LIITE 3. Mittausohjelmiston lähdekoodi

Timo Tuominen

Tiedostot

NativeBenchmark

Java-komponentit

Hakemistossa src/fi/helsinki/cs/tituomin/nativebenchmark/.

ApplicationState.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3  import android.content.res.Resources;
4
5  public interface ApplicationState {
6      public void updateState(State state);
7
8      public void updateState(State state, String message);
9
10     public boolean userWantsToRetry(Exception exception);
11
12     public DetailedState getState();
13
14     public Resources getResources();
15
16     public static enum State {
17         INITIALISED(R.string.app_name),
18         MEASURING_STARTED(R.string.measuring_started),
19         INTERRUPTING(R.string.interrupting),
20         INTERRUPTED(R.string.interrupted),
21         MILESTONE(R.string.measuring_milestone),
22         ERROR(R.string.error),
23         INIT_FAIL(R.string.error),
24         MEASURING_FINISHED(R.string.measuring_finished);
25
26         public final int stringId;
27
28         State(int stringId) {
29             this.stringId = stringId;
30         }
31     }
32
33     public class DetailedState {
34         public State state;
35         public String message;
36         private ApplicationState parent;
37
38         public DetailedState(ApplicationState parent) {
39             this.parent = parent;
40             this.state = null;
41             this.message = null;
```

```
        }

        public DetailedState(ApplicationState parent, DetailedState d) {
            this.parent = parent;
            this.state = d.state;
            this.message = d.message;
        }

        public String toString() {
            Resources resources = parent.getResources();
            if (this.state == null) {
                return "<unknown state>";
            }
            String type = resources.getString(this.state.stringId);
            return String.format(
                    "%s%s", type,
                    (this.message != null) ? " " + this.message : "");
        }
    }
}
```

ApplicationStateListener.java

```
package fi.helsinki.cs.tituomin.nativebenchmark;

public interface ApplicationStateListener {

    public void stateUpdated(ApplicationState.DetailedState state);
}
```

BenchmarkController.java

```
package fi.helsinki.cs.tituomin.nativebenchmark;

import android.app.ActivityManager;
import android.content.Context;
import android.content.res.Resources;
import android.os.PowerManager;
import android.util.Log;

import java.io.File;
import java.io.IOException;

import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool;
```

```
13
14 public class BenchmarkController implements ApplicationState {
15
16     public BenchmarkController(Context aContext, File dataDir) {
17         this.detailedState = new ApplicationState.DetailedState(this);
18         this.dataDir = dataDir;
19         this.listeners = new Listeners();
20
21         PowerManager pm = (PowerManager) aContext.getSystemService(Context
22                 .POWER_SERVICE);
23         wakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
24                 "Benchmarking");
25         ActivityManager am = (ActivityManager) aContext.getSystemService
26                 (Context.ACTIVITY_SERVICE);
27         this.resources = aContext.getResources();
28         int memoryClass = am.getLargeMemoryClass();
29         Log.v("Selector", "Memory size " + Runtime.getRuntime().maxMemory());
30         Log.v("onCreate", "memoryClass:" + Integer.toString(memoryClass));
31
32     }
33
34     public Resources getResources() {
35         return this.resources;
36     }
37
38     public void updateState(ApplicationState.State state) {
39         updateState(state, null);
40     }
41
42     public void updateState(ApplicationState.State state, String message) {
43         synchronized (this) {
44             this.detailedState.state = state;
45             this.detailedState.message = message;
46
47             switch (state) {
48                 case MEASURING_STARTED:
49                     try {
50                         LogAccess.start(dataDir);
51                     } catch (IOException e) {
52                         this.detailedState.state = ApplicationState.State.ERROR;
53                         this.detailedState.message = "Could not initialize " +
54                                 "log file.";
55                         Log.e(TAG, this.detailedState.message);
56                     }
57                     wakeLock.acquire();
58                     break;
59                 case ERROR:
60                 case INTERRUPTED:
61                 case MEASURING_FINISHED:
62                     if (wakeLock.isHeld()) {
63                         wakeLock.release();
64                     }
65                     LogAccess.end();
```

```
66                 case INITIALISED:
67                 case INIT_FAIL:
68                 case MILESTONE:
69                     if (this.listeners.milestoneListener != null) {
70                         this.listeners.milestoneListener.stateUpdated(this
71                                 .detailedState);
72                     }
73             }
74         }
75     }
76
77     public ApplicationState.DetailedState getState() {
78         synchronized (this) {
79             return new ApplicationState.DetailedState(this, this.detailedState);
80         }
81     }
82
83     public boolean userWantsToRetry(Exception e) {
84         return false;
85     }
86
87     private class BenchRunnable implements Runnable {
88         BenchRunnable(BenchmarkRunner runner, ToolConfig configuration) {
89             this.runner = runner;
90             this.configuration = configuration;
91         }
92
93         public void run() {
94             this.runner.runBenchmarks(
95                     BenchmarkController.this,
96                     this.configuration,
97                     dataDir);
98         }
99
100        private BenchmarkRunner runner;
101        private ToolConfig configuration;
102    }
103
104    public void startMeasuring(BenchmarkRunner runner, ToolConfig
105            configuration) {
106        String message = null;
107        if (this.resources.getString(R.string.app_dirty).equals("1")) {
108            message = this.resources.getString(R.string.warning_changed);
109        }
110
111        BenchmarkRunner.BenchmarkSet set = configuration.getBenchmarkSet();
112        runner.setBenchmarkSet(set);
113        measuringThread = new Thread(new BenchRunnable(runner, configuration));
114        if (message != null) {
115            this.updateState(ApplicationState.State.MEASURING_STARTED, message);
116        } else {
117            this.updateState(ApplicationState.State.MEASURING_STARTED);
118        }
```

```
119         measuringThread.start();
120     }
121 
122     public void interruptMeasuring() {
123         MeasuringTool.userInterrupt();
124         measuringThread.interrupt();
125     }
126 
127     public void addListener(ApplicationStateListener listener,
128                             ApplicationState.State state) {
129         if (state == ApplicationState.State.MILESTONE) {
130             this.listeners.milestoneListener = listener;
131         }
132     }
133 
134     public void removeListeners() {
135         this.listeners = null;
136     }
137 
138     private class Listeners {
139         public ApplicationStateListener milestoneListener;
140 
141         public Listeners() {
142             milestoneListener = null;
143         }
144     }
145 
146     private Listeners listeners;
147     private ApplicationState.State state;
148     private String message;
149     private ApplicationState.DetailedState detailedState;
150     private PowerManager.WakeLock wakeLock;
151     private File dataDir;
152     private Thread measuringThread;
153     private static final String TAG = "BenchmarkController";
154     private Resources resources;
155 
156 }
```

Benchmark.java

```
1 package fi.helsinki.cs.tituomin.nativebenchmark;
2 
3 import android.os.Process;
4 
5 public abstract class Benchmark implements Runnable {
6     protected abstract void runInternal();
7 
8     public abstract String from();
```

```
9
10      public abstract String to();
11
12      public abstract String description();
13
14      public abstract String id();
15
16      public abstract int sequenceNo();
17
18      public abstract boolean isAllocating();
19
20      public abstract boolean isNonvirtual();
21
22      public abstract boolean dynamicParameters();
23
24      public abstract boolean representative();
25
26      public long repetitionsLeft;
27
28      public void run() {
29          Process.setThreadPriority(-5);
30          runInternal();
31      }
32
33      protected BenchmarkParameter benchmarkParameter;
34      protected long repetitions;
35
36      public void init(BenchmarkParameter bp) {
37          repetitionsLeft = 0;
38          benchmarkParameter = bp;
39          repetitions = -1;
40      }
41
42      public void setRepetitions(long reps) {
43          if (reps < 1) {
44              return;
45          }
46          repetitions = reps;
47          BenchmarkRegistry.setRepetitions(reps);
48      }
49  }
```

BenchmarkParameter.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3  import android.content.pm.PermissionInfo;
4  import android.util.Log;
5
```

```
6  import java.nio.ByteBuffer;
7  import java.util.Arrays;
8  import java.util.Iterator;
9  import java.util.NoSuchElementException;
10
11 import fi.helsinki.cs.tituomin.nativebenchmark.benchmark.JavaCounterparts;
12
13 // Important! Imported as an example class.
14
15 public class BenchmarkParameter implements Iterable<Integer> {
16
17     private native int initReturnvalues(int size, MockObject o);
18
19     private native void freeReturnvalues();
20
21     public void setUp() {
22         initReturnvalues(index * DEFAULTSIZE, mockObjectInstance);
23         JavaCounterparts.initParams(this);
24     }
25
26     public void tearDown() {
27         freeReturnvalues();
28     }
29
30     public static final int DEFAULTSIZE = 64;
31     public static final int RANGE = 8;
32     public static final int MAXSIZE = DEFAULTSIZE * RANGE;
33
34     // The byte buffer has to hold longs and doubles too.
35     public static final int NIO_MAXSIZE = DEFAULTSIZE * RANGE * 8;
36
37     public static MockObject mockObjectInstance = new MockObject();
38
39     public MockObject retrieveMockObject() {
40         return mockObjectInstance;
41     }
42
43     public BenchmarkParameter() {
44         index = 1;
45         if (!generated) {
46             generateAll();
47             generated = true;
48         }
49
50         for (int i = 0; i < RANGE + 1; i++) {
51             if (STRINGS[i] == null) {
52                 Log.v("Parameter", "STRINGS is null at " + i);
53             }
54             if (OBJECTS[i] == null) {
55                 Log.v("Parameter", "OBJECTS is null at " + i);
56             }
57             if (THROWABLES[i] == null) {
58                 Log.v("Parameter", "THROWABLES is null at " + i);
```

```
59              }
60              if (BOOLEAN_ARRAYS[i] == null) {
61                  Log.v("Parameter", "BOOLEAN_ARRAYS is null at " + i);
62              }
63              if (BYTE_ARRAYS[i] == null) {
64                  Log.v("Parameter", "BYTE_ARRAYS is null at " + i);
65              }
66              if (CHAR_ARRAYS[i] == null) {
67                  Log.v("Parameter", "CHAR_ARRAYS is null at " + i);
68              }
69              if (DOUBLE_ARRAYS[i] == null) {
70                  Log.v("Parameter", "DOUBLE_ARRAYS is null at " + i);
71              }
72              if (FLOAT_ARRAYS[i] == null) {
73                  Log.v("Parameter", "FLOAT_ARRAYS is null at " + i);
74              }
75              if (INT_ARRAYS[i] == null) {
76                  Log.v("Parameter", "INT_ARRAYS is null at " + i);
77              }
78              if (LONG_ARRAYS[i] == null) {
79                  Log.v("Parameter", "LONG_ARRAYS is null at " + i);
80              }
81              if (SHORT_ARRAYS[i] == null) {
82                  Log.v("Parameter", "SHORT_ARRAYS is null at " + i);
83              }
84              if (OBJECT_ARRAYS[i] == null) {
85                  Log.v("Parameter", "OBJECT_ARRAYS is null at " + i);
86              }
87          }
88      }
89
90
91      // Must call initreturnvalues and freereturnvalues afterwards.
92      public void setIndex(int index) {
93          if (index < (RANGE + 1)) {
94              this.index = index;
95          } else {
96              throw new IllegalArgumentException("Requested size too large. " +
97                      index);
98          }
99      }
100
101     public int getIndex() {
102         return this.index;
103     }
104
105     public int getSize() {
106         return DEFAULTSIZE * index;
107     }
108
109
110     public Iterator<Integer> iterator() {
111         return new RangeIterator();
```

```
112     }
113
114     private class RangeIterator implements Iterator<Integer> {
115         private int index;
116
117         public RangeIterator() {
118             index = -1;
119         }
120
121         public boolean hasNext() {
122             return index < RANGE;
123         }
124
125         public Integer next() {
126             if (!hasNext()) {
127                 throw new NoSuchElementException();
128             }
129             index++;
130             setIndex(index);
131             return DEFAULTSIZE * index;
132
133         }
134
135         public void remove() {
136             throw new UnsupportedOperationException();
137         }
138     }
139
140     public boolean[] retrieveBooleanArray() {
141         return BOOLEAN_ARRAYS[index];
142     }
143
144     public byte[] retrieveByteArray() {
145         return BYTE_ARRAYS[index];
146     }
147
148     public char[] retrieveCharArray() {
149         return CHAR_ARRAYS[index];
150     }
151
152     public double[] retrieveDoubleArray() {
153         return DOUBLE_ARRAYS[index];
154     }
155
156     public float[] retrieveFloatArray() {
157         return FLOAT_ARRAYS[index];
158     }
159
160     public int[] retrieveIntArray() {
161         return INT_ARRAYS[index];
162     }
163
164     public long[] retrieveLongArray() {
```

```
        return LONG_ARRAYS[index];
    }

    public short[] retrieveShortArray() {
        return SHORT_ARRAYS[index];
    }

    public Object[] retrieveObjectArray() {
        return OBJECT_ARRAYS[index];
    }

    public Object retrieveObject() {
        return OBJECTS[index];
    }

    public Class retrieveClass() {
        return OBJECTS[index].getClass();
    }

    public String retrieveString() {
        String ret = STRINGS[index];
        return ret;
    }

    public Throwable retrieveThrowable() {
        return THROWABLES[index];
    }

    public ByteBuffer retrieveDirectByteBuffer() {
        return BYTEBUFFER;
    }

    private static void generateAll() {
        int index = RANGE - 1;
        int size = MAXSIZE - DEFAULTSIZE;
        generateMax();
        while (index > -1) {
            BOOLEAN_ARRAYS[index] = Arrays.copyOf(BOOLEAN_ARRAYS[RANGE], size);
            CHAR_ARRAYS[index] = Arrays.copyOf(CHAR_ARRAYS[RANGE], size);
            BYTE_ARRAYS[index] = Arrays.copyOf(BYTE_ARRAYS[RANGE], size);
            INT_ARRAYS[index] = Arrays.copyOf(INT_ARRAYS[RANGE], size);
            LONG_ARRAYS[index] = Arrays.copyOf(LONG_ARRAYS[RANGE], size);
            SHORT_ARRAYS[index] = Arrays.copyOf(SHORT_ARRAYS[RANGE], size);
            DOUBLE_ARRAYS[index] = Arrays.copyOf(DOUBLE_ARRAYS[RANGE], size);
            FLOAT_ARRAYS[index] = Arrays.copyOf(FLOAT_ARRAYS[RANGE], size);

            OBJECT_ARRAYS[index] = Arrays.copyOf(OBJECT_ARRAYS[RANGE], size);

            OBJECTS[index] = OBJECT;
            THROWABLES[index] = THROWABLE;
            STRINGS[index] = STRING_BUILDER.substring(0, size);

            index--;
```

```
218             size -= DEFAULTSIZE;
219         }
220     }
221 
222     private static void generateMax() {
223         char c = 0;
224         boolean b = true;
225         byte by = 0;
226         int v = 0;
227         long l = 0;
228         short s = 0;
229         double d = 0;
230         float f = 0;
231 
232         BOOLEAN_ARRAYS[RANGE] = new boolean[MAXSIZE];
233         CHAR_ARRAYS[RANGE] = new char[MAXSIZE];
234         BYTE_ARRAYS[RANGE] = new byte[MAXSIZE];
235         INT_ARRAYS[RANGE] = new int[MAXSIZE];
236         LONG_ARRAYS[RANGE] = new long[MAXSIZE];
237         SHORT_ARRAYS[RANGE] = new short[MAXSIZE];
238         DOUBLE_ARRAYS[RANGE] = new double[MAXSIZE];
239         FLOAT_ARRAYS[RANGE] = new float[MAXSIZE];
240         OBJECT_ARRAYS[RANGE] = new Object[MAXSIZE];
241 
242         for (int i = 0; i < MAXSIZE; i++) {
243             STRING_BUILDER.append(c);
244 
245             BOOLEAN_ARRAYS[RANGE][i] = b;
246             CHAR_ARRAYS[RANGE][i] = c;
247             BYTE_ARRAYS[RANGE][i] = by;
248             INT_ARRAYS[RANGE][i] = v;
249             LONG_ARRAYS[RANGE][i] = l;
250             SHORT_ARRAYS[RANGE][i] = s;
251             DOUBLE_ARRAYS[RANGE][i] = d;
252             FLOAT_ARRAYS[RANGE][i] = f;
253             OBJECT_ARRAYS[RANGE][i] = OBJECT;
254             OBJECTS[RANGE] = OBJECT;
255             THROWABLES[RANGE] = THROWABLE;
256 
257             b = !b;
258             by = (byte) ((by + 1) % Byte.MAX_VALUE);
259             v = (v + 1) % Integer.MAX_VALUE;
260             l = (l + 1) % Long.MAX_VALUE;
261             s = (short) ((s + 1) % Short.MAX_VALUE);
262             d = (d + 0.1);
263             f = (f + 0.1f);
264             c = (char) ((char) (c + '\u0001') % (char) Character.MAX_VALUE);
265         }
266         STRINGS[RANGE] = STRING_BUILDER.substring(0, MAXSIZE);
267     }
268 
269     private int index;
270     private static boolean generated = false;
```

```
271
272     private static final StringBuilder STRING_BUILDER = new StringBuilder();
273     private static final Object OBJECT = new PermissionInfo();
274     private static final Throwable THROWABLE = new Exception();
275
276     private static final String[] STRINGS = new String[RANGE + 1];
277     private static final Object[] OBJECTS = new PermissionInfo[RANGE + 1];
278     private static final Throwable[] THROWABLES = new Exception[RANGE + 1];
279
280     private static final boolean[][] BOOLEAN_ARRAYS = new boolean[RANGE + 1][];
281     private static final byte[][] BYTE_ARRAYS = new byte[RANGE + 1][];
282     private static final char[][] CHAR_ARRAYS = new char[RANGE + 1][];
283     private static final double[][] DOUBLE_ARRAYS = new double[RANGE + 1][];
284     private static final float[][] FLOAT_ARRAYS = new float[RANGE + 1][];
285     private static final int[][] INT_ARRAYS = new int[RANGE + 1][];
286     private static final long[][] LONG_ARRAYS = new long[RANGE + 1][];
287     private static final short[][] SHORT_ARRAYS = new short[RANGE + 1][];
288     private static final Object[][] OBJECT_ARRAYS = new Object[RANGE + 1][];
289
290     private static final ByteBuffer BYTEBUFFER = ByteBuffer.allocateDirect
291             (NIO_MAXSIZE);
292
293 }
```

BenchmarkRegistry.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3  import java.util.LinkedList;
4  import java.util.List;
5
6  import fi.helsinki.cs.tituomin.nativebenchmark.benchmark.JavaCounterparts;
7
8  public class BenchmarkRegistry {
9
10     private static List<Benchmark> benchmarks;
11
12     public static long repetitions;
13     public static final long CHECK_INTERRUPTED_INTERVAL = 1000;
14
15     public static List<Benchmark> getBenchmarks() {
16         return benchmarks;
17     }
18
19     public static void init(long reps) throws ClassNotFoundException {
20         repetitions = reps;
21         benchmarks = new LinkedList<Benchmark>();
22         Class jCounterparts = Class.forName("fi.helsinki.cs.tituomin" +
23                 ".nativebenchmark.benchmark.JavaCounterparts");
```

```
24          Class threadClass = Class.forName("java.lang.Thread");
25          initNative(reps, CHECK_INTERRUPTED_INTERVAL, jCounterparts,
26                  JavaCounterparts.INSTANCE, threadClass);
27      }
28
29      public static native void initNative(long repetitions, long interval,
30                                           Class javaCounterparts,
31                                           JavaCounterparts counterInstance,
32                                           Class threadClass);
33
34      public static native void setRepetitions(long repetitions);
35
36      public static native void interruptNative();
37
38      public static native void resetInterruptFlag();
39  }
```

BenchmarkResult.java

```
1   package fi.helsinki.cs.tituomin.nativebenchmark;
2
3   import android.util.Log;
4
5   import java.util.HashMap;
6   import java.util.Map;
7   import java.util.Map.Entry;
8
9
10  public class BenchmarkResult {
11
12      public BenchmarkResult() {
13          values = new String[ESTIMATED_CAPACITY];
14          valueCount = 0;
15      }
16
17      public void put(String label, String value) {
18          Integer labelIndex = labelIndexes.get(label);
19          if (labelIndex == null) {
20              lastIndex++;
21              if (lastIndex > labels.length) {
22                  Log.e("BenchmarkResults", "Error, too many kinds of values, " +
23                          "increase capacity!");
24              }
25              labels[lastIndex] = label;
26              labelIndexes.put(label, lastIndex);
27              labelIndex = lastIndex;
28          }
29          values[labelIndex] = value;
30          valueCount++;
```

