and death of cells is managed differently. Bob Horvitz, Sydney Brenner and John E. Sulston discovered that genes control the death of a cell [1]. In this process healthy cells have the ability to kill themselves. This is a necessary part of developing tissues, organs, and the central nervous system. Furthermore, the same process is used to remove unnecessary or damaged cells.

In this paper we propose *suicide objects*, objects that make their own decisions about their life and death. With two examples we demonstrate how the traditional garbage collector can be replaced, and how developers can benefit from an object-centric memory management: In Section 2 we look at file streams, and in Section 3 at object caches.

*Jorge Ressia, Fabrizio Perin, and Lukas Renggli. 2012. Suicide objects. In Proceedings of the 6th Workshop on Dynamic Languages and Applications (DYLA '12). ACM, New York, NY, USA, Article 1, 2 pages. DOI=10.1145/2307196.2307197 http://doi.acm.org/10.1145/2307196.2307197

## 2. FILE STREAMS
In most object-oriented systems file streams are represented as objects that refer to a file-handle and some other internal data structures. As with any other object, the GC reclaims the memory of the file stream only if the object is no longer referenced from the application. This does not necessarily happen at the same time as the application closes the file stream. Other objects might still refer to the stream object, leading to memory leaks, or to subtle bugs if clients try to work with a closed file stream.

With suicide objects we can avoid these problems. The file stream decides itself what should happen when it is closed:

```
FileStream>>close
    handle primitiveClose.
    self die: ClosedStream default
```

First the stream does what the traditional implementation did, namely free its external handle. Then it tells the memory management that it is dead and that all existing referenced should be replaced with a shared closed stream reference. This does not only free all buffers referenced from the file stream, but also avoids expensive checks in the implementation of the stream if it is still open. The code ensures that instances of `FileStream` only exist if the object is valid and open.

## 3. OBJECT CACHES
Implementing good caches is difficult. Either they are too agressive and keep objects around for too long[1] or they are too relaxed and let objects die too soon[2]. If the objects can decide themselves how long they should stick around it is much simpler to implement a good cache.

In our paper on ranking software artifacts [2] we noticed that a typical Pharo[3] image contains multiple megabytes of objects that cache the ancestry of Monticello versions, but that these objects are rarely referenced and used. With suicide objects we can implement an arbitrary threshold of when

[1]For example, in Java, strings are cached aggressively. Reading a huge file into a string and referring to a small substring only never frees the original string.
[2]For example, in Smalltalk symbols are held in a weak table causing them to be collected as soon as there are no references. Symbols often disappear and reappear repeatedly causing an unnecessary overhead.
[3]http://www.pharo-project.org/

to prune these objects, or even prune them and replace the referenced parts with an object that automatically reloads them from disk when they are needed.

# 4. SUMMARY

Suicide objects take back the control of when and how objects die. In our demo we will demonstrate our Smalltalk implementation and show how suicide objects integrate with traditional garbage collection. Our prototype does not modify the virtual machine's GC implementation. We provide a mechanism for objects to avoid garbage collection and we explicitly model the existence of a *dead object* whose responsibility is to replace objects that have triggered their own death. We plan to analyze which applications are better suited for suicide objects and which applications are better served by traditional GC. Moreover, we plan to analyze if suicide objects would have a negative or positive impact on the performance.

**Acknowledgments.** We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Synchronizing Models and Code" (SNF Project No. 200020-131827, Oct. 2010 - Sept. 2012).

# 5. REFERENCES

[1] Horvitz, H.R.: Worms, life, and death (nobel lecture). Chembiochem A European Journal Of Chemical Biology **4**(8) (2003) 697–711

[2] Perin, F., Renggli, L., Ressia, J.: Ranking software artifacts. In: 4th Workshop on FAMIX and Moose in Reengineering (FAMOOSr 2010). (2010)