Improved JNI Memory Management Using Allocations from the Java Heap
Jeffrey Sorensen and Daniel Bikel
IBM T. J. Watson Research Center
Yorktown Heights, New York
{sorensj,dbikel}@us.ibm.com

1. Introduction

Because Java only accounts for the storage required for the handles to the C++ objects, the memory management system sees little reason to finalize objects that are unreachable. For Java to effectively manage persistent C++ objects, Java must know (at a minimum) when their aggregate resource consumption necessitates a garbage collection.

2. Java Heap Allocations

Our solution is to use the language features of C++ to override the default heap allocation methods and instead depend upon the Java virtual machine to allocate storage for C++ objects. The principle advantage of this approach is that Java will now become aware of each allocation and will trigger more aggressive garbage collection when these allocations are becoming tight. While Java cannot directly force C++ objects to free up their storage, any unreachable Java objects that maintain handles to C++ objects will be finalized, and in turn, will free up their associated additional heap space.

3. Edge Conditions

Similarly supporting C libraries (that use malloc/free) would pose additional challenges, and would destroy performance. This is because Java itself uses these functions, and C, unlike C++, was not designed for overriding of heap allocators. However, using library interceptors and other tools this idea could be explored.

4. SWIG

SWIG (Simplified Wrapper and Interface Generator) is a popular tool that allows one to publish C and C++ libraries through JNI. Due to its ability to inserting custom code, our implementation of new and delete is entirely compatible with the SWIG environment and can be easily added to a SWIG interface file, automating the use of JNI allocations for all SWIG managed objects. We propose that the inclusion of this code be a standard (or at least optional) feature of the SWIG tool, as the current implementations of SWIG generates code where heap management is likely to be problematical.

5. Conclusion

The difference between the "call-your-own" style of memory management in C++ and the Java Virtual Machine's garbage collection subsystems has generated endless debate about their relative merits. With JNI programming, developers truly have the worst of both worlds. Our own experience suggests that it is far too easy to create C++ objects that persist and consume large amounts of space. Furthermore, with the incorporation of approximately 70 lines of code, a form of détenté can be achieved. While C++ may still consume and hold resources for arbitrary amounts of time, by restricting allocation to the Java heap we are reasserting Java's own memory controls. In addition, the consumption of resources will trigger more frequent garbage collection and, when Java objects are proxies for C++ objects, those objects will be freed through finalization.

6. Complete Listing

```java
package mypackage;
public class MyClass {
    public static void main(String[] args) {
        new MyClass();
    }
}
```