```
    }

    public String get(String label) {
        Integer index = labelIndexes.get(label);
        if (index != null) {
            return values[index];
        } else {
            return null;
        }
    }

    public String get(int i) {
        return values[i];
    }

    public void putAll(BenchmarkResult other) {
        String[] otherValues = other.getValues();
        for (int i = 0; i < size(); i++) {
            if (otherValues[i] != null) {
                values[i] = otherValues[i];
                valueCount++;
            }
        }
    }

    public void putAll(Map<String, String> map) {
        for (Entry<String, String> entry : map.entrySet()) {
            put(entry.getKey(), entry.getValue());
        }
    }

    public String[] getValues() {
        return values;
    }

    public static String getLabel(int i) {
        return labels[i];
    }

    public static String[] labels() {
        return labels;
    }

    public boolean isEmpty() {
        return (valueCount == 0);
    }

    public static int size() {
        return (lastIndex + 1);
    }

    private static final int ESTIMATED_CAPACITY = 200;
    private String[] values;
```

```
84      private int valueCount;
85
86      private static String[] labels = new String[ESTIMATED_CAPACITY];
87      private static Map<String, Integer> labelIndexes = new HashMap<String,
88              Integer>(ESTIMATED_CAPACITY);
89      private static int lastIndex = -1;
90  }
```

BenchmarkRunner.java

```
1   package fi.helsinki.cs.tituomin.nativebenchmark;
2
3   import android.os.SystemClock;
4   import android.util.Log;
5   import android.util.Pair;
6
7   import java.io.File;
8   import java.io.FileInputStream;
9   import java.io.FileNotFoundException;
10  import java.io.IOException;
11  import java.io.InputStream;
12  import java.io.OutputStream;
13  import java.io.PrintWriter;
14  import java.lang.reflect.Method;
15  import java.lang.reflect.Modifier;
16  import java.util.ArrayList;
17  import java.util.Arrays;
18  import java.util.Collections;
19  import java.util.Date;
20  import java.util.HashMap;
21  import java.util.Iterator;
22  import java.util.List;
23  import java.util.Map;
24  import java.util.zip.ZipEntry;
25  import java.util.zip.ZipOutputStream;
26
27  import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool;
28  import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool
29          .RunnerException;
30
31  import static fi.helsinki.cs.tituomin.nativebenchmark.Utils.colPr;
32
33  public enum BenchmarkRunner {
34      INSTANCE; // singleton enum pattern
35
36      private static final String SEPARATOR = ",";
37      private static final String MISSING_VALUE = "-";
38      private static final long WARMUP_REPS = 50000;
39      private static BenchmarkParameter benchmarkParameter;
```

```
40      private static List<MeasuringTool> measuringTools;
41      private static int benchmarkCounter = 0;
42
43      private static boolean interrupted = false;
44
45      public static BenchmarkParameter getBenchmarkParameter() {
46          if (benchmarkParameter == null) {
47              benchmarkParameter = new BenchmarkParameter();
48          }
49          return benchmarkParameter;
50      }
51
52      private BenchmarkRunner() {
53          this.allocatingRepetitions = -1;
54      }
55
56      public void initTools(ToolConfig conf, long repetitions, long
57              allocRepetitions) throws IOException, InterruptedException {
58
59          conf.setDefaultRepetitions(this.repetitions);
60          conf.setDefaultRunAllBenchmarks(this.runAllBenchmarks);
61          if (this.allocatingRepetitions > 0) {
62              conf.setDefaultAllocRepetitions(this.allocatingRepetitions);
63          }
64
65          measuringTools = new ArrayList<MeasuringTool>();
66          for (MeasuringTool tool : conf) {
67              measuringTools.add(tool);
68          }
69      }
70
71      private long repetitions;
72      private long allocatingRepetitions;
73      private CharSequence appRevision;
74      private CharSequence appChecksum;
75      private File cacheDir;
76      private boolean runAllBenchmarks;
77      private boolean runAtMaxSpeed;
78      private String benchmarkSubstring;
79
80      public enum BenchmarkSet {
81          ALLOC,
82          NON_ALLOC
83      }
84
85      ;
86      private BenchmarkSet benchmarkSet;
87
88      public BenchmarkRunner setRepetitions(long x) {
89          repetitions = x;
90          return this;
91      }
92
```

```
    public BenchmarkRunner setAllocatingRepetitions(long x) {
        allocatingRepetitions = x;
        return this;
    }

    public BenchmarkRunner setAppRevision(CharSequence x) {
        appRevision = x;
        return this;
    }

    public BenchmarkRunner setAppChecksum(CharSequence x) {
        appChecksum = x;
        return this;
    }

    public BenchmarkRunner setCacheDir(File x) {
        cacheDir = x;
        return this;
    }

    public BenchmarkRunner setRunAllBenchmarks(boolean x) {
        runAllBenchmarks = x;
        return this;
    }

    public BenchmarkRunner setRunAtMaxSpeed(boolean x) {
        runAtMaxSpeed = x;
        return this;
    }

    public BenchmarkRunner setBenchmarkSubstring(String x) {
        benchmarkSubstring = x;
        return this;
    }

    public BenchmarkRunner setBenchmarkSet(BenchmarkSet x) {
        benchmarkSet = x;
        return this;
    }

    public void runBenchmarks(ApplicationState mainUI, ToolConfig config,
                              File dataDir) {
        interrupted = false;

        try {
            BenchmarkRegistry.init(this.repetitions);
            MeasuringTool.setDataDir(dataDir);
            Log.v(TAG, config.toString());
            initTools(config, this.repetitions, this.allocatingRepetitions);
        } catch (Exception e) {
            mainUI.updateState(ApplicationState.State.ERROR);
            Log.e("BenchmarkRunner", "Error initialising", e);
            return;
```

```
146         }
147
148         benchmarkParameter = getBenchmarkParameter();
149         BenchmarkInitialiser.init(benchmarkParameter);
150
151         List<Benchmark> allBenchmarks = BenchmarkRegistry.getBenchmarks();
152
153         long seed = System.currentTimeMillis();
154         Log.i(TAG, String.format("Random seed %d", seed));
155         java.util.Random random = new java.util.Random(seed);
156         Collections.shuffle(allBenchmarks, random);
157         try {
158             Init.initEnvironment(true); // run warmup at max speed
159         } catch (IOException e) {
160             handleException(e, mainUI);
161             return;
162         }
163         for (MeasuringTool tool : measuringTools) {
164             if (tool == null) {
165                 return;
166             }
167             if (interrupted) {
168                 return;
169             }
170
171             List<Benchmark> benchmarks = new ArrayList<Benchmark>();
172
173             if (this.benchmarkSubstring == null) {
174                 this.benchmarkSubstring = "";
175             }
176             String substringToApply = "";
177
178             String filter = tool.getFilter();
179             if (filter != null && !filter.equals("")) {
180                 substringToApply = tool.getFilter().toLowerCase();
181             }
182             this.benchmarkSubstring = substringToApply;
183
184             for (Benchmark b : allBenchmarks) {
185                 boolean selected;
186                 if (tool.runAllBenchmarks()) {
187                     selected = true;
188                 } else if (!substringToApply.equals("")) {
189                     selected = (
190                             b.getClass().getSimpleName().toLowerCase().indexOf(
191                                     substringToApply) != -1);
192                 } else {
193                     selected = (
194                             b.representative() &&
195                                     ((!b.isAllocating()) && this.benchmarkSet
196                                             == BenchmarkSet.NON_ALLOC) ||
197                                     (b.isAllocating() && this.benchmarkSet ==
198                                             BenchmarkSet.ALLOC));
```

```
199                 }
200                 if (b.isNonvirtual() &&
201                         b.from() != "C" &&
202                         b.to() != "J") {
203                     if (L.og) {
204                         Log.i(TAG, String.format("skipping nonvirtual %s", b
205                                 .id()));
206                     }
207                     selected = false;
208                 }
209                 if (selected) {
210                     benchmarks.add(b);
211                 }
212             }
213             int numOfBenchmarks = benchmarks.size();
214 
215             if (L.og) {
216                 Log.i(TAG, tool.getClass().getSimpleName());
217             }
218 
219             if (!tool.ignore()) {
220                 // set the slower CPU frequency etc. after the warmup
221                 // round(s), taking less time
222                 if (!this.runAtMaxSpeed) {
223                     try {
224                         Init.initEnvironment(false);
225                     } catch (IOException e) {
226                         handleException(e, mainUI);
227                         return;
228                     }
229                 }
230             }
231 
232             int max_rounds = tool.getRounds();
233             String measurementID = Utils.getUUID();
234             File resultFile = new File(dataDir, "benchmarks-" + measurementID
235                     + ".csv");
236             long startTime = SystemClock.uptimeMillis();
237             Date start = new Date();
238             long endTime = 0;
239 
240             int round = -1;
241             boolean labelsWritten = false;
242 
243             ROUNDLOOP:
244             while (++round < max_rounds) {
245                 benchmarkCounter = 0;
246                 PrintWriter tempWriter = null;
247                 File tempFile = new File(this.cacheDir, "benchmarks-temp.csv");
248                 try {
249                     tempWriter = Utils.makeWriter(tempFile, false);
250                 } catch (FileNotFoundException e) {
251                     handleException(e, mainUI);
```

```
252                     return;
253                 }
254
255
256                 List<BenchmarkResult> collectedData;
257
258                 try {
259                     if (tool.explicitGC()) {
260                         System.gc();
261                         Thread.sleep(500);
262                     }
263                 } catch (InterruptedException e) {
264                     logE("Measuring thread was interrupted");
265                     mainUI.updateState(
266                             ApplicationState.State.INTERRUPTED);
267                     interrupted = true;
268                     break ROUNDLOOP;
269                 }
270                 int count = benchmarks.size();
271                 int j = 0;
272                 for (Benchmark benchmark : benchmarks) {
273                     if (L.og) {
274                         Log.i(TAG, (count - j) + " left");
275                         Log.i(TAG, benchmark.getClass().getSimpleName());
276                     }
277                     ;
278                     j++;
279                     try {
280                         collectedData = runSeries(benchmark, mainUI, tool,
281                                 round, max_rounds, count, j);
282                     } catch (RunnerException e) {
283                         logE("Exception was thrown", e.getCause());
284                         mainUI.updateState(
285                                 ApplicationState.State.ERROR);
286                         interrupted = true;
287                         break ROUNDLOOP;
288                     } catch (InterruptedException e) {
289                         logE("Measuring thread was interrupted");
290                         mainUI.updateState(
291                                 ApplicationState.State.INTERRUPTED);
292
293                         interrupted = true;
294                         break ROUNDLOOP;
295                     }
296
297                     if (collectedData.isEmpty() || tool.ignore()) {
298                         continue;
299                     }
300
301                     endTime = SystemClock.uptimeMillis();
302
303                     // print data
304                     for (BenchmarkResult result : collectedData) {
```

```
                    for (int i = 0; i < BenchmarkResult.size(); i++) {
                        String value = result.get(i);
                        if (value == null) {
                            value = MISSING_VALUE;
                        }
                        tempWriter.print(value);
                        tempWriter.print(SEPARATOR);
                    }
                    tempWriter.println("");
                    tempWriter.flush();
                }
            }
            endTime = SystemClock.uptimeMillis();
            tempWriter.close();

            if (tool.ignore()) {
                continue;
            }

            InputStream in = null;
            OutputStream out = null;
            PrintWriter resultWriter = null;
            try {
                resultWriter = Utils.makeWriter(resultFile, true);
                // note: labels should be written last, after
                // all possible keys have been created
                if (!labelsWritten) {
                    String[] labels = BenchmarkResult.labels();
                    for (int i = 0; i < BenchmarkResult.size(); i++) {
                        resultWriter.print(labels[i] + SEPARATOR);
                    }
                    resultWriter.println("");
                    resultWriter.close();
                    labelsWritten = true;
                }

                in = new FileInputStream(tempFile);
                out = Utils.makeOutputStream(resultFile, true);
                Utils.copyStream(in, out);
            } catch (Exception e) {
                mainUI.updateState(ApplicationState.State.ERROR);
                Log.e("BenchmarkRunner", "Error writing results", e);
                interrupted = true;
                break ROUNDLOOP;
            } finally {
                try {
                    if (in != null) {
                        in.close();
                    }
                    if (out != null) {
                        out.flush();
                        out.close();
                    }
```

```
                    if (resultWriter != null) {
                        resultWriter.close();
                    }
                } catch (IOException e) {
                    mainUI.updateState(ApplicationState.State.ERROR);
                    Log.e("BenchmarkRunner", "Error closing files", e);
                    interrupted = true;
                    break ROUNDLOOP;
                }
            }
        }

        if (!tool.ignore()) {
            // there has been at least one succesful round
            writeMeasurementMetadata(
                    new File(dataDir, "measurements.txt"),
                    measurementID,
                    (this.runAtMaxSpeed ? Init.CPUFREQ_MAX : Init.CPUFREQ),
                    startTime, endTime, start, tool, numOfBenchmarks,
                    round);

            List<String> filenames = tool.getFilenames();
            if (!filenames.isEmpty()) {
                OutputStream os = null;

                try {
                    File zip = new File(dataDir, "perfdata-" +
                            measurementID + ".zip");
                    os = Utils.makeOutputStream(zip, false);
                    Log.v("BenchmarkRunner", "filenames " + filenames
                            .size());
                    File measurementMetadataFile = new File(
                            dataDir, "benchmarks-" + measurementID + "" +
                            ".csv");
                    if (measurementMetadataFile.exists()) {
                        filenames.add(measurementMetadataFile
                                .getAbsolutePath());
                    }
                    writeToZipFile(os, filenames, measurementID);
                    deleteFiles(filenames);
                } catch (FileNotFoundException e) {
                    logE(e);
                } catch (IOException e) {
                    logE("Error writing zip file.", e);
                }
                try {
                    if (os != null) {
                        os.close();
                    }
                } catch (IOException e) {
                    logE("Error closing file.", e);
                }
            }
```

```
411             }
412
413         }
414         mainUI.updateState(
415                 ApplicationState.State.MEASURING_FINISHED);
416     }
417
418     private void writeMeasurementMetadata(
419             File catalogFile, String measurementID, int cpuFreq,
420             long startTime, long endTime, Date start, MeasuringTool tool,
421             int numOfBenchmarks, int rounds) {
422
423         Date end = new Date();
424         PrintWriter c = null;
425         try {
426             c = Utils.makeWriter(catalogFile, true);
427
428             c.println("");
429             for (Pair<String, String> pair : tool.configuration()) {
430                 colPr(c, pair.first, pair.second);
431             }
432             colPr(c, "id", measurementID);
433             colPr(c, "cpu-freq", cpuFreq);
434             colPr(c, "logfile", LogAccess.filename());
435             colPr(c, "rounds", rounds);
436             colPr(c, "start", start);
437             colPr(c, "end", end);
438             colPr(c, "duration", Utils.humanTime(endTime - startTime));
439             colPr(c, "benchmarks", numOfBenchmarks);
440             colPr(c, "code-revision", this.appRevision);
441             colPr(c, "code-checksum", this.appChecksum);
442             colPr(c, "benchmark-set", this.benchmarkSet);
443             colPr(c, "substring-filter", this.benchmarkSubstring);
444             c.println("");
445         } catch (IOException e) {
446             logE(e);
447         } finally {
448             c.close();
449         }
450     }
451
452     private final static String TAG = "BenchmarkRunner";
453
454     private static void logE(String message, Throwable e) {
455         Log.e(TAG, message, e);
456     }
457
458     private static void logE(Throwable e) {
459         Log.e(TAG, "exception", e);
460     }
461
462     private static void logE(String msg) {
463         Log.e(TAG, msg);
```

```
464     }
465
466     private static void deleteFiles(List<String> filenames) {
467         for (String filename : filenames) {
468             boolean success = new File(filename).delete();
469             if (!success) {
470                 logE("Error deleting file " + filename);
471             }
472         }
473     }
474
475     private static void handleException(Exception e, ApplicationState UI) {
476         logE(e);
477         UI.updateState(ApplicationState.State.ERROR);
478         return;
479     }
480
481     private static void
482     writeToZipFile(OutputStream os, List<String> filenames, String mID)
483             throws IOException {
484         ZipOutputStream zos = new ZipOutputStream(os);
485         final int byteCount = 512 * 1024;
486         byte[] bytes = new byte[byteCount];
487         for (String filename : filenames) {
488             try {
489                 InputStream is = Utils.makeInputStream(filename);
490                 ZipEntry entry = new ZipEntry(mID + "/" + new File(filename)
491                         .getName());
492                 int bytesRead = -1;
493                 zos.putNextEntry(entry);
494                 while ((bytesRead = is.read(bytes, 0, byteCount)) != -1) {
495                     zos.write(bytes, 0, bytesRead);
496                 }
497                 zos.closeEntry();
498             } finally {
499                 zos.flush();
500             }
501         }
502         try {
503             zos.close();
504         } catch (IOException e) {
505             logE(e);
506         }
507     }
508
509     private static List<BenchmarkResult> runSeries(
510             Benchmark benchmark, ApplicationState mainUI, MeasuringTool tool,
511             int roundNo, int roundCount, int benchmarkCount, int benchmarkIndex)
512             throws InterruptedException, RunnerException {
513
514         List<BenchmarkResult> compiledMetadata = new
515                 ArrayList<BenchmarkResult>();
516
```

```
517         if (Thread.interrupted()) {
518             throw new InterruptedException();
519         }
520         BenchmarkParameter bPar = getBenchmarkParameter();
521         BenchmarkResult introspected;
522         try {
523             introspected = inspectBenchmark(benchmark);
524         } catch (ClassNotFoundException e) {
525             Log.e("BenchmarkRunner", "Could not find class", e);
526             return compiledMetadata;
527         }
528 
529         String refTypesString = introspected.get("has_reference_types");
530         boolean hasRefTypes = (refTypesString != null) && (refTypesString
531                 .equals("1"));
532 
533         String parameterCountString = introspected.get("parameter_count");
534         int parameterCount = (parameterCountString == null) ? -1 : Integer
535                 .parseInt(parameterCountString);
536 
537         boolean dynamicParameters =
538                 benchmark.dynamicParameters() ||
539                         (hasRefTypes && (-1 < parameterCount &&
540                                 parameterCount < 2));
541 
542         Iterator<Integer> iterator = bPar.iterator();
543         Integer i;
544         int j = -1;
545         if (iterator.hasNext()) {
546             i = iterator.next();
547             j++;
548         } else {
549             i = null;
550         }
551         while (i != null) {
552             if (Thread.interrupted()) {
553                 throw new InterruptedException();
554             }
555             bPar.setUp(); // (I) needs tearDown (see II)
556 
557             try {
558                 tool.startMeasuring(benchmark);
559                 benchmarkCounter++;
560             } catch (IOException e) {
561                 logE("Measuring caused IO exception", e);
562                 if (mainUI.userWantsToRetry(e)) {
563                     continue; // without incrementing i
564                 } else {
565                     throw new InterruptedException("User wants to abort");
566                 }
567             } finally {
568                 bPar.tearDown(); // (II) needs setUp (see I)
569             }
```

```
570             if (tool.explicitGC() && benchmarkCounter % 25 == 0) {
571                 System.gc();
572                 Thread.sleep(350);
573             }
574 
575             String message = String.format(
576                     "%s%s round %d/%d benchmark %d/%d range %d/%d",
577                     tool.getClass().getSimpleName(),
578                     tool.isWarmupRound() ? " (warmup)" : "",
579                     roundNo + 1,
580                     roundCount,
581                     benchmarkIndex,
582                     benchmarkCount,
583                     j,
584                     dynamicParameters ? BenchmarkParameter.RANGE : 1
585             );
586             mainUI.updateState(ApplicationState.State.MILESTONE, message);
587 
588             List<BenchmarkResult> measurements = tool.getMeasurements();
589             for (BenchmarkResult measurement : measurements) {
590                 if (!measurement.isEmpty()) {
591                     if (dynamicParameters) {
592                         measurement.put("dynamic_size", "" + i);
593                     } else {
594                         measurement.put("dynamic_size", MISSING_VALUE);
595                     }
596                     measurement.put("id", benchmark.id());
597                     measurement.put("class", benchmark.getClass()
598                             .getSimpleName());
599                     measurement.putAll(introspected);
600                     compiledMetadata.add(measurement);
601                 }
602             }
603             // if parameter size can be varied, vary it - else break with
604             // first size
605             if (!dynamicParameters) {
606                 break;
607             }
608             if (iterator.hasNext()) {
609                 i = iterator.next();
610                 j++;
611             } else {
612                 break;
613             }
614         }
615         return compiledMetadata;
616     }
617 
618 
619     private static BenchmarkResult inspectBenchmark(Benchmark benchmark)
620             throws ClassNotFoundException {
621         BenchmarkResult bdata = new BenchmarkResult();
622         int seqNo = benchmark.sequenceNo();
```

