LibSBML Java Bindings and JVM Crashes

This document describes the situation where looping / repeated invocation of the libSBML Java bindings causes a crash of the JVM. Here we summarize the cause and provide workarounds that allow to run applications without these issues.

Symptoms

Consider the following Java snippet, which reads an SBML model and then iterates over all reactions including reactants to do something with the stoichiometry and species:

```
01  SBMLDocument document = libsbml.readSBMLFromFile(fileName);
02  Model model = document.getModel();
03
04  long numReactions = model.getNumReactions();
05  long iter = 0;
06  while (iter < iterations) {
07      for (long i = 0; i < numReactions; ++i) {
08          Reaction reaction = model.getReaction(i);
09          for (long a = 0; a < reaction.getNumReactants(); ++a) {
10              SpeciesReference reactant = reaction.getReactant(a);
11              String species = reactant.getSpecies();
12              double stoich = reactant.getStoichiometry();
13
14              // do something with species and stoichiometry below
15
16          }
17      }
18      ++iter;
19  }
```

Listing 1: initial code example

When this code is run on OSX / Linux occasionally a crash is seen. The crash report will not always display the same position. We have analyzed these crashes by running the Listing 1 on several versions of the OpenJDK (5.0u22, 6u10, 6u18-6u26, 7) under valgrind. No issues with libSBML have been found. The crashes were less likely to occur using Java 5 and Java 6 until update 18 as well as Java 7.

Cause

The most likely cause for these crashes is the garbage collection of libSBML's objects. Since the Java bindings are generated by SWIG, each Java object is pinned to a native object. These native objects have to be destroyed (i.e. deleted) in the same thread as they were created in. However, Java runs a non-deterministic garbage collector, and so determines itself when objects need to be deleted.

In cases like the one in Listing 1, there also is the chance for the following to happen: The reactant (line 10) accessed in one iteration of the outer loop (line 06) is also accessed in subsequent iterations. In other words while each invocation of model.getReaction(0).getReactant(0) returns a new Java object,

the underlying native object is the same in all cases. If it were deleted by the GC, then all subsequent calls will fail on it and possibly crash the VM.

Workaround

So what can be done? The recommended way would be to copy the data out of the libSBML structures into the applications native structure, that way the overhead of crossing the managed <-> native boundaries would not be incurred too often. Baring that there are still a couple of things that can be done:

Use serial garbage collector: -XX:+UseSerialGC

The Hotspot JVM contains a number of garbage collector implementations. The one most likely to work fine is to use the serial garbage collector, as it would minimize the risk of the objects being freed on a different thread than the one that created them.

Specify a large heap size: -Xms1024M

For a larger heap it is less likely that the elements are deleted, while they are still in use. Given that memory is not as low a quantity it used to be spending a gigabyte or two for the heap seems a good idea. It will also cause the garbage collector to run less often, which might speed up the program.

Collect memory periodically

It can be advantageous to run the garbage collection manually every once in a while. This can be done using the commands:

```
System.gc() which is equivalent to running Runtime.getRuntime().gc()
System.runFinalization()
```

Running the GC manually comes of course with the price of a runtime cost.

Delete proxy objects when no longer needed

All SWIG generated proxy objects provide a .delete() method that allows to delete the underlying native object. It is recommended to remove objects whenever they are no longer needed. The ‘delete()’ method will is invoked during finalization by the garbage collector.

Further Information

Further information can be found here:

Troubleshooting Guide for Java SE 6 with HotSpot VM:
http://www.oracle.com/technetwork/java/javase/index-137495.html

Java HotSpot VM Options:
http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html