```
623         String from = benchmark.from();
624         String to = benchmark.to();
625         bdata.put("no", "" + benchmark.sequenceNo());
626         bdata.put("description", benchmark.description());
627         bdata.put("from", from);
628         bdata.put("to", to);
629         bdata.put("representative", benchmark.representative() ? "1" : "0");
630         bdata.put("nonvirtual", benchmark.isNonvirtual() ? "1" : "0");
631 
632         if (seqNo == -1) {
633             bdata.put("custom", "1");
634             return bdata;
635         }
636 
637         Class c = Class.forName(
638                 String.format(
639                         "fi.helsinki.cs.tituomin.nativebenchmark.benchmark" +
640                                 ".J2CBenchmark%05d",
641                         seqNo));
642 
643         Method[] methods = c.getDeclaredMethods();
644         for (int i = 0; i < methods.length; i++) {
645 
646             Method m = methods[i];
647             int modifiers = m.getModifiers();
648 
649             if (Modifier.isNative(modifiers)) {
650                 return inspectMethod(m, bdata);
651             }
652         }
653         return bdata;
654     }
655 
656     private static BenchmarkResult inspectMethod(Method m, BenchmarkResult
657             bdata) {
658 
659         Class[] parameters = m.getParameterTypes();
660         List<Class> parameterList = new ArrayList<Class>(Arrays.asList
661                 (parameters));
662         int modifiers = m.getModifiers();
663 
664         Map<String, Integer> parameterTypes = new HashMap<String, Integer>();
665         for (Class param : parameterList) {
666             Integer previousValue = null;
667             String param_typename = param.getSimpleName();
668             previousValue = parameterTypes.get(param_typename);
669             parameterTypes.put(
670                     param_typename,
671                     (previousValue == null ? 1 : ((int) previousValue) + 1));
672         }
673         for (String typename : parameterTypes.keySet()) {
674             bdata.put("parameter_type_" + typename + "_count", parameterTypes
675                     .get(typename) + "");
```

```
676         }
677 
678         Class returnType = m.getReturnType();
679 
680         boolean hasRefTypes = false;
681         parameterList.add(returnType);
682         for (Class cl : parameterList) {
683             if (Object.class.isAssignableFrom(cl)) {
684                 hasRefTypes = true;
685                 break;
686             }
687         }
688 
689         bdata.put("has_reference_types", hasRefTypes ? "1" : "0");
690         bdata.put("parameter_type_count", parameterTypes.keySet().size() + "");
691         bdata.put("parameter_count", parameters.length + "");
692         bdata.put("return_type", returnType.getCanonicalName());
693         bdata.put("native_static", Modifier.isStatic(modifiers) ? "1" : "0");
694         bdata.put("native_private", Modifier.isPrivate(modifiers) ? "1" : "0");
695         bdata.put("native_protected", Modifier.isProtected(modifiers) ? "1" :
696                 "0");
697         bdata.put("native_public", Modifier.isPublic(modifiers) ? "1" : "0");
698 
699         return bdata;
700     }
701 }
```

BenchmarkSelector.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark;
2 
3  //import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkInitialiser;
4  //import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
5  //import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRunner;
6 
7  import android.app.Activity;
8  import android.app.AlertDialog;
9  import android.app.Dialog;
10 import android.app.DialogFragment;
11 import android.app.Notification;
12 import android.app.NotificationManager;
13 import android.app.PendingIntent;
14 import android.app.TaskStackBuilder;
15 import android.content.Context;
16 import android.content.DialogInterface;
17 import android.content.Intent;
18 import android.content.res.Resources;
19 import android.media.RingtoneManager;
20 import android.net.Uri;
```

```
21 import android.os.Bundle;
22 import android.os.Environment;
23 import android.util.Log;
24 import android.view.View;
25 import android.view.WindowManager;
26 import android.widget.AdapterView;
27 import android.widget.AdapterView.OnItemSelectedListener;
28 import android.widget.ArrayAdapter;
29 import android.widget.Button;
30 import android.widget.Checkable;
31 import android.widget.NumberPicker;
32 import android.widget.Spinner;
33 import android.widget.TextView;
34
35 import java.io.File;
36 import java.io.InputStream;
37 import java.io.OutputStream;
38 import java.util.Map;
39
40 public class BenchmarkSelector extends Activity {
41     /**
42      * Called when the activity is first created.
43      */
44     @Override
45     public void onCreate(Bundle savedInstanceState) {
46         super.onCreate(savedInstanceState);
47         this.requestWindowFeature(getWindow().FEATURE_NO_TITLE);
48         setContentView(R.layout.main);
49
50         this.retry = false;
51
52         this.resultView = (TextView) findViewById(R.id.resultview);
53         this.button = (Button) findViewById(R.id.mybutton);
54         this.repView = (TextView) findViewById(R.id.repetitions);
55         this.numPick = (NumberPicker) findViewById(R.id.picker_num);
56         this.expPick = (NumberPicker) findViewById(R.id.picker_exp);
57
58         NumberPicker.OnValueChangeListener listener = new RepsListener();
59
60         numPick.setMinValue(1);
61         numPick.setMaxValue(9);
62         numPick.setValue(1);
63         expPick.setMinValue(0);
64         expPick.setMaxValue(9);
65         expPick.setValue(6);
66         numPick.setOnValueChangedListener(listener);
67         expPick.setOnValueChangedListener(listener);
68
69         listener.onValueChange(numPick, 0, 0);
70
71         if (getResources().getString(R.string.app_dirty).equals("1")) {
72             this.resultView.setText(R.string.warning_changed);
73         }
```

```
74
75          final Resources resources = getResources();
76          this.runner = BenchmarkRunner.INSTANCE
77                  .setAppChecksum(resources.getText(R.string.app_checksum))
78                  .setAppRevision(resources.getText(R.string.app_revision))
79                  .setCacheDir(getCacheDir());
80
81          File sd = Environment.getExternalStorageDirectory();
82          dataDir = new File(sd, "results");
83          dataDir.mkdir();
84
85          this.controller = new BenchmarkController(this, dataDir);
86
87          this.socketCommunicator = new SocketCommunicator();
88
89          File configFile = new File(
90                  Environment.getExternalStorageDirectory(),
91                  "nativebenchmark_setup.json"
92          );
93          try {
94              if (!configFile.exists()) {
95                  InputStream templateStream = getResources()
96                          .openRawResource(R.raw.setup);
97                  OutputStream configFileStream = Utils.makeOutputStream
98                          (configFile, false);
99                  Utils.copyStream(templateStream, configFileStream);
100             }
101             this.configuration = ToolConfig.readConfigFile();
102         } catch (Exception e) {
103             String msg = getResources().getString(R.string.config_error);
104             Log.e(TAG, msg, e);
105             displayMessage(ApplicationState.State.INIT_FAIL, msg);
106         }
107
108         if (this.configuration != null) {
109             initSpinner(this.configuration);
110             if (allocationArray == null) {
111                 allocationArray = new byte[1024 * 1024 * 100];
112             }
113         }
114         this.socketCommunicator.startServer(this.controller, this.runner);
115     }
116
117     public void onDestroy() {
118         socketCommunicator.stopServer();
119         super.onDestroy();
120     }
121
122     private void initSpinner(Map<String, ToolConfig> conf) {
123         Spinner spinner = (Spinner) findViewById(R.id.config_spinner);
124         String keys[] = conf.keySet().toArray(new String[1]);
125         ArrayAdapter<CharSequence> adapter = new ArrayAdapter(this, android.R
126                 .layout.simple_spinner_item, keys);
```

```
127
128         int indexOfDefault = -1;
129         while (++indexOfDefault < keys.length &&
130                 !keys[indexOfDefault].equals("default")) ;
131
132         adapter.setDropDownViewResource(android.R.layout
133                 .simple_spinner_dropdown_item);
134         spinner.setAdapter(adapter);
135         spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
136             public void onItemSelected(
137                     AdapterView<?> parent, View view,
138                     int pos, long id) {
139                 selectedConfiguration = (String) parent.getItemAtPosition(pos);
140             }
141
142             public void onNothingSelected(AdapterView<?> parent) {
143             }
144         });
145
146         spinner.setSelection(indexOfDefault);
147
148     }
149
150     public void displayMessage(ApplicationState.State state, String message) {
151         displayMessage(state.stringId, message);
152     }
153
154     public void displayMessage(int id) {
155         this.resultView.setText(id);
156     }
157
158     public void displayMessage(String message) {
159         this.resultView.setText(message);
160     }
161
162     public void displayMessage(int id, String message) {
163         this.resultView.setText(getResources().getString(id) + " " + message);
164     }
165
166     private class StateChanger implements Runnable {
167         public void run() {
168             ApplicationState.DetailedState detailedState = controller
169                     .getState();
170             ApplicationState.State state = detailedState.state;
171             String message = detailedState.message;
172
173             if (message == null) {
174                 displayMessage(state.stringId);
175             } else {
176                 displayMessage(state.stringId, message);
177             }
178             switch (state) {
179                 case MEASURING_STARTED:
```

```
                    getWindow().addFlags(WindowManager.LayoutParams
                            .FLAG_KEEP_SCREEN_ON);
                    expPick.setEnabled(false);
                    numPick.setEnabled(false);
                    switchButton(button);
                    state = ApplicationState.State.MILESTONE;
                    break;
                case MILESTONE:
                    break;
                case ERROR:
                    displayMessage(ApplicationState.State.ERROR, message);
                    // intended fallthrough
                case INTERRUPTED:
                    // intended fallthrough
                case MEASURING_FINISHED:
                    getWindow().clearFlags(WindowManager.LayoutParams
                            .FLAG_KEEP_SCREEN_ON);
                    stateThread.interrupt();
                    notifyFinished();
                    // intended fallthrough
                case INITIALISED:
                    resetButton(button);
                    numPick.setEnabled(true);
                    expPick.setEnabled(true);
                    break;
                case INIT_FAIL:
            }
        }
    }

    public boolean userWantsToRetry(final Exception e) {
        runOnUiThread(new Runnable() {
            public void run() {
                DialogFragment dialog = new RetryDialog(e.getMessage());
                dialog.show(getFragmentManager(), "foo");
            }
        });
        boolean waiting = true;
        while (waiting) {
            synchronized (this) {
                try {
                    this.wait();
                    waiting = false;
                } catch (InterruptedException ie) {
                    waiting = true;
                }
            }
        }
        return this.retry;
    }

    private void setRetry(boolean answer) {
        this.retry = answer;
```

```
233         synchronized (this) {
234             this.notify();
235         }
236     }
237
238     private void resetButton(Button button) {
239         button.setText(R.string.start_task);
240         button.setOnClickListener(new View.OnClickListener() {
241             public void onClick(View v) {
242                 startMeasuring(v);
243             }
244         });
245     }
246
247     private void switchButton(Button bt) {
248         bt.setText(R.string.end_task);
249         bt.setOnClickListener(
250                 new View.OnClickListener() {
251                     public void onClick(View v) {
252                         button.setText(R.string.end_task_confirmation);
253                         button.setOnClickListener(
254                                 new View.OnClickListener() {
255                                     public void onClick(View v) {
256                                         displayMessage(ApplicationState.State
257                                                 .INTERRUPTING.stringId);
258                                         controller.interruptMeasuring();
259                                     }
260                                 });
261                     }
262                 });
263     }
264
265
266     private boolean isChecked(int id) {
267         return ((Checkable) findViewById(id)).isChecked();
268     }
269
270     private String textValue(int id) {
271         return ((TextView) findViewById(id)).getText().toString();
272     }
273
274     public void startMeasuring(View view) {
275         this.runner
276                 .setRepetitions(repetitions)
277                 .setAllocatingRepetitions(Long.parseLong(textValue(R.id
278                         .alloc_reps)))
279                 .setBenchmarkSubstring(textValue(R.id.benchmark_substring)
280                         .toLowerCase())
281                 .setRunAllBenchmarks(isChecked(R.id.checkbox_long))
282                 .setRunAtMaxSpeed(isChecked(R.id.checkbox_max))
283                 .setBenchmarkSet(isChecked(R.id.run_alloc) ?
284                         BenchmarkRunner.BenchmarkSet.ALLOC :
285                         BenchmarkRunner.BenchmarkSet.NON_ALLOC);
```

```
286
287         stateThread = new Thread(
288                 new Runnable() {
289                     public void run() {
290                         while (!Thread.currentThread().isInterrupted()) {
291                             runOnUiThread(new StateChanger());
292                             try {
293                                 Thread.sleep(5000);
294                             } catch (InterruptedException e) {
295                                 break;
296                             }
297                         }
298                     }
299                 });
300
301         stateThread.start();
302         allocationArray = null;
303         controller.startMeasuring(this.runner, this.configuration.get
304                 (selectedConfiguration));
305     }
306
307     private void notifyFinished() {
308         Uri alarmSound = RingtoneManager.getDefaultUri(
309                 RingtoneManager.TYPE_NOTIFICATION);
310
311         Notification.Builder mBuilder =
312                 new Notification.Builder(this)
313                         .setSmallIcon(android.R.drawable.ic_media_play)
314                         .setContentTitle(getResources().getText(R.string
315                                 .measuring_finished))
316                         .setContentText(getResources().getText(R.string
317                                 .measuring_finished))
318                         .setSound(alarmSound);
319
320         Intent resultIntent = new Intent(this, BenchmarkSelector.class);
321
322         TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
323         stackBuilder.addParentStack(BenchmarkSelector.class);
324         stackBuilder.addNextIntent(resultIntent);
325         PendingIntent resultPendingIntent =
326                 stackBuilder.getPendingIntent(
327                         0,
328                         PendingIntent.FLAG_UPDATE_CURRENT
329                 );
330         NotificationManager mNotificationManager =
331                 (NotificationManager) getSystemService(Context
332                         .NOTIFICATION_SERVICE);
333         // mId allows you to update the notification later on.
334         mNotificationManager.notify(1, mBuilder.build());
335     }
336
337     private class RepsListener implements NumberPicker.OnValueChangeListener {
338         public void onValueChange(NumberPicker picker, int oldVal, int newVal) {
```

```
339             long exp = BenchmarkSelector.this.expPick.getValue();
340             long value = BenchmarkSelector.this.numPick.getValue();
341             while (exp-- > 0) {
342                 value *= 10;
343             }
344             repetitions = value;
345             repView.setText("" + repetitions);
346         }
347     }
348
349     private class RetryDialog extends DialogFragment {
350         private String message;
351
352         public RetryDialog(String message) {
353             this.message = message;
354         }
355
356         @Override
357         public Dialog onCreateDialog(Bundle savedInstanceState) {
358             // Use the Builder class for convenient dialog construction
359             AlertDialog.Builder builder = new AlertDialog.Builder(getActivity
360                     ());
361             builder.setMessage(getResources().getText(R.string
362                     .retry_question) + ":\n" + this.message)
363                     .setPositiveButton(R.string.retry_answer_positive, new
364                             DialogInterface.OnClickListener() {
365                                 public void onClick(DialogInterface dialog,
366                                                     int id) {
367                                     setRetry(true);
368                                 }
369                             })
370                     .setNegativeButton(R.string.retry_answer_negative, new
371                             DialogInterface.OnClickListener() {
372                                 public void onClick(DialogInterface dialog,
373                                                     int id) {
374                                     setRetry(false);
375                                 }
376                             });
377             // Create the AlertDialog object and return it
378             return builder.create();
379         }
380     }
381
382     private long repetitions;
383     private boolean retry;
384     private TextView textView, resultView, repView;
385     private NumberPicker numPick, expPick;
386     private Button button;
387     private Thread stateThread;
388     private static byte[] allocationArray;
389     private File dataDir;
390     private BenchmarkController controller;
391     private SocketCommunicator socketCommunicator;
```

```
392     private BenchmarkRunner runner;
393
394     private String selectedConfiguration;
395
396     private static final String TAG = "BenchmarkSelector";
397
398     private Map<String, ToolConfig> configuration;
399
400     static {
401         System.loadLibrary("nativebenchmark");
402     }
403
404 }
```

Init.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3  import java.io.IOException;
4  import java.util.ArrayList;
5  import java.util.List;
6
7  public class Init {
8
9      private static final String TAG = "NativeBenchmark";
10     public static final int CPUFREQ = 400000;
11     public static final int CPUFREQ_MAX = 1000000;
12
13     public static List<String> initScript(int cpufreq) {
14         List<String> script = new ArrayList<String>();
15         script.add(
16                 "echo \"userspace\" > " +
17                         "/sys/devices/system/cpu/cpu0/cpufreq" +
18                         "/scaling_governor");
19         script.add(
20                 "echo \"" + cpufreq + "\" > " +
21                         "/sys/devices/system/cpu/cpu0/cpufreq" +
22                         "/scaling_setspeed");
23         return script;
24
25     }
26
27     public static void initEnvironment(boolean maxSpeed) throws IOException {
28         ShellEnvironment.runAsRoot(initScript(maxSpeed ? CPUFREQ_MAX :
29                 CPUFREQ));
30     }
31 }
```

L.java

```java
package fi.helsinki.cs.tituomin.nativebenchmark;

public class L {
    // Set L.og to false to statically remove
    // debugging logging statements from
    // measuring code.
    public static final boolean og = false;
}
```

LogAccess.java

```java
package fi.helsinki.cs.tituomin.nativebenchmark;

import android.util.Log;

import java.io.File;
import java.io.IOException;
import java.util.regex.Pattern;

public class LogAccess {

    private static final String LOGCAT_COMMAND =
            "logcat -f %s " +
                    "-b main -b system -b radio -b events " +
                    "-v time";

    public static void start(File dir) throws IOException {
        currentRunId = Utils.getUUID();
        mark(START, currentRunId);
        String filename = new File(dir, filename()).getAbsolutePath();
        String command = String.format(LOGCAT_COMMAND, filename);
        logcatProcess = Runtime.getRuntime().exec(command);
    }

    public static void end() {
        if (logcatProcess != null) {
            mark(END, currentRunId);
            logcatProcess.destroy();
            logcatProcess = null;
        }
    }

    private static void mark(String type, String id) {
        Log.println(LOGLEVEL, TAG, marker(type) + id);
    }
```

```
35
36      private static String marker(String type) {
37          return "[" + type + "] ";
38      }
39
40      public static String filename() {
41          return "log-" + currentRunId + ".txt";
42      }
43
44      private static Pattern makeMarkerPattern(String type) {
45          return Pattern.compile("[-:. [0-9]]+ I/" + TAG + "\\([ 0-9]+\\): \\["
46                  + type + "\\] " + currentRunId);
47      }
48
49      private static final int LOGLEVEL = Log.INFO;
50      private static final String TAG = "LogAccess";
51      private static final String START = "Start";
52      private static final String END = "End";
53      private static String currentRunId;
54      private static Process logcatProcess;
55  }
```

measuringtool/AllocatingBenchmarkLongRunningWrapper.java

```
1   package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3   import java.io.IOException;
4
5   import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
6   import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
7
8   public class AllocatingBenchmarkLongRunningWrapper extends
9           AllocatingBenchmarkWrapper {
10
11      public AllocatingBenchmarkLongRunningWrapper(Benchmark b, long r) {
12          super(b, r);
13      }
14
15      private static final long MAX_REPS = Long.MAX_VALUE;
16
17      public void begin(MeasuringTool tool) throws InterruptedException,
18              IOException {
19          Benchmark benchmark = getBenchmark();
20          init(benchmark);
21          tool.putMeasurement("repetitions", this.repetitions + "");
22          tool.start(this);
23          tool.finishMeasurement();
24      }
25
```

```
26      public void run() {
27          // This method ensures the garbage collector is run
28          // every benchmark.maxrepetitions iteration
29          // but otherwise the measurement is
30          // run for a period long enough for profiling.
31          Benchmark benchmark = getBenchmark();
32          long interval = BenchmarkRegistry.CHECK_INTERRUPTED_INTERVAL;
33          long division, remainder;
34          long repetitions = MAX_REPS;
35
36          division = repetitions / interval + 1;
37          remainder = repetitions % interval + 1;
38
39          while (--division != 0) {
40              interval = BenchmarkRegistry.CHECK_INTERRUPTED_INTERVAL + 1;
41              while (--interval != 0) {
42                  try {
43                      benchmark.run();
44                      System.gc();
45                      Thread.sleep(GC_PAUSE_MS);
46                  } catch (InterruptedException e) {
47                      setInterrupted();
48                      return;
49                  } catch (Exception e) {
50                      setException(e);
51                      return;
52                  }
53              }
54              if (Thread.currentThread().isInterrupted()) {
55                  setInterrupted();
56                  return;
57              }
58          }
59
60          while (--remainder != 0) {
61              try {
62                  benchmark.run();
63                  System.gc();
64                  Thread.sleep(GC_PAUSE_MS);
65              } catch (InterruptedException e) {
66                  setInterrupted();
67                  return;
68              } catch (Exception e) {
69                  setException(e);
70                  return;
71              }
72          }
73
74      }
75
76  }
```

measuringtool/AllocatingBenchmarkShortRunningWrapper.java

```
package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;

import java.io.IOException;

import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;

public class AllocatingBenchmarkShortRunningWrapper extends
        AllocatingBenchmarkWrapper {

    public AllocatingBenchmarkShortRunningWrapper(Benchmark b, long r) {
        super(b, r);
    }

    private static final long MULTIPLIER = 25;

    public void begin(MeasuringTool tool) throws InterruptedException,
            IOException {
        Benchmark benchmark = getBenchmark();
        init(benchmark);
        long reps = MULTIPLIER;
        reps += 1;
        while (--reps != 0) {
            tool.putMeasurement("repetitions", this.repetitions + "");
            tool.putMeasurement("multiplier", MULTIPLIER + "");
            try {
                tool.start(getBenchmark());
                tool.finishMeasurement();
                System.gc();
                Thread.sleep(GC_PAUSE_MS);
            } catch (InterruptedException e) {
                setInterrupted();
                return;
            } catch (Exception e) {
                setException(e);
                return;
            }
        }
    }
}
```

measuringtool/AllocatingBenchmarkWrapper.java

```
package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;

import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
```

```
4
5  public class AllocatingBenchmarkWrapper extends RunningWrapper {
6
7      public AllocatingBenchmarkWrapper(Benchmark b, long repetitions) {
8          super(b);
9          this.repetitions = repetitions;
10     }
11
12     public static final int GC_PAUSE_MS = 400;
13
14     public void init(Benchmark benchmark) {
15         super.init(benchmark);
16         benchmark.setRepetitions(repetitions);
17     }
18
19     protected long repetitions;
20
21 }
```

measuringtool/BasicOption.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3  import android.util.Pair;
4
5  public class BasicOption implements MeasuringOption {
6
7      public BasicOption(OptionSpec type) {
8          this(type, null);
9      }
10
11     public BasicOption(OptionSpec type, String value) {
12         this.type = type;
13         this.value = value;
14     }
15
16     // -----
17
18     public OptionSpec id() {
19         return this.type;
20     }
21
22     public void set(String value) {
23         this.value = value;
24     }
25
26     public OptionSpec type() {
27         return this.type;
28     }
```

```
29
30      public String value() {
31          return value;
32      }
33
34      public Pair<String, String> toStringPair() {
35          return new Pair<String, String>(this.type.id(), this.value);
36      }
37
38      public String toString() {
39          return this.type() + " " + this.value();
40      }
41
42      // ----
43
44      private OptionSpec type;
45      private String value;
46
47      // ----
48
49  }
```

measuringtool/CommandlineTool.java

```
1   package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3   import android.util.Log;
4
5   import java.io.IOException;
6   import java.text.DateFormat;
7   import java.util.ArrayList;
8   import java.util.Date;
9   import java.util.LinkedList;
10  import java.util.List;
11  import java.util.Random;
12
13  import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
14  import fi.helsinki.cs.tituomin.nativebenchmark.ShellEnvironment;
15
16
17  public abstract class CommandlineTool extends MeasuringTool {
18
19      private static final long REPETITIONS = Long.MAX_VALUE;
20
21      public CommandlineTool(int rounds, long repetitions, long allocreps,
22                             boolean warmup, boolean x) throws IOException,
23              InterruptedException {
24          super(rounds, REPETITIONS, allocreps, false, x);
25          this.filenames = new ArrayList<String>();
```

```
26      }
27
28      protected abstract String command();
29
30      protected String formatParameter(MeasuringOption option) {
31          throw new UnsupportedOptionException();
32      }
33
34      protected String formatDefaultParameter(MeasuringOption option) {
35          if (option.type() == OptionSpec.COMMAND_STRING) {
36              return option.value();
37          } else {
38              return formatParameter(option);
39          }
40      }
41
42      public void initCommand() {
43          List<String> commandline = new LinkedList<String>();
44          commandline.addAll(generateCommandlineArguments());
45          this.command = "";
46          for (String s : commandline) {
47              this.command += s + " ";
48          }
49      }
50
51      protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec> options) {
52          options = super.specifyAllowedOptions(options);
53          options.add(OptionSpec.COMMAND_STRING);
54          return options;
55      }
56
57      protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
58                                                              options) {
59          options.add(new BasicOption(OptionSpec.COMMAND_STRING, command()));
60          return options;
61      }
62
63      protected void init() throws IOException, InterruptedException {
64          super.init();
65          if (!ShellEnvironment.runAsRoot(initScript())) {
66              throw new IOException("Error executing as root.");
67          }
68      }
69
70      protected abstract List<String> initScript();
71
72      public boolean isLongRunning() {
73          return true;
74      }
75
76      public void start(Runnable benchmark)
77              throws InterruptedException, IOException {
78          if (Thread.interrupted()) {
```

```
79              throw new InterruptedException();
80          }
81
82          initCommand();
83          BenchmarkRegistry.resetInterruptFlag();
84          Thread benchmarkThread = new Thread(benchmark);
85          Random r = new Random();
86          int delay = r.nextInt(20);
87
88          benchmarkThread.start();
89          Thread.sleep(delay);
90
91          try {
92              ShellEnvironment.run(this.command);
93          } catch (InterruptedException e) {
94              throw e;
95          } finally {
96              benchmarkThread.interrupt();
97              BenchmarkRegistry.interruptNative();
98              benchmarkThread.join();
99          }
100     }
101
102     public void setFilename(String name, String path) {
103         filenames.add(path + "/" + name);
104         putMeasurement("Filename", name);
105     }
106
107     public List<String> getFilenames() {
108         return filenames;
109     }
110
111
112     public void setUUID(String uuid) {
113         putMeasurement("UUID", uuid);
114     }
115
116     public List<String> generateCommandlineArguments() {
117         List<String> result = new LinkedList<String>();
118         for (OptionSpec type : allowedOptions) {
119             MeasuringOption option = options.get(type);
120
121             if (option == null) {
122                 Log.v("mt", type + " is null");
123             } else {
124                 result.add(formatDefaultParameter(option));
125             }
126         }
127         return result;
128     }
129
130     private Date startDate;
131     private Date endDate;
```

```
132     private long startTime;
133     private String command;
134 
135     private List<String> filenames;
136     private static final DateFormat dateFormat = DateFormat
137             .getDateTimeInstance();
138 
139 }
```

measuringtool/JavaSystemNanoResponseTimeRecorder.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2  
3  import java.io.IOException;
4  
5  public class JavaSystemNanoResponseTimeRecorder extends ResponseTimeRecorder {
6  
7      public JavaSystemNanoResponseTimeRecorder(
8              int i,
9              long reps,
10             long allocreps,
11             boolean warmup,
12             boolean runAllBenchmarks
13     ) throws IOException, InterruptedException {
14         super(i, reps, allocreps, warmup, runAllBenchmarks);
15     }
16 
17     public void start(Runnable benchmark)
18             throws InterruptedException, IOException {
19         long endTime, startTime;
20         startTime = System.nanoTime();
21         benchmark.run();
22         endTime = System.nanoTime();
23         String delta = "" + (endTime - startTime);
24         putMeasurement("response_time", delta);
25         putMeasurement("time_unit", "nanoseconds");
26     }
27 }
```

measuringtool/LinuxPerfRecordTool.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2  
3  import java.io.File;
```

```
4 import java.io.IOException;
5 import java.util.LinkedList;
6 import java.util.List;
7
8 import fi.helsinki.cs.tituomin.nativebenchmark.Utils;
9
10 public class LinuxPerfRecordTool extends CommandlineTool {
11
12     public LinuxPerfRecordTool
13             (
14                     int rounds,
15                     long repetitions,
16                     long allocreps,
17                     boolean warmup,
18                     boolean runAllBenchmarks) throws
19             IOException, InterruptedException {
20         super(rounds, repetitions, allocreps, warmup, runAllBenchmarks);
21     }
22
23     protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec> options) {
24         options = super.specifyAllowedOptions(options);
25         options.add(OptionSpec.OUTPUT_FILEPATH);
26         options.add(OptionSpec.MEASURE_LENGTH); // must be last
27         return options;
28     }
29
30     protected List<String> initScript() {
31         List<String> commands = new LinkedList<String>();
32         commands.add("echo \"0\" > /proc/sys/kernel/kptr_restrict");
33         commands.add("echo \"-1\" > /proc/sys/kernel/perf_event_paranoid");
34         return commands;
35     }
36
37     protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
38                                                    options) {
39         options = super.defaultOptions(options);
40         String uuid = Utils.getUUID();
41         String filename = generateFilename(uuid);
42         String basePath = getPerfDir().getPath() + "/";
43         setFilename(filename, basePath);
44         setUUID(uuid);
45         options.add(new BasicOption(OptionSpec.OUTPUT_FILEPATH, basePath +
46                 filename));
47         return options;
48     }
49
50     protected void init() throws IOException, InterruptedException {
51         super.init();
52         getPerfDir().mkdir();
53     }
54
55     private File getPerfDir() {
56         return new File(getDataDir(), "perf");
```

```
57      }
58
59      protected String command() {
60          return "perf record -a -g";
61      }
62
63      private String generateFilename(String uuid) {
64          return "perf-" + uuid + ".data";
65      }
66
67      public String formatParameter(MeasuringOption option) {
68          if (option.type() == OptionSpec.OUTPUT_FILEPATH) {
69              return "--output=" + option.value();
70          } else if (option.type() == OptionSpec.MEASURE_LENGTH) {
71              return "sleep " + option.value();
72          }
73          return super.formatParameter(option);
74      }
75  }
```

measuringtool/MeasuringOption.java

```
1   package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3   import android.util.Pair;
4
5   public interface MeasuringOption {
6
7       public void set(String value);
8
9       public OptionSpec id();
10
11      public String value();
12
13      public OptionSpec type();
14
15      public Pair<String, String> toStringPair();
16  }
```

measuringtool/MeasuringTool.java

```
1   package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3
```

```
4 import android.util.Log;
5 import android.util.Pair;
6
7 import java.io.File;
8 import java.io.IOException;
9 import java.text.SimpleDateFormat;
10 import java.util.ArrayList;
11 import java.util.Date;
12 import java.util.HashMap;
13 import java.util.HashSet;
14 import java.util.LinkedList;
15 import java.util.List;
16 import java.util.Locale;
17 import java.util.Map;
18 import java.util.Set;
19
20 import fi.helsinki.cs.tituomin.nativebenchmark.ApplicationState;
21 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
22 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
23 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkResult;
24
25 public abstract class MeasuringTool implements Runnable {
26
27     public MeasuringTool
28             (
29                     int rounds,
30                     long repetitions,
31                     long allocRepetitions,
32                     boolean warmup,
33                     boolean runAllBenchmarks
34             ) throws
35             IOException, InterruptedException {
36         clearMeasurements();
37         specifyOptions();
38         this.rounds = rounds;
39         this.repetitions = repetitions;
40         this.allocRepetitions = allocRepetitions;
41         this.warmup = warmup;
42         this.explicitGC = !warmup;
43         this.runAllBenchmarks = runAllBenchmarks;
44         init();
45     }
46
47     public static synchronized void userInterrupt() {
48         userInterrupted = true;
49         BenchmarkRegistry.interruptNative();
50     }
51
52     public static synchronized boolean userInterrupted() {
53         if (userInterrupted == true) {
54             userInterrupted = false;
55             return true;
56         }
```

```
57          return false;
58      }
59
60      protected void init() throws IOException, InterruptedException {
61      }
62
63      protected abstract void start(Runnable benchmark)
64              throws InterruptedException, IOException;
65
66      public boolean isLongRunning() {
67          return false;
68      }
69
70      public RunningWrapper wrap(Benchmark benchmark) {
71          if (!benchmark.isAllocating()) {
72              // Default running algorithm
73              return new RunningWrapper(benchmark);
74          } else {
75              // the benchmark does allocations, we have
76              // to limit the amount of loops -> compensate
77              // by repeating the loop many times
78              if (isLongRunning()) {
79                  return new AllocatingBenchmarkLongRunningWrapper(benchmark,
80                          allocRepetitions);
81              } else {
82                  return new AllocatingBenchmarkShortRunningWrapper(benchmark,
83                          allocRepetitions);
84              }
85          }
86      }
87
88      public void startMeasuring(Benchmark benchmark) throws
89              InterruptedException, IOException, RunnerException {
90          String benchmarkName = benchmark.getClass().getSimpleName();
91          clearMeasurements();
92          setDefaultOptions();
93          benchmark.setRepetitions(this.repetitions);
94          RunningWrapper wrapper = wrap(benchmark);
95          Date start = null, end = null;
96
97          start = new Date();
98          wrapper.begin(this);
99          end = new Date();
100
101         putMeasurement("start", DATEFORMAT.format(start));
102         putMeasurement("end", DATEFORMAT.format(end));
103
104         if (wrapper.wasInterrupted() && userInterrupted()) {
105             throw new InterruptedException("Interrupted by user");
106         }
107         if (wrapper.exceptionWasThrown()) {
108             throw new RunnerException(wrapper.getException());
109         }
```

```
110     }
111
112     public class RunnerException extends Exception {
113         public RunnerException(Throwable t) {
114             super(t);
115         }
116     }
117
118
119     public void run() {
120         try {
121             start(benchmark);
122         } catch (InterruptedException e) {
123             Log.e("BenchmarkRunner", "Measuring was interrupted ", e);
124         } catch (IOException e) {
125             Log.e("BenchmarkRunner", "IOException", e);
126         }
127     }
128
129     public boolean explicitGC() {
130         return this.explicitGC;
131     }
132
133     public boolean runAllBenchmarks() {
134         return this.runAllBenchmarks;
135     }
136
137     public void setExplicitGC(boolean e) {
138         this.explicitGC = e;
139     }
140
141     protected abstract List<MeasuringOption>
142     defaultOptions(List<MeasuringOption> container);
143
144     protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec>
145                                                                 container) {
146         return container;
147     }
148
149     protected void specifyOptions() {
150         this.allowedOptions = specifyAllowedOptions(
151                 new LinkedList<OptionSpec>());
152
153         if (this.requiredOptions == null) {
154             this.requiredOptions = new HashSet<OptionSpec>();
155         }
156         for (OptionSpec o : this.allowedOptions) {
157             if (o.required()) {
158                 this.requiredOptions.add(o);
159             }
160         }
161     }
162
```

```
163     protected void setDefaultOptions() {
164         for (MeasuringOption op : defaultOptions(
165                 new LinkedList<MeasuringOption>())) {
166             setOption(op);
167         }
168         ;
169     }
170 
171     public MeasuringTool set(String id, String value) {
172         setOption(new BasicOption(OptionSpec.byId(id), value));
173         return this;
174     }
175 
176     public MeasuringTool set(OptionSpec spec, String value) {
177         setOption(new BasicOption(spec, value));
178         return this;
179     }
180 
181     public String getOption(OptionSpec id) {
182         return this.options.get(id).value();
183     }
184 
185     public void setDescription(String d) {
186         this.description = d;
187     }
188 
189     public void setOption(MeasuringOption option) {
190         if (this.allowedOptions == null) {
191             specifyOptions();
192         }
193         if (!allowedOptions.contains(option.type())) {
194             throw new UnsupportedOptionException();
195         } else {
196             this.options =
197                     options != null ?
198                             options :
199                             new HashMap<OptionSpec, MeasuringOption>();
200 
201             this.options.put(option.type(), option);
202         }
203     }
204 
205     public void setBenchmark(Benchmark b) {
206         benchmark = b;
207     }
208 
209     public boolean ignore() {
210         return false;
211     }
212 
213     private boolean hasRequiredOptions() {
214         return options.keySet().containsAll(requiredOptions);
215     }
```

```
216
217     protected List<OptionSpec> allowedOptions;
218     protected Set<OptionSpec> requiredOptions;
219     protected Map<OptionSpec, MeasuringOption> options;
220
221     private List<ApplicationState> observers;
222
223     private BenchmarkResult currentMeasurement;
224     private List<BenchmarkResult> measurements;
225
226     protected long repetitions;
227
228     protected void putMeasurement(String key, String value) {
229         currentMeasurement.put(key, value);
230     }
231
232     protected void finishMeasurement() {
233         currentMeasurement = new BenchmarkResult();
234         measurements.add(currentMeasurement);
235     }
236
237     public List<BenchmarkResult> getMeasurements() {
238         for (BenchmarkResult measurement : measurements) {
239             if (this.options != null) {
240                 for (MeasuringOption op : options.values()) {
241                     measurement.put(
242                             op.toStringPair().first,
243                             op.toStringPair().second);
244                 }
245             }
246         }
247         return this.measurements;
248     }
249
250     public void clearMeasurements() {
251         this.currentMeasurement = new BenchmarkResult();
252         this.measurements = new LinkedList<BenchmarkResult>();
253         measurements.add(currentMeasurement);
254     }
255
256     private static final List<String> emptyList = new ArrayList<String>();
257
258     public List<String> getFilenames() {
259         return emptyList;
260     }
261
262     public int getRounds() {
263         return rounds;
264     }
265
266     public static void setDataDir(File dir) {
267         dataDir = dir;
268     }
```

```
269
270     public static File getDataDir() {
271         return dataDir;
272     }
273
274     public void setFilter(String substring) {
275         filterSubstring = substring;
276     }
277
278     public String getFilter() {
279         return filterSubstring;
280     }
281
282     public List<Pair<String, String>> configuration() {
283         List<Pair<String, String>> pairs = new ArrayList<Pair<String,
284                 String>>();
285         pairs.add(new Pair<String, String>("tool", this.getClass()
286                 .getSimpleName()));
287         pairs.add(new Pair<String, String>("repetitions", "" + this
288                 .repetitions));
289         pairs.add(new Pair<String, String>("description", this.description));
290         pairs.add(new Pair<String, String>("warmup", "" + this.warmup));
291         if (options != null) {
292             for (MeasuringOption opt : options.values()) {
293                 pairs.add(new Pair<String, String>(opt.type().id(), opt.value
294                         ()));
295             }
296         }
297         return pairs;
298     }
299
300     public boolean isWarmupRound() {
301         return warmup;
302     }
303
304     private static File dataDir;
305
306     private Benchmark benchmark;
307     private int rounds;
308     private long allocRepetitions;
309     private String description;
310     private String filterSubstring;
311     protected boolean warmup;
312     private boolean explicitGC;
313     private boolean runAllBenchmarks;
314     private static boolean userInterrupted = false;
315     private final static String TAG = "MeasuringTool";
316
317     public static class UnsupportedOptionException extends RuntimeException {
318     }
319
320     private static SimpleDateFormat DATEFORMAT = new SimpleDateFormat("MM-dd " +
321             "HH:mm:ss.SSS", Locale.US);
```

```
}
```

measuringtool/MockCommandlineTool.java

```
package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;

import java.io.IOException;
import java.util.List;

public class MockCommandlineTool extends CommandlineTool {

    public MockCommandlineTool(int i, long reps) throws IOException,
            InterruptedException {
        super(i, reps, reps, false, false);
    }

    protected List<String> initScript() {
        return null;
    }

    public boolean ignore() {
        return true;
    }

    protected String command() {
        return "cat /dev/null";
    }

}
```

measuringtool/OptionSpec.java

```
package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;

import java.util.HashMap;
import java.util.Map;

public enum OptionSpec {

    COMMAND_STRING("Command run", "command_string", true),
    OUTPUT_FILEPATH("Output path", "output_filepath", true),
    MEASURE_LENGTH("Measuring time (sec)", "measure_length", true),
    VARIABLE("Variable parameter in benchmark", "variable", false),
```

```
12      CPUFREQ("Fixed CPU frequency", "cpu_freq", true);
13
14      OptionSpec(String name, String id, boolean required) {
15          this.name = name;
16          this.id = id;
17          this.required = required;
18          put(id, this);
19      }
20
21      public String id() {
22          return id;
23      }
24
25      public boolean required() {
26          return required;
27      }
28
29      public static OptionSpec byId(String id) {
30          return specs.get(id);
31      }
32
33      private static void put(String id, OptionSpec value) {
34          getSpecs().put(id, value);
35      }
36
37      public static Map<String, OptionSpec> getSpecs() {
38          if (specs == null) {
39              specs = new HashMap<String, OptionSpec>();
40          }
41          return specs;
42      }
43
44      private String name;
45      private String id;
46      private boolean required;
47
48      private static Map<String, OptionSpec> specs;
49
50  }
```

measuringtool/PlainRunner.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3  import java.io.IOException;
4  import java.util.List;
5
6  public class PlainRunner extends MeasuringTool {
7
```

```
8       public PlainRunner(int i, long reps, long allocreps) throws IOException,
9               InterruptedException {
10          super(i, reps, allocreps, true, false);
11      }
12
13      protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
14                                                              options) {
15          return options;
16      }
17
18      protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec> options) {
19          options = super.specifyAllowedOptions(options);
20          options.add(OptionSpec.CPUFREQ);
21          return options;
22      }
23
24      public boolean explicitGC() {
25          return false;
26      }
27
28      public boolean ignore() {
29          return true;
30      }
31
32      public long repetitions() {
33          return 10000;
34      }
35
36      public void start(Runnable benchmark)
37              throws InterruptedException, IOException {
38          benchmark.run();
39      }
40  }
```

measuringtool/ResponseTimeRecorder.java

```
1   package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3   import android.os.SystemClock;
4
5   import java.io.IOException;
6   import java.util.List;
7
8   public class ResponseTimeRecorder extends MeasuringTool {
9
10      public ResponseTimeRecorder(
11              int rounds,
12              long reps,
13              long allocreps,
```

```
14              boolean warmup,
15              boolean x)
16              throws IOException, InterruptedException {
17          super(rounds, reps, allocreps, warmup, x);
18      }
19
20      protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
21                                                              options) {
22          // no options needed, time is returned as is (not in extra file)
23          return options;
24      }
25
26      public boolean ignore() {
27          return warmup;
28      }
29
30      public void start(Runnable benchmark)
31              throws InterruptedException, IOException {
32          long endTime, startTime;
33          startTime = SystemClock.uptimeMillis();
34          benchmark.run();
35          endTime = SystemClock.uptimeMillis();
36          String delta = "" + (endTime - startTime);
37          putMeasurement("response_time", delta);
38          putMeasurement("time_unit", "milliseconds");
39      }
40
41  }
```

measuringtool/RunningWrapper.java

```
1   package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3   import java.io.IOException;
4
5   import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
6
7   public class RunningWrapper implements Runnable {
8       private Benchmark benchmark;
9       private Exception exceptionThrown;
10      private boolean interrupted;
11      private long repetitions;
12
13      public RunningWrapper(Benchmark b) {
14          benchmark = b;
15      }
16
17      protected void setException(Exception e) {
18          exceptionThrown = e;
```

```
19      }
20
21      protected void setInterrupted() {
22          interrupted = true;
23      }
24
25      public boolean exceptionWasThrown() {
26          return exceptionThrown != null;
27      }
28
29      public void setRepetitions(long r) {
30          repetitions = r;
31      }
32
33      public Benchmark getBenchmark() {
34          return benchmark;
35      }
36
37      public long getRepetitions() {
38          return repetitions;
39      }
40
41      public boolean wasInterrupted() {
42          return interrupted;
43      }
44
45      public Exception getException() {
46          return exceptionThrown;
47      }
48
49      public void init(Benchmark benchmark) {
50          interrupted = false;
51          exceptionThrown = null;
52      }
53
54      public void run() {
55          benchmark.run();
56      }
57
58      public void begin(MeasuringTool tool) throws InterruptedException,
59              IOException {
60          init(benchmark);
61          tool.start(this);
62          if (Thread.currentThread().isInterrupted()) {
63              setInterrupted();
64          }
65      }
66
67  }
```

MockObject.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3  public class MockObject {
4
5      public boolean jbooleanField;
6      public byte jbyteField;
7      public char jcharField;
8      public double jdoubleField;
9      public float jfloatField;
10     public int jintField;
11     public long jlongField;
12     public short jshortField;
13     public Object jobjectField;
14
15     public static boolean jbooleanStaticField;
16     public static byte jbyteStaticField;
17     public static char jcharStaticField;
18     public static double jdoubleStaticField;
19     public static float jfloatStaticField;
20     public static int jintStaticField;
21     public static long jlongStaticField;
22     public static short jshortStaticField;
23     public static Object jobjectStaticField;
24
25     public MockObject() {
26         jbooleanField = true;
27         jbyteField = 1;
28         jcharField = 2;
29         jdoubleField = 1.2;
30         jfloatField = 3.1f;
31         jintField = 3;
32         jlongField = 4;
33         jshortField = 5;
34         jobjectField = this;
35
36         jbooleanStaticField = true;
37         jbyteStaticField = 1;
38         jcharStaticField = 2;
39         jdoubleStaticField = 1.2;
40         jfloatStaticField = 3.1f;
41         jintStaticField = 3;
42         jlongStaticField = 4;
43         jshortStaticField = 5;
44         jobjectStaticField = this;
45     }
46 }
```

ShellEnvironment.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3  import android.util.Log;
4
5  import java.io.BufferedReader;
6  import java.io.DataOutputStream;
7  import java.io.IOException;
8  import java.io.InputStream;
9  import java.io.InputStreamReader;
10 import java.util.List;
11
12 public class ShellEnvironment {
13
14     // Thank you http://muzikant-android.blogspot
15     // .fi/2011/02/how-to-get-root-access-and-execute.html
16     public static boolean runAsRoot(List<String> commands) throws IOException {
17         if (commands == null) {
18             return true;
19         }
20         boolean retval = false;
21         DataOutputStream os = null;
22         Process suProcess = null;
23         try {
24             if (null != commands && commands.size() > 0) {
25                 suProcess = Runtime.getRuntime().exec("su");
26
27                 os = new DataOutputStream(suProcess.getOutputStream());
28
29                 // Execute commands that require root access
30                 for (String currCommand : commands) {
31                     os.writeBytes(currCommand + "\n");
32                     os.flush();
33                 }
34
35                 os.writeBytes("exit\n");
36                 os.flush();
37
38                 try {
39                     int suProcessRetval = suProcess.waitFor();
40                     if (255 != suProcessRetval) {
41                         // Root access granted
42                         retval = true;
43                     } else {
44                         // Root access denied
45                         retval = false;
46                     }
47                 } catch (Exception ex) {
48                     Log.e("ROOT", "Error executing root action", ex);
49                 }
50             }
```

```
51          } catch (IOException ex) {
52              Log.w("ROOT", "Can't get root access", ex);
53          } catch (SecurityException ex) {
54              Log.w("ROOT", "Can't get root access", ex);
55          } catch (Exception ex) {
56              Log.w("ROOT", "Error executing internal operation", ex);
57          } finally {
58              if (suProcess != null) {
59                  suProcess.destroy();
60              }
61          }
62          if (os != null) {
63              os.close();
64          }
65
66          return retval;
67      }
68
69      public static void run(String command)
70              throws IOException, InterruptedException {
71
72          Process process = null;
73          InputStream err = null;
74          try {
75              process = Runtime.getRuntime().exec(command);
76              err = process.getErrorStream();
77              process.waitFor();
78
79              if (process.exitValue() != 0) {
80                  String line;
81                  BufferedReader br = new BufferedReader(new InputStreamReader
82                          (err));
83                  StringBuilder sb = new StringBuilder("Command failed.\n");
84                  while ((line = br.readLine()) != null) {
85                      Log.e("tm", line);
86                      sb.append(line);
87                      sb.append("\n");
88                  }
89                  process.destroy();
90                  br.close();
91                  throw new IOException(sb.toString());
92              }
93          } catch (IOException e) {
94              throw e;
95          } catch (InterruptedException e) {
96              throw e;
97          } finally {
98              if (err != null) err.close();
99              if (process != null) process.destroy();
100         }
101     }
102
103 }
```

SocketCommunicator.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3
4  import android.util.Log;
5
6  import java.io.IOException;
7  import java.io.InputStream;
8  import java.io.OutputStream;
9  import java.io.PrintWriter;
10 import java.net.InetSocketAddress;
11 import java.net.ServerSocket;
12 import java.net.Socket;
13 import java.net.SocketTimeoutException;
14 import java.util.Map;
15
16 public class SocketCommunicator implements ApplicationStateListener {
17     /**
18      * Thread to initialize Socket connection
19      */
20     private final Runnable InitializeConnection = new Thread() {
21         @Override
22         public void run() {
23             // initialize server socket
24             while (!Thread.currentThread().isInterrupted()) {
25                 try {
26                     server = new ServerSocket();
27                     server.setReuseAddress(true);
28                     server.bind(new InetSocketAddress(38300));
29
30                     //attempt to accept a connection
31                     client = server.accept();
32
33                     SocketCommunicator.this.out = client.getOutputStream();
34                     SocketCommunicator.this.printWriter = new PrintWriter
35                             (out, true);
36                     SocketCommunicator.nis = client.getInputStream();
37                     try {
38                         SocketCommunicator.this.output(helpMessage);
39
40                         byte[] bytes = new byte[1024];
41                         int numRead = 0;
42                         while (numRead >= 0) {
43                             SocketCommunicator.this.output("[Awaiting input]");
44                             numRead = SocketCommunicator.nis.read(bytes, 0,
45                                     1024);
46                             if (numRead < 1) {
47                                 executeCommand("quit");
48                                 return;
49                             }
50                             receivedCommand = new String(bytes, 0, numRead)
```

```
                                .trim();
                        executeCommand(receivedCommand);
                    }
                } catch (IOException ioException) {
                    Log.e(SocketCommunicator.TAG, "" + ioException);
                }
            } catch (SocketTimeoutException e) {
                receivedCommand = "Connection has timed out! Please try " +
                        "again";
                Log.v(TAG, receivedCommand);
            } catch (IOException e) {
                Log.e(SocketCommunicator.TAG, "" + e);
            }

            if (client != null) {
                receivedCommand = "Connection was successful!";
                try {
                    server.close();
                } catch (IOException e) {
                    Log.e(TAG, "Error closing server connection.");
                } finally {
                    server = null;
                }
                Log.v(TAG, receivedCommand);
            }
        }
    }
};

private class StateChanger implements Runnable {
    public void run() {
    }
}

private void executeStart(String command) {
    String[] split = command.split(":");
    if (split.length < 2) {
        Log.e(TAG, "No configuration key provided.");
        return;
    }
    String configKey = split[1];
    Map<String, ToolConfig> configurations = null;
    ToolConfig config = null;
    try {
        configurations = ToolConfig.readConfigFile();
        config = configurations.get(configKey);
    } catch (Exception e) {
        Log.e(TAG, "Error reading configuration file.", e);
    }
    if (config == null) {
        ApplicationState.DetailedState ds = new ApplicationState
                .DetailedState(controller);
        ds.state = ApplicationState.State.ERROR;
```

```
104             ds.message = "Could not find configuration for " + configKey;
105             stateUpdated(ds);
106             return;
107         }
108         stateThread = new Thread(
109                 new Runnable() {
110                     public void run() {
111                         while (!Thread.currentThread().isInterrupted()) {
112                             ApplicationState.DetailedState detailedState =
113                                     controller.getState();
114                             ApplicationState.State state = detailedState.state;
115                             stateUpdated(detailedState);
116                             if (state == ApplicationState.State
117                                     .MEASURING_FINISHED ||
118                                     state == ApplicationState.State.ERROR) {
119                                 return;
120                             }
121                             try {
122                                 Thread.sleep(10000);
123                             } catch (InterruptedException e) {
124                                 break;
125                             }
126                         }
127                     }
128                 }
129         );
130 
131         stateThread.start();
132         this.controller.startMeasuring(this.runner, config);
133     }
134 
135     private void output(String message) {
136         synchronized (this) {
137             this.printWriter.println(message);
138         }
139     }
140 
141     public void executeCommand(String command) {
142         boolean executed = false;
143         command = command.trim();
144         if (command.length() == 0) {
145             return;
146         } else if (command.startsWith("start")) {
147             this.executeStart(command);
148             executed = true;
149         } else if (command.startsWith("end")) {
150             this.controller.interruptMeasuring();
151             executed = true;
152         } else if (command.startsWith("quit")) {
153             this.restartServer();
154             executed = true;
155         } else {
156             this.output(String.format("Unkown command %s", command));
```

```
157             Log.v(TAG, "" + command + " == unknown command.");
158         }
159         if (!executed) {
160             return;
161         }
162         this.output(String.format(
163                 "[Executed %s]", command));
164     }
165 
166     public void startServer(
167             BenchmarkController controller,
168             BenchmarkRunner runner)
169 
170     {
171         this.controller = controller;
172         this.runner = runner;
173         //initialize server socket in a new separate thread
174         this.serverThread = new Thread(InitializeConnection);
175         this.serverThread.start();
176         String msg = "Attempting to connect";
177         Log.v(TAG, msg);
178     }
179 
180     public boolean stopServer() {
181         try {
182             this.output("Stopping socket server.");
183             serverThread.interrupt();
184             if (SocketCommunicator.nis != null) {
185                 SocketCommunicator.nis.close();
186             }
187             if (this.out != null) {
188                 this.out.close();
189             }
190             if (server != null) {
191                 server.close();
192             }
193             return true;
194         } catch (IOException ec) {
195             Log.e(SocketCommunicator.TAG, "Cannot close server socket" + ec);
196             return false;
197         }
198     }
199 
200     public void restartServer() {
201         if (this.stopServer()) {
202             this.startServer(this.controller, this.runner);
203         }
204     }
205 
206     public void stateUpdated(ApplicationState.DetailedState state) {
207         this.output(state.toString());
208     }
209 
```

```
210     public static final String TAG = "SocketCommunicator";
211     public static final int TIMEOUT = 10;
212     private ServerSocket server = null;
213     private Socket client = null;
214     private OutputStream out;
215     private PrintWriter printWriter;
216     private String receivedCommand = null;
217     public static InputStream nis = null;
218     private BenchmarkController controller;
219     private Map<String, ToolConfig> configurations;
220     private BenchmarkRunner runner;
221     private Thread serverThread;
222     private Thread stateThread;
223 
224     private final String helpMessage = "\n" +
225             "Measuring application ready.\n" +
226             "Available commands:\n" +
227             "  start :CONFIG_KEY\n" +
228             "    Starts measuring with the configuration\n" +
229             "    loaded from nativebenchmark_setup.json file\n" +
230             "    under the top level key CONFIG_KEY.\n" +
231             "  end\n" +
232             "    Interrupts measuring.\n";
233 
234 }
```

ToolConfig.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark;
2  
3  import android.os.Environment;
4  import android.util.Log;
5  
6  import org.json.JSONArray;
7  import org.json.JSONException;
8  import org.json.JSONObject;
9  
10 import java.io.File;
11 import java.io.IOException;
12 import java.lang.reflect.Constructor;
13 import java.lang.reflect.InvocationTargetException;
14 import java.util.HashMap;
15 import java.util.Iterator;
16 import java.util.Map;
17 import java.util.NoSuchElementException;
18 
19 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool;
20 
21 public class ToolConfig implements Iterable<MeasuringTool> {
```

```
22
23      public static Map<String, ToolConfig> readConfigurations(String jsonConfig)
24              throws JSONException {
25          JSONObject cfgObject = new JSONObject(jsonConfig);
26          Map<String, ToolConfig> result = new HashMap<String, ToolConfig>();
27          Iterator i = cfgObject.keys();
28          while (i.hasNext()) {
29              String key = (String) i.next();
30              result.put(key, new ToolConfig(cfgObject.getJSONObject(key)));
31          }
32          return result;
33      }
34
35      public static Map<String, ToolConfig> readConfigFile() throws
36              IOException, JSONException {
37          String jsonContents = null;
38          File configFile = new File(
39                  Environment.getExternalStorageDirectory(),
40                  "nativebenchmark_setup.json"
41          );
42          jsonContents = Utils.readFileToString(configFile);
43          return ToolConfig.readConfigurations(jsonContents);
44      }
45
46      public ToolConfig(JSONObject job) {
47          this.globalOptions = job;
48          this.defaultAllocRepetitions = 400;
49      }
50
51      public String toString() {
52          return this.globalOptions.toString();
53      }
54
55      public Iterator<MeasuringTool> iterator() {
56          try {
57              return new ToolIterator();
58          } catch (JSONException e) {
59              Log.e("ToolConfig", "Error reading json config", e);
60          }
61          return null;
62      }
63
64      private class ToolIterator implements Iterator<MeasuringTool> {
65          private JSONArray toolArray;
66          private int currentToolIndex;
67
68          public ToolIterator() throws JSONException {
69              currentToolIndex = -1;
70              toolArray = globalOptions.getJSONArray("tools");
71          }
72
73          public boolean hasNext() {
74              return currentToolIndex + 1 < toolArray.length();
```

```
75          }
76
77          public MeasuringTool next() {
78              MeasuringTool tool = null;
79              try {
80                  tool = createTool(toolArray.getJSONObject(++currentToolIndex));
81                  if (tool == null) {
82                      throw new NoSuchElementException();
83                  }
84              } catch (JSONException e) {
85                  Log.e("ToolConfig", "Error reading json config", e);
86              }
87              return tool;
88          }
89
90          public void remove() {
91              throw new UnsupportedOperationException();
92          }
93      }
94
95      private MeasuringTool createTool(JSONObject toolOptions) {
96
97          MeasuringTool tool = null;
98          try {
99              long repetitions = toolOptions.optLong(
100                     "repetitions", globalOptions.optLong(
101                             toolOptions.optString("repetitions"), this
102                                     .defaultRepetitions));
103
104             int defaultRounds = 1;
105
106             int rounds = toolOptions.optInt(
107                     "rounds", globalOptions.optInt(
108                             toolOptions.optString("rounds"), defaultRounds));
109
110             long allocRepetitions = toolOptions.optLong(
111                     "alloc_repetitions", globalOptions.optLong(
112                             toolOptions.optString("alloc_repetitions"),
113                             this.defaultAllocRepetitions));
114
115             boolean warmup = toolOptions.optBoolean("warmup", false);
116
117             boolean runAllBenchmarks = toolOptions.optBoolean(
118                     "run_all", this.defaultRunAllBenchmarks);
119
120             Class<?> _class = Class.forName(TOOL_PACKAGE + "." + toolOptions
121                     .getString("class"));
122
123             Constructor<?> ctor = _class.getConstructor(
124                     Integer.TYPE, Long.TYPE, Long.TYPE, Boolean.TYPE, Boolean
125                             .TYPE);
126
127             Log.v("ToolConfig", "Tool instantiation " + rounds + " " +
```

```
                    repetitions + " " + warmup);

            try {
                tool = (MeasuringTool) ctor.newInstance(
                        rounds, repetitions, allocRepetitions,
                        warmup, runAllBenchmarks);
            } catch (InvocationTargetException e) {
                Log.e("ToolConfig", "Constructor exception", e.getCause());
            }
            tool.setDescription(toolOptions.optString("description", ""));
            tool.setFilter(toolOptions.optString("filter", ""));
            tool.setExplicitGC(toolOptions.optBoolean("gc", !warmup));

            JSONObject options = toolOptions.optJSONObject("options");
            if (options != null) {
                Iterator keys = options.keys();
                while (keys.hasNext()) {
                    String key = (String) keys.next();
                    tool.set(key, options.getString(key));
                }
            }

        } catch (Exception e) {
            Log.e("ToolConfig", "Error instantiating tool", e);
            return null;
        }
        return tool;
    }

    public ToolConfig setDefaultRepetitions(long r) {
        defaultRepetitions = r;
        return this;
    }

    public ToolConfig setDefaultAllocRepetitions(long r) {
        defaultAllocRepetitions = r;
        return this;
    }

    public ToolConfig setDefaultRunAllBenchmarks(boolean r) {
        defaultRunAllBenchmarks = r;
        return this;
    }

    public BenchmarkRunner.BenchmarkSet getBenchmarkSet() {
        String key = globalOptions.optString("benchmark_set", "non_alloc");
        if (key.equals("alloc")) {
            return BenchmarkRunner.BenchmarkSet.ALLOC;
        }
        return BenchmarkRunner.BenchmarkSet.NON_ALLOC;
    }

    private long defaultRepetitions;
```

```
181     private long defaultAllocRepetitions;
182     private boolean defaultRunAllBenchmarks;
183 
184     private JSONObject globalOptions;
185     private static final String TOOL_PACKAGE = "fi.helsinki.cs.tituomin" +
186             ".nativebenchmark.measuringtool";
187     private static final String TAG = "nativebenchmark.ToolConfig";
188 
189 }
```

Utils.java

```
1  package fi.helsinki.cs.tituomin.nativebenchmark;
2  
3  import java.io.BufferedInputStream;
4  import java.io.BufferedOutputStream;
5  import java.io.File;
6  import java.io.FileInputStream;
7  import java.io.FileNotFoundException;
8  import java.io.FileOutputStream;
9  import java.io.IOException;
10 import java.io.InputStream;
11 import java.io.OutputStream;
12 import java.io.PrintWriter;
13 import java.nio.MappedByteBuffer;
14 import java.nio.channels.FileChannel;
15 import java.nio.charset.Charset;
16 import java.util.UUID;
17 
18 
19 public class Utils {
20 
21     public static String getUUID() {
22         return UUID.randomUUID().toString();
23     }
24 
25     public static String humanTime(long millis) {
26         String time;
27         long seconds = millis / 1000;
28         long minutes = seconds / 60;
29         long hours = minutes / 60;
30         long seconds_total = seconds;
31         seconds %= 60;
32         minutes %= 60;
33         return (hours + "h " +
34                 minutes + "m " +
35                 seconds + "s " +
36                 "(" + seconds_total + " s tot.)");
37     }
```

```
38
39      public static String colFmt(String label, String value) {
40          return String.format("%20s: %s", label, value);
41      }
42
43      public static void colPr(PrintWriter p, String label, Object value) {
44          p.println(colFmt(label, value.toString()));
45      }
46
47      public static void copyStream(InputStream in, OutputStream out) throws
48              IOException {
49          int count;
50          while ((count = in.read(buffer, 0, BUFFERSIZE)) != -1) {
51              out.write(buffer, 0, count);
52          }
53          out.flush();
54      }
55
56      public static PrintWriter
57      makeWriter(File dir, String filename, boolean append)
58              throws FileNotFoundException {
59          return new PrintWriter(makeOutputStream(dir, filename, append));
60      }
61
62      public static PrintWriter
63      makeWriter(File file, boolean append)
64              throws FileNotFoundException {
65          return new PrintWriter(makeOutputStream(file, append));
66      }
67
68      public static OutputStream
69      makeOutputStream(File dir, String filename, boolean append)
70              throws FileNotFoundException {
71          return makeOutputStream(new File(dir, filename), append);
72      }
73
74      public static OutputStream
75      makeOutputStream(File file, boolean append)
76              throws FileNotFoundException {
77          return new BufferedOutputStream(
78                  new FileOutputStream(
79                          file, append));
80      }
81
82      public static InputStream
83      makeInputStream(File file)
84              throws FileNotFoundException {
85          return new BufferedInputStream(new FileInputStream(file));
86      }
87
88
89      public static InputStream
90      makeInputStream(String filename)
```

```
91              throws FileNotFoundException {
92          return makeInputStream(new File(filename));
93      }
94
95      public static String readFileToString(File file) throws IOException {
96          FileInputStream stream = new FileInputStream(file);
97          try {
98              FileChannel fc = stream.getChannel();
99              MappedByteBuffer bb = fc.map(FileChannel.MapMode.READ_ONLY, 0, fc
100                     .size());
101             return Charset.defaultCharset().decode(bb).toString();
102         } finally {
103             stream.close();
104         }
105     }
106
107     private static final int BUFFERSIZE = 128 * 1024;
108     private static byte[] buffer = new byte[BUFFERSIZE];
109 }
```

Python-komponentit (koodingenerointi)

Hakemistossa script/.

benchmark_generator.py

```
from templates import java_benchmark
from templates import java_counterparts
from templates import c_module
from templates import c_jni_function
from templates import c_nativemethod

from templating import put

import itertools
import logging

import jni_types
from jni_types import types, primitive_types, return_types

packagename = [
    'fi',
     'helsinki',
     'cs',
     'tituomin',
     'nativebenchmark',
     'benchmark']
library_name = 'nativebenchmark'
java_counterpart_classname = 'JavaCounterparts'
native_method_name = 'nativemethod'
class_counter = 0

def next_sequence_no():
    global class_counter
    class_counter += 1
    return str(class_counter).zfill(5)


def benchmark_classname(prefix, number):
    return prefix + 'Benchmark' + number


def parameter_initialisation(language, typespec, name):
    if typespec.get('is-array', False):
        if language == 'java':
            expression = 'benchmarkParameter.retrieve{_type}Array()'.format(
                _type=typespec['java-element-type'].capitalize())
        else:
            expression = '{_type}ArrayValue'.format(
                _type=typespec['c-element-type'])
```

```
45
46      elif typespec.get('is-object', False):
47          if language == 'java':
48              expression = 'benchmarkParameter.retrieve{_type}()'.format(
49                  _type=typespec['java'])
50          else:
51              expression = '{_type}Value'.format(
52                  _type=typespec['c'])
53
54      elif language == 'java':
55          if typespec.get('java-literal'):
56              expression = typespec['java-literal']
57          else:
58              expression = ''
59
60      elif language == 'c':
61          if typespec.get('c-literal'):
62              expression = typespec['c-literal']
63          else:
64              expression = ''
65
66      if name:
67          return name + " = " + expression
68      else:
69          return expression
70
71
72  def modifier_combinations():
73      privacy = ['public', 'private', 'protected']
74      static = ['static', '']
75      return list(itertools.product(privacy, static))
76
77
78  def method_combinations():
79      combinations = []
80      combinations.append({
81          'description': 'no arguments or return types',
82              'representative': True,
83              'return_types': [return_types['v']],
84              'target_modifiers': modifier_combinations(),
85              'types': []})
86
87      for symbol, _type in types.iteritems():
88          combinations.append({
89              'description': 'varying count {0}'.format(symbol),
90              'representative': _type.get('representative', False),
91              'return_types': [return_types['v']],
92              'target_modifiers': [('public', '')],
93              'types': jni_types.type_combinations(
94                  size=20,
95                  typeset=[types[symbol]])})
96
97      # Start from 1 to avoid duplicates.
```

```
    combinations.append({
        'description': 'vary number of types',
            'representative': True,
            'skip': 1,
            'return_types': [return_types['v']],
            'target_modifiers': [('public', '')],
            'types': jni_types.type_combinations(
                typeset=types.values())
    })

    combinations.append({
        'description': 'modifier combinations',
            'representative': True,
            'return_types': [return_types['l']],
            'target_modifiers': modifier_combinations(),
            'include_nonvirtual': True,
            'types': [types['i']]
    })

    filtered_return_types = filter(
        lambda x: x['symbol'] != 'l',
        jni_types.type_combinations(typeset=types.values()))

    combinations.append({
        'description': 'return types',
            'return_types': filtered_return_types,
            'target_modifiers': [('public', '')],
            'types': [types['i']]
    })

    return combinations


def generate_benchmarks():
    global class_counter
    java = []
    java_callees = {}
    c_implementations = []
    c_runners = []
    c_methodid_inits = []

    for spec in method_combinations():
        virtualities = [True]
        if spec.get('include_nonvirtual'):
            virtualities.append(False)
        for virtualcall in virtualities:
            for target_modifier in spec['target_modifiers']:
                for return_type in spec['return_types']:

                    if 'representative' not in spec:
                        representative = return_type.get(
                            'representative', False)
                    else:
```

```
                    representative = spec['representative']

                representative = "true" if representative else "false"

                type_combination = spec['types']

                native_method_modifiers = " ".join(target_modifier)
                return_expression = parameter_initialisation(
                    'c', return_type, None)

                # Declare/initialize the parameters that are passed
                # to the called function/method.

                parameter_names = []

                parameter_declarations = []
                parameter_initialisations = []
                c_parameter_declarations = []
                c_parameter_initialisations = []

                if target_modifier[1] == 'static':
                    c_parameter_declarations.append('jclass cls')
                else:
                    c_parameter_declarations.append('jobject instance')

                for i, type_data in enumerate(type_combination):
                    parameter_names.append(
                        type_data['symbol'] + str(i + 1))

                    parameter_declarations.append(
                        type_data['java'] + ' ' + parameter_names[-1])
                    c_parameter_declarations.append(
                        type_data['c'] + ' ' + parameter_names[-1])
                    parameter_initialisations.append(
                        parameter_initialisation(
                            'java', type_data, parameter_names[-1])){
                    c_parameter_initialisations.append(
                        parameter_initialisation(
                            'c', type_data, parameter_names[-1]))

                skip = spec.get('skip', 0)
                upper_bound = len(type_combination){
                if upper_bound == 0:
                    upper_bound = 1
                for i in range(skip, upper_bound):

                    sequence_no = next_sequence_no()

                    for from_lang, to_lang in [
                            ('J', 'C'), ('J', 'J'),
                            ('C', 'C'), ('C', 'J')]:

                        pairing = (from_lang, to_lang)
```

```
204
205                     if virtualcall == False:
206                         # Nonvirtual calls only apply to C -> J
207                         # instance methods J2C is generated as a proxy
208                         # to get benchmark metadata
209                         if (target_modifier[1] == 'static' or
210                                 pairing in [('J', 'J'), ('C', 'C')]):
211                                 continue
212
213                     # 1. Set up call targets.
214
215                     if to_lang == 'C':
216                         counterpart_method_name = 'nativemethod'
217                     if to_lang == 'J':
218                         counterpart_method_unqualified = (
219                             'benchmark' + sequence_no)
220
221                     if pairing == ('J', 'J'):
222                         if target_modifier[0] == 'private':
223                             # A private method cannot
224                             # be called from Java.
225                             # Don't generate a benchmark.
226                             continue
227
228                         if target_modifier[1] == 'static':
229                             clsname = java_counterpart_classname
230                         else:
231                             clsname = 'counterpartInstance'
232                         counterpart_method_name = (
233                             clsname + '.' +
234                             counterpart_method_unqualified)
235                         native_method = ''
236
237                     if pairing == ('C', 'J'):
238                         counterpart_method_name = 'benchmark' + \
239                             sequence_no
240
241                     # 2. Set up calling sources.
242
243                     if from_lang == 'C':
244                         run_method = java_benchmark.native_run_method_t
245                         native_method = ''
246
247                     if from_lang == 'J':
248                         run_method = put(
249                             java_benchmark.java_run_method_t,
250                             parameter_declarations="; ".join(
251                                 parameter_declarations[0:i + 1]),
252                             parameter_initialisations="; ".join(
253                                 parameter_initialisations[0:i + 1]),
254                             counterpart_method_name=(
255                                 counterpart_method_name),
256                             counterpart_method_arguments=(
```

```
                ", ".join(parameter_names[0:i + 1])))

        if pairing == ('J', 'C'):
            native_method = put(
                java_benchmark.native_method_t,
                modifiers=native_method_modifiers,

                return_type=return_type[
                    'java'],
                name=native_method_name,
                parameters=", ".join(
                    parameter_declarations[0:i + 1]))

        # 3. Common benchmark class wrapper for all
        # benchmarks.

        classname = benchmark_classname(
            '2'.join(pairing), sequence_no)

        nm = native_method if to_lang == 'C' else ''
        java.append({
            'filename': classname + ".java",
                'class':
                    '.'.join(
                        packagename) + "." + classname,
                'path': '/'.join(packagename),
                'code': put(
                    java_benchmark.t,
                    representative=representative,
                    imports='',
                    has_dynamic_parameters='false',
                    is_allocating='false',
                    is_nonvirtual=(
                        'false' if virtualcall else 'true'),
                    _id=benchmark_classname(
                        "", sequence_no),
                    description=spec[
                        'description'],
                    seq_no=class_counter,
                    from_language=from_lang,
                    to_language=to_lang,
                    class_relations='',
                    packagename='.'.join(
                    packagename),
                    classname=classname,
                    library_name=library_name,
                    run_method=run_method,
                    native_method=nm})

        if (return_type.get('is-object') or
            return_type.get('is-array')):
            ret_expression = "{_type}Value".format(
                _type=return_type['c'])
```

```
                    else:
                        ret_expression = parameter_initialisation(
                            'java', return_type, None)

                    # 4. Call target implementations.

                    if to_lang == 'J':
                        cmu = counterpart_method_unqualified
                        java_callees[cmu] = put(
                            java_counterparts.counterpart_t,
                                return_type=return_type[
                                    'java'],
                                privacy=target_modifier[0],
                                static=target_modifier[1],
                                methodname=cmu,
                                parameters=", ".join(
                                    parameter_declarations[0:i + 1]),
                                return_expression=ret_expression
                        )

                    if pairing == ('J', 'C'):
                        c_implementations.append(
                            put(c_nativemethod.t,
                                return_type=return_type['c'],
                                packagename=(
                                '_').join(
                                packagename),
                                classname=classname,
                                function=native_method_name,
                                parameters=", ".join(
                                c_parameter_declarations[0:i + 2]),
                                return_expression=return_expression))

                    if pairing == ('C', 'C'):
                        jni_param_names = ['env', 'instance']
                        jni_param_names.extend(parameter_names)

                        c_runners.append(
                            put(c_nativemethod.t_caller_native,
                                packagename='_'.join(packagename),
                                classname=classname,
                                parameter_declarations="; ".join(
                                    c_parameter_declarations[1:i + 2]),
                                parameter_initialisations="; ".join(
                                    c_parameter_initialisations[
                                        0:i + 1]),
                                counterpart_method_name=(
                                    "Java_{}_{}_{}".format(
                                        ('_').join(packagename),
                                        ('J2CBenchmark' +
                                         str(sequence_no)),
                                        native_method_name),
                                counterpart_method_arguments=", ".join(
```

```
                                            jni_param_names[0:i + 3])))

                            if pairing == ('C', 'J'):
                                if return_type.get('is-object',
                                                   return_type.get(
                                                       'is-array', False)):
                                    java_method_type = 'Object'
                                else:
                                    java_method_type = return_type[
                                        'java'].capitalize()

                                arguments = ', '.join(parameter_names[0:i + 1])
                                if arguments != '':
                                    arguments = ', ' + arguments

                                c_runners.append(
                                    c_nativemethod.call_java_from_c(
                                        static=(
                                            target_modifier[1] == 'static'),
                                        nonvirtual=not virtualcall,
                                        seq_no=class_counter,
                                        packagename='_'.join(packagename),
                                        classname=classname,
                                        parameter_declarations="; ".join(
                                            c_parameter_declarations[1:i + 2]),
                                        parameter_initialisations="; ".join(
                                            c_parameter_initialisations[
                                                0:i + 1]),
                                        java_method_type=java_method_type,
                                        call_variant='',
                                        arguments=arguments))

                                c_methodid_inits.append(
                                    put(c_module.mid_init_t,
                                        seq_no=class_counter,
                                        static=target_modifier[
                                            1].capitalize(),
                                        method_name=counterpart_method_name,
                                        method_descriptor=(
                                            jni_types.method_descriptor(
                                                return_type,
                                                type_combination[0:i + 1]))))

    ref_types = jni_types.object_types.values(
    ) + jni_types.array_types.values()
    jcp_decl = ''
    jcp_init = ''
    for _type in ref_types:
        jcp_decl += "private static {_type1} {_type2}Value;\n".format(
            _type1=_type['java'], _type2=_type['c'])
        jcp_init += parameter_initialisation(
            'java', _type, _type['c'] + 'Value') + ";\n"

```

```
    java_counterparts_class = {
        'filename': java_counterpart_classname + ".java",
            'class': '.'.join(
                packagename) +
     "." +
     java_counterpart_classname,
            'path': '/'.join(packagename),
            'code': put(java_counterparts.t,
                        packagename='.'.join(packagename),
                        imports='',
                        return_value_declarations=jcp_decl,
                        return_value_inits=jcp_init,
                        counterpart_methods=''.join(java_callees.values()))}

    c_file = put(
        c_module.t,
        jni_function_templates=''.join(c_implementations),
        initialisers='')

    c_runners_file = put(
        c_module.t,
        jni_function_templates=''.join(c_runners),
        initialisers=put(
            c_module.initialisers_t,
            mid_inits=''.join(c_methodid_inits),
            amount_of_methods=class_counter))

    return {'java': java,
            'java_counterparts': java_counterparts_class,
            'c': c_file,
            'c_runners': c_runners_file}
```

jni_types.py

```
import itertools

primitive_types = None
object_types = None
other_types = None
types = None
return_types = None

representative_types = None

array_types = None

primitive_type_definitions = [
    {
        'symbol': 'b',
```

```
        'java': 'boolean',
        'c': 'jboolean',
        'c-literal': '1',
        'java-literal': 'true',
        'jvm-desc': 'Z',
        'byte_count': '1'
    },

    {
        'symbol': 'y',
        'java': 'byte',
        'c': 'jbyte',
        'c-literal': "'a'",
        'java-literal': '(byte)100',
        'jvm-desc': 'B',
        'byte_count': '1',
        'representative': True,
    },

    {
        'symbol': 'c',
        'java': 'char',
        'c': 'jchar',
        'c-literal': '12',
        'java-literal': "'\u0012'",
        'jvm-desc': 'C',
        'byte_count': '2'
    },

    {
        'symbol': 's',
        'java': 'short',
        'c': 'jshort',
        'c-literal': '101',
        'java-literal': '(short)101',
        'jvm-desc': 'S',
        'byte_count': '2'
    },

    {
        'symbol': 'i',
        'java': 'int',
        'c': 'jint',
        'c-literal': '102',
        'java-literal': '102',
        'jvm-desc': 'I',
        'representative': True,
        'byte_count': '4'
    },

    {
        'symbol': 'l',
        'java': 'long',
```

```
        'c': 'jlong',
        'c-literal': '103',
        'java-literal': '103',
        'jvm-desc': 'J',
        'representative': True,
        'byte_count': '8'
    },

    {
        'symbol': 'f',
        'java': 'float',
        'c': 'jfloat',
        'c-literal': '104.1',
        'java-literal': '104.1f',
        'jvm-desc': 'F',
        'representative': True,
        'byte_count': '4'
    },

    {
        'symbol': 'd',
        'java': 'double',
        'c': 'jdouble',
        'c-literal': '105.1',
        'java-literal': '105.1',
        'jvm-desc': 'D',
        'representative': True,
        'byte_count': '8'
    },
]

object_type_definitions = [

    {
        'symbol': 'O',
        'java': 'Object',
        'package': 'java.lang',
        'c': 'jobject',
        'c-literal': None,
        'java-literal': None,
        'is-object': True,
        'representative': True,
        'jvm-desc': 'Ljava/lang/Object;'
    },

    {
        'symbol': 'C',
        'java': 'Class',
        'package': 'java.lang',
        'c': 'jclass',
        'c-literal': None,
        'java-literal': None,
        'is-object': True,
```

```
122         'jvm-desc': 'Ljava/lang/Class;'
123     },
124 
125     {
126         'symbol': 'S',
127         'java': 'String',
128         'package': 'java.lang',
129         'c': 'jstring',
130         'c-literal': None,
131         'java-literal': '"a string"',
132         'is-object': True,
133         'jvm-desc': 'Ljava/lang/String;'
134     },
135 
136     {
137         'symbol': 'T',
138         'java': 'Throwable',
139         'package': 'java.lang',
140         'c': 'jthrowable',
141         'c-literal': None,
142         'java-literal': None,
143         'is-object': True,
144         'jvm-desc': 'Ljava/lang/Throwable;'
145     }
146 
147 ]
148 
149 other_type_definitions = [
150 
151     {
152         'symbol': 'v',
153         'java': 'void',
154         'c': 'void',
155         'c-literal': None,
156         'java-literal': None,
157         'representative': True,
158         'jvm-desc': 'V'
159     }
160 ]
161 
162 
163 def java_native_methodname(is_static, returntype, parametertypes):
164     ret = "_"
165     if is_static:
166         ret += "st_"
167     ret += types.get(returntype)['java'] + "_"
168     for t in parametertypes:
169         ret += types.get(t)
170 
171 
172 def java_native_methodsignature(is_static, returntype, parametertypes):
173     ret = "private"
174     if is_static:
```

```
        ret += "static "

def type_combinations(size=0, typeset=None):
    if size == 0:
        size = len(typeset)

    return list(itertools.islice(itertools.cycle(typeset), 0, size))


def method_descriptor(return_type, parameter_types):
    return "({parameters}){returndesc}".format(
        parameters=''.join([td['jvm-desc'] for td in parameter_types]),
        returndesc=return_type['jvm-desc'])


def init_types():
    global primitive_types, object_types, other_types, types
    global return_types, array_types, representative_types

    primitive_types = dict([(typedef['symbol'], typedef)
                            for typedef in primitive_type_definitions])
    object_types = dict([(typedef['symbol'], typedef)
                         for typedef in object_type_definitions])
    other_types = dict([(typedef['symbol'], typedef)
                        for typedef in other_type_definitions])

    array_element_types = {}
    array_element_types.update(primitive_types)
    array_element_types['O'] = object_types['O']

    array_types = dict(
        [
            ('A' + key,
             {'symbol': 'A' + key,
              'java': tipe['java'] + '[]',
              'package': tipe.get('package', None),
              'c': tipe['c'] + 'Array',
              'c-literal': None,
              'java-literal': None,
              'is-array': True,
              'representative': tipe.get('representative', False),
              'java-element-type': tipe['java'],
              'c-element-type': tipe['c'],
              'jvm-desc': '[' + tipe['jvm-desc']
             })
            for key, tipe in array_element_types.iteritems()])

    types = dict()
    types.update(primitive_types)
    types.update(object_types)
    types.update(array_types)

    return_types = dict()
```

```
228     return_types.update(types)
229     return_types.update(other_types)
230 
231 #     types.update(other_types)
232 
233 init_types()
```

make_benchmarks.py

```
1  from benchmark_generator import generate_benchmarks, packagename
2  from make_custom_benchmarks import write_custom_benchmarks
3  from templates import java_registry_init
4  from templating import put
5  
6  import sys
7  from sys import argv
8  import os.path
9  import os
10 import logging
11 
12 # Log everything, and send it to stderr.
13 logging.basicConfig(level=logging.DEBUG)
14 
15 
16 def write_benchmark(benchmark, java_output_dir):
17         java_output_path = os.path.join(
18             java_output_dir,
19             benchmark["path"])
20 
21         try:
22             os.makedirs(java_output_path)
23         except OSError:
24             pass
25 
26         java_output = open(
27             os.path.join(
28                 java_output_path,
29                 benchmark["filename"]), 'w')
30 
31         java_output.write(benchmark["code"])
32 
33 
34 def write_benchmarks(c_output, c_runners_output, java_output_dir):
35     benchmarks = generate_benchmarks()
36 
37     c_output.write(benchmarks['c'])
38     c_runners_output.write(benchmarks['c_runners'])
39 
40     write_benchmark(benchmarks['java_counterparts'], java_output_dir)
```

```
41      for benchmark in benchmarks['java']:
42          write_benchmark(benchmark, java_output_dir)
43
44      return [benchmark['class'] for benchmark in benchmarks['java']]
45
46
47  def write_benchmark_initialiser(classes):
48      benchmark_inits = []
49
50      for _class in classes:
51          benchmark_inits.append(java_registry_init.inits(_class))
52
53      path = os.path.join(
54          java_output_dir,
55          'fi/helsinki/cs/tituomin/nativebenchmark',
56          'BenchmarkInitialiser.java')
57
58      init_output = open(path, 'w')
59      init_output.write(
60          put(java_registry_init.t,
61              register_benchmarks="\n".join(benchmark_inits)))
62
63
64  if __name__ == "__main__":
65      try:
66          argv.pop(0)
67          c_output_name = argv.pop(0)
68          c_run_output_name = argv.pop(0)
69          c_custom_output_name = argv.pop(0)
70          java_output_dir = argv.pop(0)
71          c_definition_filename = argv.pop(0)
72          java_definition_filename = argv.pop(0)
73
74          definition_files = {
75              'C': open(c_definition_filename),
76                  'J': open(java_definition_filename)}
77
78          c_run_output = open(c_run_output_name, 'w')
79          c_output = open(c_output_name, 'w')
80
81          classes = (write_benchmarks(c_output, c_run_output, java_output_dir) +
82                     write_custom_benchmarks(
83                             definition_files,
84                             c_custom_output_name,
85                             java_output_dir))
86
87          write_benchmark_initialiser(classes)
88          print(",".join(classes))
89      except Exception as e:
90          logging.exception("Exception was thrown.")
91          sys.exit(1)
92      else:
93          sys.exit(0)
```

make_custom_benchmarks.py

```
import re
import logging
from os import path
from sys import argv
import sys

from templating import put

from templates import arrays
from templates import loop_code
from templates import c_nativemethod
from templates import java_benchmark
from templates import java_registry_init

import jni_types

# Log everything, and send it to stderr.
logging.basicConfig(level=logging.DEBUG)

MAX_ALLOC_REPETITIONS = 500

i = re.IGNORECASE
begin_re = re.compile('\s*//\s*@begin\s*', flags=i)
end_re = re.compile('\s*//\s*@end\s*', flags=i)
inits_re = re.compile('\s*//\s*@inits-end\s*', flags=i)
benchmark_re = re.compile('\s*//\s*@(\S+)\s*')


def inits_block_end(line):
    return inits_re.match(line)


def begins_block(line):
    return begin_re.match(line)


def ends_block(line):
    return end_re.match(line)


def is_benchmark_header(line):
    return benchmark_re.match(line)


def parse_benchmark_header(line):
    tokens = line.split()[1:]
    b_properties = parse_properties(tokens[1:])
    b_properties['id'] = tokens[0][1:]
    return b_properties

```

```
51
52 def parse_properties(seq):
53     kvs = []
54     for s in seq:
55         splitted = s.split('=')
56         kvs.append((splitted[0], splitted[1]))
57     try:
58         return dict(kvs)
59     except ValueError as e:
60         print seq
61         print seq[0].split('=')
62         exit(1)
63
64
65 def abort_if_last(line):
66     if line == '':
67         logging.error("Invalid benchmark input file.")
68         exit(1)
69
70
71 def read_until(f, predicate, collect=None):
72     line = ''
73     while not predicate(line):
74         if collect is not None:
75             collect.append(line)
76         line = f.readline()
77         abort_if_last(line)
78     abort_if_last(line)
79     return line
80
81
82 def read_benchmarks(definition_files):
83     benchmarks = {}
84     for lang, f in definition_files.iteritems():
85         benchmarks[lang] = {'module': None, 'benchmarks': []}
86
87         module_start = []
88         inits = []
89         read_until(f, begins_block, collect=module_start)
90         read_until(f, inits_block_end, collect=inits)
91         benchmarks[lang]['inits'] = ''.join(inits)
92         benchmarks[lang]['module'] = ''.join(module_start)
93         line = read_until(f, is_benchmark_header)
94
95         while line != '':
96             bm_props = parse_benchmark_header(line)
97
98             bm_code = []
99             line = read_until(f,
100                 lambda x: ends_block(x) or is_benchmark_header(x),
101                 collect=bm_code)
102
103             bm_props['code'] = ''.join(bm_code)
```

```
104             benchmarks[lang]['benchmarks'].append(bm_props)
105
106             if ends_block(line):
107                 break
108
109     add_field_and_array_benchmarks(benchmarks)
110     add_overhead_benchmarks(benchmarks)
111     return benchmarks
112
113 OVERHEAD_STEP = 2
114 OVERHEAD_STEPS = 11  # incl. zero
115 OVERHEAD_CODE_STATEMENT = "__a = (((__a * __a * __a) / __b) + __b) / __a;\n"
116
117
118 def add_overhead_benchmark(benchmarks, i, prefix, alloc):
119     overhead_code = []
120     for j in range(0, i):
121         overhead_code.append(OVERHEAD_CODE_STATEMENT)
122
123     benchmark = {
124         'code': ''.join(overhead_code),
125         'id': prefix + 'Overhead' + str(i).zfill(5),
126         'description': i
127     }
128
129     if alloc:
130         benchmark['alloc'] = True
131
132     c_b = benchmark.copy()
133     double_benchmark = benchmark.copy()
134     # double the amount of work for java (uses optimizations unlike c)
135     double_benchmark['code'] = ''.join(overhead_code + overhead_code)
136     j_b = double_benchmark
137     c_b['direction'] = 'cj'
138     j_b['direction'] = 'jj'
139     benchmarks['C']['benchmarks'].append(c_b)
140     benchmarks['J']['benchmarks'].append(j_b)
141
142
143 def add_overhead_benchmarks(benchmarks):
144     for i in range(1, OVERHEAD_STEPS * OVERHEAD_STEP, OVERHEAD_STEP):
145         for prefix, alloc in [('Alloc', True), ('Normal', False)]:
146             add_overhead_benchmark(benchmarks, i, prefix, alloc)
147     add_overhead_benchmark(benchmarks, 200, 'Warmup', False)
148
149
150 def macro_call(template, _type):
151     return template.format(
152         _type=_type['c'],
153         java_type_name=_type['java'].capitalize())
154
155
156 def make_id(template, _type):
```

```
157     return template.format(
158         _type=_type['java'].capitalize())
159
160
161 def add_field_and_array_benchmarks(benchmarks):
162     c = benchmarks['C']['benchmarks']
163     java = benchmarks['J']['benchmarks']
164
165     for _type in (
166             jni_types.primitive_types.values() +
167             [jni_types.object_types['O']]):
168         representative = _type.get('representative', False)
169
170         c.append({
171                 'id': make_id('GetStatic{_type}Field', _type),
172                 'representative': representative,
173                 'direction': 'cj',
174                 'code': macro_call(
175                     'GET_STATIC_TYPE_FIELD({_type}, {java_type_name});',
176                     _type)})
177
178         java.append({
179                 'id': make_id('GetStatic{_type}Field', _type),
180                 'representative': representative,
181                 'direction': 'jj',
182                 'class_init':
183                     'public {} persistentValue;'.format(_type['java']),
184                 'method_init': '{} localPersistentValue = {};'.format(
185                     _type['java'], _type.get('java-literal') or 'objectValue'),
186                 'code':
187                     ("localPersistentValue = "
188                      "mockObject.{_ctype}StaticField;"
189                     ).format(
190                         _javatype=_type['java'],
191                         _ctype=_type['c']
192                     ),
193                 'finished': 'persistentValue = localPersistentValue;'
194                 })
195
196         c.append({
197                 'direction': 'cj',
198                 'representative': representative,
199                 'id': make_id('SetStatic{_type}Field', _type),
200                 'code': macro_call(
201                     'SET_STATIC_TYPE_FIELD({_type}, {java_type_name});',
202                     _type)})
203
204         java.append({
205                 'id': make_id('SetStatic{_type}Field', _type),
206                 'representative': representative,
207                 'direction': 'jj',
208                 'code':
209                     "mockObject.{_ctype}StaticField = {_literal} ;".format(
```

```
                    _ctype=_type['c'],
                    _literal=_type.get('java-literal') or 'objectValue'

                ),
            })

    c.append({
            'id': make_id('Get{_type}Field', _type),
            'direction': 'cj',
            'representative': representative,
            'code': macro_call(
                'GET_TYPE_FIELD({_type}, {java_type_name});',
                _type)})

    java.append({
            'id': make_id('Get{_type}Field', _type),
            'representative': representative,
            'class_init':
                'public {} persistentValue;'.format(_type['java']),
            'method_init': '{} localPersistentValue = {};'.format(
                _type['java'], _type.get('java-literal') or 'objectValue'),
            'direction': 'jj',
            'code':
                "localPersistentValue = mockObject.{_ctype}Field;".format(
                    _javatype=_type['java'],
                    _ctype=_type['c']

                ),
            'finished': 'persistentValue = localPersistentValue;'
            })

    c.append({
            'id': make_id('Set{_type}Field', _type),
            'direction': 'cj',
            'representative': representative,
            'code': macro_call(
                'SET_TYPE_FIELD({_type}, {java_type_name});',
                _type)})

    java.append({
            'id': make_id('Set{_type}Field', _type),
            'representative': representative,
            'direction': 'jj',
            'code':
                "mockObject.{_ctype}Field = {_literal} ;".format(
                    _ctype=_type['c'],
                    _literal=_type.get('java-literal') or 'objectValue'

                ),
            })

for _type in jni_types.primitive_types.values():
    representative = _type.get('representative', False)
```

```
263         c.append({
264                 'id': make_id('New{_type}Array', _type),
265                 'representative': representative,
266                 'direction': 'cj',
267                 'vary': 'size',
268                 'alloc': 'true',
269                 'code': macro_call(
270                     'NEW_PRIMITIVE_ARRAY({_type}, {java_type_name});',
271                     _type)
272                 })
273
274         # java
275
276         c.append({
277                 'id': make_id('Get{_type}ArrayElements', _type),
278                 'representative': representative,
279                 'direction': 'cj',
280                 'vary': 'size',
281                 'code': macro_call(
282                     ('GET_PRIMITIVE_ARRAY_ELEMENTS({_type}, {java_type_name});'
283                     'RELEASE_PRIMITIVE_ARRAY_ELEMENTS'
284                      '({_type}, {java_type_name});'),
285                     _type)})
286
287         c.append({
288                 'vary': 'size',
289                 'direction': 'cj',
290                 'representative': representative,
291                 'id': make_id('Get{_type}ArrayRegion', _type),
292                 'code': macro_call(
293                     'GET_PRIMITIVE_ARRAY_REGION({_type}, {java_type_name});',
294                     _type)})
295
296         c.append({
297                 'vary': 'size',
298                 'representative': representative,
299                 'direction': 'cj',
300                 'id': make_id('Set{_type}ArrayRegion', _type),
301                 'code': macro_call(
302                     'SET_PRIMITIVE_ARRAY_REGION({_type}, {java_type_name});',
303                     _type)})
304
305         c.append({
306                 'vary': 'size',
307                 'representative': representative,
308                 'direction': 'cj',
309                 'id': make_id('Get{_type}ArrayLength', _type),
310                 'code': macro_call(
311                     'GET_PRIMITIVE_ARRAY_LENGTH({_type});',
312                     _type)})
313
314         c.append({
315                 'vary': 'size',
```

```
            'representative': representative,
            'direction': 'cc',
            'id': make_id('ReadComplete{_type}Array', _type),
            'code': put(
                arrays.t_read,
                declare_idx='jint idx;',
                variable_in='%s__IN' % _type['c'],
                array_variable='%s_buf__IN' % _type['c'],
                element_literal=_type['c-literal']
                )})

    java_declarations = ('{0} {0}In;\n{0} [] {0}Arr = '
                         'benchmarkParameter.retrieve{1}Array();').format(
        _type['java'], _type['java'].capitalize())

    java.append({
            'vary': 'size',
            'direction': 'jj',
            'representative': representative,
            'class_init': 'public int persistentValue;',
            'method_init': 'int localPersistentValue = 0;',
            'id': make_id('ReadComplete{_type}Array', _type),
            'code': put(
                arrays.t_read,
                declare_idx='int idx;',
                declare_variables=java_declarations,
                variable_in='%sIn' % _type['java'],
                array_variable='%sArr' % _type['java'],
                element_literal=_type['java-literal']
                ),
            'finished': 'persistentValue = localPersistentValue;'
            })

    c.append({
            'vary': 'size',
            'direction': 'cc',
            'representative': representative,
            'id': make_id('WriteComplete{_type}Array', _type),
            'code': put(
                arrays.t_write,
                declare_idx='jint idx;',
                array_variable='%s_buf__IN' % _type['c'],
                element_literal=_type['c-literal']
                )})

    java.append({
            'vary': 'size',
            'direction': 'jj',
            'representative': representative,
            'class_init': 'public int persistentValue;',
            'method_init': 'int localPersistentValue = 0;',
            'id': make_id('WriteComplete{_type}Array', _type),
            'code': put(
```

```
                arrays.t_write,
                declare_variables=java_declarations,
                declare_idx='int idx;',
                # todo: writing affects other tests?
                array_variable='%sArr' % _type['java'],
                element_literal=_type['java-literal']),
            'finished': 'persistentValue = localPersistentValue;'
            })

    # NIO variations of array/buffer reading/writing
    if _type['java'] == 'boolean':
        # Not available for booleans
        continue

    if _type['java'] == 'byte':
        uppercase_typename = ''
    else:
        uppercase_typename = _type['java'].title()

    # Read with hardcoded type method
    java.append({
        'vary': 'size',
        'direction': 'jj',
        'representative': True,
        'class_init': 'public int persistentValue;',
        'method_init': put(
            arrays.t_init_nio,
            type_declarations=java_declarations),
        'id': make_id('ReadComplete{_type}NioByteBuffer', _type),
        'code': put(
            arrays.t_read_nio,
            declare_idx='int idx;',
            variable_in='%sIn' % _type['java'],
            array_variable='directByteBufferValue',
            type_name=uppercase_typename,
            element_literal=_type['java-literal']),
        'finished': 'persistentValue = localPersistentValue;'
    })
    # Write with hardcoded type method
    java.append({
        'vary': 'size',
        'direction': 'jj',
        'representative': True,
        'class_init': 'public int persistentValue;',
        'method_init': put(
            arrays.t_init_nio,
            type_declarations=java_declarations),
        'id': make_id('WriteComplete{_type}NioByteBuffer', _type),
        'code': put(
            arrays.t_write_nio,
            declare_variables='',
            declare_idx='int idx;',
            array_variable='directByteBufferValue',
```

```
            type_name=uppercase_typename,
            element_literal=_type['java-literal']),
        'finished': 'persistentValue = localPersistentValue;'
    })

    declaration = java_declarations + "\n"
    if _type['java'] == 'byte':
        array_variable = 'directByteBufferValue'
    else:
        # Views are only relevent for non-byte types.
        declaration += ('java.nio.{0}Buffer bufferView = '
                        'directByteBufferValue.as{0}Buffer();').format(
            uppercase_typename)
        array_variable = 'bufferView'

    # Bulk read through a typecast view buffer
    java.append({
        'vary': 'size',
        'direction': 'jj',
        'representative': True,
        'class_init': 'public int persistentValue;',
        'method_init': put(
            arrays.t_init_nio,
            type_declarations=declaration),
        'id': make_id('ReadBulk{_type}NioByteBufferView', _type),
        'code': put(
            arrays.t_bulk_read,
            array_variable=array_variable,
            array_in='%sArr' % _type['java']),
        'finished': 'persistentValue = localPersistentValue;'
    })

    # Bulk write through a typecast view buffer
    java.append({
        'vary': 'size',
        'direction': 'jj',
        'representative': True,
        'class_init': 'public int persistentValue;',
        'method_init': put(
            arrays.t_init_nio,
            type_declarations=declaration),
        'id': make_id('WriteBulk{_type}NioByteBufferView', _type),
        'code': put(
            arrays.t_bulk_write,
            array_variable=array_variable,
            array_in='%sArr' % _type['java']),
        'finished': 'persistentValue = localPersistentValue;'
    })

    if _type['java'] == 'byte':
        continue

    # Read through a typecast view buffer
```

```
        java.append({
            'vary': 'size',
            'direction': 'jj',
            'representative': True,
            'class_init': 'public int persistentValue;',
            'method_init': put(
                arrays.t_init_nio,
                type_declarations=declaration),
            'id': make_id('ReadComplete{_type}NioByteBufferView', _type),
            'code': put(
                arrays.t_read_nio_as_view,
                declare_idx='int idx;',
                variable_in='%sIn' % _type['java'],
                array_variable='bufferView',
                type_name=uppercase_typename,
                element_literal=_type['java-literal']),
            'finished': 'persistentValue = localPersistentValue;'
        })
        # Write through a typecast view buffer
        java.append({
            'vary': 'size',
            'direction': 'jj',
            'representative': True,
            'class_init': 'public int persistentValue;',
            'method_init': put(
                arrays.t_init_nio,
                type_declarations=declaration),
            'id': make_id('WriteComplete{_type}NioByteBufferView', _type),
            'code': put(
                arrays.t_write_nio_as_view,
                declare_idx='int idx;',
                array_variable='bufferView',
                element_literal=_type['java-literal']),
            'finished': 'persistentValue = localPersistentValue;'
        })


def write_custom_benchmarks(
    definition_files,
     c_custom_output_name,
     java_output_dir):
    packagename = (
        'fi',
        'helsinki',
     'cs',
     'tituomin',
     'nativebenchmark',
     'benchmark')

    all_benchmarks = read_benchmarks(definition_files)

    out_c = open(c_custom_output_name, 'w')
    out_c.write(all_benchmarks['C']['module'])
```

```
528
529     java_classes = {}  # classname, contents
530
531     for lang, data in all_benchmarks.iteritems():
532         for benchmark in data['benchmarks']:
533
534             direction = benchmark['direction']
535             from_lang, to_lang = direction[0].upper(), direction[1].upper()
536             if from_lang != lang:
537                 logging.error("Invalid language spec.")
538                 exit(1)
539
540             classname = '{0}2{1}'.format(from_lang, to_lang) + benchmark['id']
541             if 'vary' in benchmark:
542                 dyn_par = 'true'
543             else:
544                 dyn_par = 'false'
545             if 'alloc' in benchmark:
546                 # large heap 128/2 = 64 Mb, 128 el 8 byte array...
547                 is_allocating = 'true'
548             else:
549                 is_allocating = 'false'
550
551             representative = benchmark.get('representative', True)
552
553             if representative:
554                 representative = "true"
555             else:
556                 representative = "false"
557
558             if from_lang == 'C':
559                 out_c.write(put(
560                         c_nativemethod.t_run_method,
561                         return_type='void',
562                         parameters='jobject instance',
563                         function='runInternal',
564                         packagename='_'.join(packagename),
565                         classname=classname,
566                         body=put(
567                             loop_code.t_c_jni_call,
568                             debug=classname,
569                             benchmark_body=benchmark['code'])))
570
571                 java_classes[classname] = {
572                     'filename': classname + '.java',
573                     'code': (put(
574                         java_benchmark.t,
575                         representative=representative,
576                         _id=benchmark['id'],
577                         packagename='.'.join(packagename),
578                         classname=classname,
579                         description=benchmark.get('description', ''),
580                         is_allocating=is_allocating,
```

```
                        from_language=from_lang,
                        to_language=to_lang,
                        seq_no='-1',
                        has_dynamic_parameters=dyn_par,
                        is_nonvirtual='false',
                        run_method='public native void runInternal();',
                        ))}

            elif from_lang == 'J' and to_lang == 'J':
                java_classes[classname] = {
                    'filename': classname + '.java',
                    'code': (
                        put(
                            java_benchmark.t,
                            representative=representative,
                            _id=benchmark['id'],
                            packagename='.'.join(packagename),
                            imports="\n".join(
                                ['import android.content.pm.PermissionInfo;',
                                 'import java.nio.ByteBuffer;'
                                 'import java.lang.ref.WeakReference;'
                                 ]),
                            classname=classname,
                            class_fields=benchmark.get('class_init', ''),
                            description=benchmark.get('description' ''),
                            is_allocating=is_allocating,
                            from_language=from_lang,
                            to_language=to_lang,
                            is_nonvirtual='false',
                            seq_no='-1',
                            has_dynamic_parameters=dyn_par,
                            run_method=put(
                                java_benchmark.java_run_method_inline_t,
                                init=data['inits'],
                                type_init=benchmark.get('method_init', ''),
                                loop=put(
                                    loop_code.t_java,
                                    finished=benchmark.get('finished', ''),
                                    benchmark_body=benchmark['code'])))))}

    out_c.flush()
    out_c.close()

    for classname, contents in java_classes.iteritems():
        f = open(
            path.join(java_output_dir,
     '/'.join(packagename),
                contents['filename']),
            'w')
        f.write(contents['code'])
        f.flush()
        f.close()
```

```
634     return ([’.’.join(packagename + (classname,))
635              for classname in java_classes.keys()])
```

templates/arrays.py

```
1  from templating import partial
2
3  t_loop = """
4  <% declare_idx %>
5  <% declare_variables %>
6  for (idx = 0; idx < current_size; idx++) {
7      <% body %>
8  }
9  """
10
11 t_read = partial(
12     t_loop,
13     body="""
14     <% variable_in %> = <% array_variable %>[idx];
15     """)
16
17 t_write = partial(
18     t_loop,
19     body="""
20     <% array_variable %>[idx] = <% element_literal %>; """)
21
22 t_init_nio = """
23     <% type_declarations %>
24     int localPersistentValue = 0;
25     current_size /= 64;
26
27 """
28
29 t_read_nio = partial(
30     t_loop,
31     body="""
32     <% variable_in %> = <% array_variable %>.get<% type_name %>(idx);
33     """)
34
35 t_write_nio = partial(
36     t_loop,
37     body="""
38     <% array_variable %>.put<% type_name %>(idx, <% element_literal %>);
39     """)
40
41 t_read_nio_as_view = partial(
42     t_loop,
43     body="""
44     <% variable_in %> = <% array_variable %>.get(idx);
```

```
    """)

t_write_nio_as_view = partial(
    t_loop,
    body="""
    <% array_variable %>.put(idx, <% element_literal %>);
    """)

t_bulk_read = """
<% declare_variables %>
<% array_variable %>.clear();
<% array_variable %>.get(<% array_in %>, 0, current_size);
"""

t_bulk_write = """
<% declare_variables %>
<% array_variable %>.clear();
<% array_variable %>.put(<% array_in %>, 0, current_size);
"""
```

templates/c_jni_function.py

```
t = """

JNIEXPORT <% return_type %> JNICALL
Java_<% package %>_<% class_name %>_<% function %>
(JNIEnv *env, <% parameters %>) <%
    <% set_returnvalues %>
 %>

"""
```

templates/c_module.py

```
t = """
#include <jni.h>
#include <android/log.h>
#include <stdio.h>
#include "natives.h"
#include "native_benchmarks.h"
#include "returnvalues.h"

<% initialisers %>
<% jni_function_templates %>
```

```
11
12 """
13
14 initialisers_t = """
15 static jmethodID mids[<% amount_of_methods %>];
16
17 static void init_methodids(JNIEnv *env) {
18     jmethodID mid;
19 <% mid_inits %>
20 }
21
22
23 int check_interrupted(JNIEnv *env) {
24     jobject current_thread = NULL;
25     current_thread = (
26         (*env)->CallStaticObjectMethod(env, thread_class, current_thread_mid));
27     if (current_thread == NULL) {
28         __android_log_write(ANDROID_LOG_ERROR, "check_interrupted",
29             "Can't get current thread");
30         return 1;
31     }
32     jboolean interrupted = (*env)->CallBooleanMethod(
33         env, current_thread, is_interrupted_mid);
34     (*env)->DeleteLocalRef(env, current_thread);
35     if (interrupted == JNI_TRUE) {
36         return 1;
37     }
38     return 0;
39 }
40
41 void throw_interrupted_exception(JNIEnv *env) {
42     jclass newExcCls;
43     newExcCls = (*env)->FindClass(env,
44     "java/lang/InterruptedException");
45     if (newExcCls == NULL) {
46         /* Unable to find the exception class, give up. */
47         return;
48     }
49     (*env)->ThrowNew(env, newExcCls, "thrown from C code");
50 }
51
52 JNIEXPORT void JNICALL
53 Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_initNative
54 (JNIEnv *env, jclass cls, jlong reps, jlong interval, jclass javaCounterparts,
55 jobject javaCounterpartsInstance, jclass thread_cls) {
56     repetitions = reps;
57     interrupted = 0;
58
59     CHECK_INTERRUPTED_INTERVAL = interval;
60
61     jclass java_counterparts_class_global_ref = NULL;
62     jclass thread_class_global_ref = NULL;
63     jobject java_counterparts_object_global_ref = NULL;
```

```
64
65      java_counterparts_class_global_ref = (*env)->NewGlobalRef(
66          env, javaCounterparts);
67      if (java_counterparts_class_global_ref == NULL) {
68          __android_log_write(ANDROID_LOG_ERROR, "initNative",
69              "Could not create global ref.");
70          return;
71      }
72      java_counterparts_class = java_counterparts_class_global_ref;
73
74      java_counterparts_object_global_ref = (*env)->NewGlobalRef(env,
75          javaCounterpartsInstance);
76      if (java_counterparts_object_global_ref == NULL) {
77          __android_log_write(ANDROID_LOG_ERROR, "initNative",
78              "Could not create global ref.");
79          return;
80      }
81      java_counterparts_object = java_counterparts_object_global_ref;
82
83      if (!(*env)->IsInstanceOf(env, java_counterparts_object,
84          java_counterparts_class)) {
85          __android_log_write(ANDROID_LOG_ERROR, "initNative",
86              "JavaCounterparts instance or class is not correct.");
87          return;
88      }
89
90      init_methodids(env);
91
92      thread_class_global_ref = (*env)->NewGlobalRef(env, thread_cls);
93      if (thread_class_global_ref == NULL) {
94          __android_log_write(ANDROID_LOG_ERROR, "initNative",
95              "Could not create global ref.");
96          return;
97      }
98      thread_class = thread_class_global_ref;
99
100     current_thread_mid = (*env)->GetStaticMethodID(env, thread_class,
101         "currentThread", "()Ljava/lang/Thread;");
102     if (current_thread_mid == NULL) {
103         __android_log_write(ANDROID_LOG_ERROR, "initNative",
104             "Could not find currentThread");
105         return;
106     }
107     is_interrupted_mid = (*env)->GetMethodID(env, thread_class,
108         "isInterrupted", "()Z");
109     if (is_interrupted_mid == NULL) {
110         __android_log_write(ANDROID_LOG_ERROR, "check_interrupted",
111             "Can't get isInterrupted method");
112         return;
113     }
114
115 }
116
```

```
JNIEXPORT void JNICALL
Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_setRepetitions
(JNIEnv *env, jclass cls, jlong reps) {
    repetitions = reps;
}

JNIEXPORT void JNICALL
Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_interruptNative
(JNIEnv *env, jclass cls) {
    interrupted = 1;
}

JNIEXPORT void JNICALL
Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_resetInterruptFlag
(JNIEnv *env, jclass cls) {
    interrupted = 0;
}

"""

mid_init_t = """
    mid = (*env)->Get<% static %>MethodID(
        env, java_counterparts_class, "<% method_name %>",
        "<% method_descriptor %>");
    if (mid == NULL) {
        __android_log_write(ANDROID_LOG_VERBOSE, "nativemethod",
            "<% method_descriptor %> not found.");
        return; /* method not found */
    }
    mids[<% seq_no %> - 1] = mid;

"""
```

templates/c_nativemethod.py

```
from templating import partial, put
import loop_code

t_run_method = """
JNIEXPORT <% return_type %> JNICALL
Java_<% packagename %>_<% classname %>_<% function %>
(JNIEnv *env, <% parameters %>) {
    <% parameter_declarations %>;
    <% parameter_initialisations %>;
    <% prebody %>
    <% body %>
}

"""
```

```
15
16 t = partial(
17     t_run_method,
18     body='return <% return_expression %>;',
19     remove=['parameter_declarations',
20             'parameter_initialisations',
21             'prebody'])
22
23 # C to C
24 t_caller_native = partial(
25     t_run_method,
26     return_type='void',
27     function='runInternal',
28     parameters='jobject instance',
29     prebody='',
30     body=partial(
31         loop_code.t_c_jni_call,
32         benchmark_body=(
33             '<% counterpart_method_name %>' +
34             '(<% counterpart_method_arguments %>);')))
35
36 # C to J
37 t_caller_java = partial(
38     t_run_method,
39     return_type='void',
40     function='runInternal',
41     parameters='jobject instance',
42     prebody='jmethodID mid = mids[<% seq_no %> - 1];')
43
44
45 def call_java_from_c(static=True, nonvirtual=False, **parameters):
46     benchmark_body = ''
47     if static:
48         benchmark_body = (
49             '(*env)->CallStatic<% java_method_type %>Method<% call_variant %>'
50             '(env, java_counterparts_class, mid<% arguments %>);')
51     elif nonvirtual:
52         benchmark_body = (
53             '(*env)->CallNonvirtual'
54             '<% java_method_type %>Method<% call_variant %>'
55             '(env, java_counterparts_object, java_counterparts_class,'
56             ' mid<% arguments %>);')
57     else:
58         benchmark_body = (
59             '(*env)->Call<% java_method_type %>Method<% call_variant %>' +
60             '(env, java_counterparts_object, mid<% arguments %>);')
61
62     parameters['body'] = put(
63         loop_code.t_c_jni_call,
64         benchmark_body=put(benchmark_body, **parameters))
65
66     return partial(t_caller_java, **parameters)
```

templates/__init__.py

```
1
```

templates/java_benchmark.py

```
from templating import partial
import loop_code
import logging

t = """
package <% packagename %>;

import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkParameter;
import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.BasicOption;
import fi.helsinki.cs.tituomin.nativebenchmark.MockObject;
<% imports %>
import android.util.Log;

public class <% classname %> <% class_relations %> extends Benchmark {

    public <% classname %> (BenchmarkParameter bp) {
        init(bp);
    }

    public String from() {
        return "<% from_language %>";
    }

    public String to() {
        return "<% to_language %>";
    }

    public int sequenceNo() {
        return <% seq_no %>;
    }

    public String id() {
        return "<% _id %>";
    }

    public boolean representative() {
        return <% representative %>;
    }

```

```
42      public boolean dynamicParameters() {
43          return <% has_dynamic_parameters %>;
44      }
45
46      public String description() {
47          return "<% description %>";
48      }
49
50      public boolean isAllocating() {
51          return <% is_allocating %>;
52      }
53
54      public boolean isNonvirtual() {
55          return <% is_nonvirtual %>;
56      }
57
58      <% class_fields %>
59
60      <% native_method %>
61
62      <% run_method %>
63
64  }
65
66  """
67
68  native_method_t = ('<% modifiers %> native <% return_type %> '
69                     '<% name %> (<% parameters %>);')
70  dynamic_parameter_t = (
71      'new BasicOption(BasicOption.VARIABLE, "<% variable %>")'.strip()
72  native_run_method_t = 'public native void runInternal();'
73
74  loop = partial(loop_code.t_java,
75                 benchmark_body=('<% counterpart_method_name %> '
76                                 '(<% counterpart_method_arguments %>);'))
77
78  java_run_method_t   = partial("""
79
80      public void runInternal() {
81          JavaCounterparts counterpartInstance = JavaCounterparts.INSTANCE;
82          <% parameter_declarations %>;
83          <% parameter_initialisations %>;
84
85          <% loop %>
86      }
87
88  """, loop=loop)
89
90  java_run_method_inline_t = """
91
92      public void runInternal() {
93          <% init %>
94          <% type_init %>
```

```
        <% loop %>
    }

"""
```

templates/java_counterparts.py

```
from templating import put

t = """
package <% packagename %>;

<% imports %>
import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkParameter;
import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRunner;
import android.util.Log;


public enum JavaCounterparts {
    INSTANCE;

    <% return_value_declarations %>
    public int persistentValue;
    public static int staticpersistentValue = 0;

    private JavaCounterparts() {
        persistentValue = 0;
    }

    public static void initParams(BenchmarkParameter benchmarkParameter) {
        <% return_value_inits %>
    }

    <% counterpart_methods %>

}

"""

counterpart_t = """

<% privacy %> <% static %> <% return_type %> <% methodname %>(<% parameters %>) {
    <% static %>persistentValue = (<% static %>persistentValue + 1) % 10;
    return <% return_expression %>;
}

"""

return_value_t = (
```

```
    "private static <% actualtype %> = "
    "benchmarkParameter.retrieve<% typename %>(<% typespecs %>);")
```

templates/java_registry_init.py

```
from templating import put

t = """
package fi.helsinki.cs.tituomin.nativebenchmark;

import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkParameter;
import fi.helsinki.cs.tituomin.nativebenchmark.benchmark.*;
import java.util.List;

public class BenchmarkInitialiser {

    public static void init(BenchmarkParameter bp) {
        List<Benchmark> benchmarks = BenchmarkRegistry.getBenchmarks();

        <% register_benchmarks %>
    }

}

"""


def inits(classname):
    return 'benchmarks.add(new {0} (bp));'.format(classname)
```

templates/loop_code.py

```
from templating import put, partial

t = """
    <% declare_counters %>
    <% additional_declaration %>

    <% init_counters %>
    division  = repetitions / interval + 1;
    remainder = repetitions % interval + 1;

```

```
    <% debug %>
    <% additional_init %>
    while (--division != 0) {
        <% init_counters %>
        interval = interval + 1;
        while (--interval != 0) {
            <% pre_body %>
            <% extra_debug %>
            <% benchmark_body %>
            <% post_body %>
        }
        if (<% test_interrupted %>) {
            <% debug_interrupted %>
            return;
        }
    }

    <% additional_init %>

    while (--remainder != 0) {
        <% pre_body %>
        <% benchmark_body %>
        <% post_body %>
    }

    <% removal_prevention %>
    <% finished %>

"""

jni_push_frame = """
if (refs == 0) {
    refs = LOCAL_FRAME_SIZE;
    if ((*env)->PushLocalFrame(env, LOCAL_FRAME_SIZE) < 0) {
        return;
    }
}
"""

jni_pop_frame = """
if (--refs == 0) {
    (*env)->PopLocalFrame(env, NULL);
}

"""


t_c_base = partial(
    t,
    declare_counters='jlong interval, division, remainder;',
    init_counters='interval = CHECK_INTERRUPTED_INTERVAL;',
    test_interrupted='interrupted')

```

```
t_c_jni_call = partial(
    t_c_base,
    additional_declaration='jlong refs;',
    additional_init='refs = 0;',
    remove=[
        'extra_debug',
        'debug',
     'debug_interrupted',
     'removal_prevention'],
    pre_body=jni_push_frame,
    post_body=jni_pop_frame)

t_c = partial(
    t_c_base,
    remove=['extra_debug', 'debug', 'debug_interrupted',
            'additional_declaration', 'additional_init',
            'pre_body', 'post_body', 'removal_prevention'])

t_java = partial(
    t,
    test_interrupted='Thread.currentThread().isInterrupted()',
    extra_debug='',
    declare_counters='long interval, division, remainder;',
    init_counters='interval = BenchmarkRegistry.CHECK_INTERRUPTED_INTERVAL;',
    removal_prevention='repetitionsLeft = division * interval + remainder;',
    remove=['additional_declaration',
            'additional_init',
            'pre_body',
            'post_body'])
```

templating.py

```
import string
import logging
formatter = string.Formatter()


class PartialDict(dict):

    def __missing__(self, key):
        return "<% " + key + " %>"


class PurgeDict(dict):

    def __missing__(self, key):
        return ""


```

```
def escape(string):
    string = string.replace('{', '__BEG__')
    string = string.replace('}', '__END__')
    string = string.replace('<% ', '{')
    string = string.replace(' %>', '}')
    return string


def unescape(string):
    string = string.replace('{', '<% ')
    string = string.replace('}', ' %>')
    string = string.replace('__BEG__', '{')
    string = string.replace('__END__', '}')
    return string


def put(template, remove=None, purge=True, **kwargs):
    try:
        template = escape(template)
        for k, v in kwargs.iteritems():
            if isinstance(v, str):
                kwargs[k] = escape(v)
            if v is None:
                kwargs[k] = ''

        if remove:
            for k in remove:
                kwargs[k] = ''

        if purge:
            fdict = PurgeDict(**kwargs)
        else:
            fdict = PartialDict(**kwargs)

        result = formatter.vformat(template, (), fdict)
        result = unescape(result)
        return result
    except ValueError as e:
        logging.error('error with template: ' + template)
        raise e
    return None


def partial(template, remove=None, **kwargs):
    return put(template, remove=remove, purge=False, **kwargs)
```

Python-komponentit (analyysi)

Hakemistossa bench-analyzer.

/analysis.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from numpy import polyfit, reshape, polyval

def linear_fit_columns(x, y):
    p, residuals, rank, singular_values, rcond = polyfit(x, y, 1, full=True)
    ynorm = normalized(x, y, p)
    pnorm, residuals, rank, singular_values, rcond = polyfit(
        x, ynorm, 1, full=True)
    return p, residuals

def normalized(x, y, poly):
    # normali
    return (x - poly[1]) / poly[0]

def linear_fit(rows):
    columns = reshape(rows, len(rows)*len(rows[0]), order='F').reshape(
        (len(rows[0]), -1))
    x = columns[0]
    columns = columns[1:]
    residuals = [linear_fit_columns(x, col)[1][0] for col in columns]
    polys = [linear_fit_columns(x, col)[0] for col in columns]
    return x, polys, residuals

def estimate_measuring_overhead(rows):
    x, polys, residuals = linear_fit(rows)
    return [p[1] for p in polys]

def optimize_bins(x):
    """
    Created on Thu Oct 25 11:32:47 2012

    Histogram Binwidth Optimization Method

    Shimazaki and Shinomoto, Neural Comput 19 1503-1527, 2007
    2006 Author Hideaki Shimazaki, Matlab
    Department of Physics, Kyoto University
    shimazaki at ton.scphys.kyoto-u.ac.jp
    Please feel free to use/modify this program.

    Version in python adapted Érbet Almeida Costa

    Data: the duration for eruptions of
```

```
    the Old Faithful geyser in Yellowstone National Park (in minutes)
    or normal distribuition.
    Version in python adapted Érbet Almeida Costa
    Bugfix by Takuma Torii 2.24.2013

    """

    import numpy as np
    from numpy import mean, size, zeros, where, transpose
    from numpy.random import normal
    from matplotlib.pyplot import hist
    from scipy import linspace
    import array

    x_max = max(x)
    x_min = min(x)
    N_MIN = 4   #Minimum number of bins (integer)
                #N_MIN must be more than 1 (N_MIN > 1).
    N_MAX = 1000  #Maximum number of bins (integer)
    N = range(N_MIN,N_MAX) # #of Bins
    N = np.array(N)
    D = (x_max-x_min)/N    #Bin size vector
    C = zeros(shape=(size(D),1))

    #Computation of the cost function
    for i in xrange(size(N)):
        edges = linspace(x_min,x_max,N[i]+1) # Bin edges
        ki = hist(x,edges) # Count # of events in bins
        ki = ki[0]
        k = mean(ki) #Mean of event count
        v = sum((ki-k)**2)/N[i] #Variance of event count
        C[i] = (2*k-v)/((D[i])**2) #The cost Function
    #Optimal Bin Size Selection

    cmin = min(C)
    idx  = where(C==cmin)
    idx = int(idx[0])
    optD = D[idx]

    edges = linspace(x_min,x_max,N[idx]+1)

    return optD, edges
```

/datafiles.py

```
#!/usr/bin/python

import re
from collections import OrderedDict as odict
```

```
import sys

SEPARATOR = ','
RE_EMPTY = re.compile('^\s*$')
RE_NUMERICAL = re.compile('^-?[0-9]+$')


def explode(line):
    return line.split(SEPARATOR)

def value(string, key=None):
    if key in ['start', 'end']:
        return string
    if key == 'class':
        return string.split('.')[-1]
    if string == '-' or RE_EMPTY.match(string):
        return None
    if RE_NUMERICAL.match(string):
        return int(string)
    else:
        return string

def empty_label():
    empty_label.cnt += 1
    return 'empty_{0}'.format(empty_label.cnt)

empty_label.cnt = 0

def read_datafiles(files, silent=False):
    if not silent:
        print 'Reading from %s files' % len(files)
    benchmarks = []
    #-1: there is an empty field at the end...

    keys_with_values = set()
    all_keys = set()

    lineno = 1
    for i, f in enumerate(files):
        line = f.readline()
        labels = explode(line)
        for i, l in enumerate(labels):
            # account for the fact that there might be an empty label
            # and corresponding column (usually the last)
            if RE_EMPTY.match(l):
                labels[i] = empty_label()

        all_keys.update(labels)

        line = f.readline()
        while line != '':
            exploded_line = explode(line)
            pad_amount = len(labels) - len(exploded_line)
```

```
58              exploded_line.extend(['-'] * pad_amount)
59              if len(labels) != len(exploded_line):
60                   print ('missing values', f.name, 'line', lineno, 'labels',
61                          len(labels), 'values', len(exploded_line))
62                   exit(1)
63
64              benchmark = dict()
65              benchmark['lineno'] = lineno
66
67              for key, string in zip(labels, exploded_line):
68                  benchmark[key] = value(string, key=key)
69
70                  if value(string, key=key) != None:
71                      keys_with_values.add(key)
72
73              #if benchmark['response_time'] != None:
74              benchmarks.append(benchmark)
75
76              line = f.readline()
77              lineno += 1
78
79      keys_without_values = all_keys - keys_with_values
80
81      benchmark_keycount = None
82      for benchmark in benchmarks:
83          for key in keys_without_values:
84              if key in benchmark:
85                  del benchmark[key]
86          current_keycount = len(benchmark.keys())
87          benchmark_keycount = benchmark_keycount or current_keycount
88          if benchmark_keycount != current_keycount:
89              print ("Benchmarks have different amount of data",
90                     benchmark_keycount, current_keycount, "at line",
91                     benchmark['lineno']
92              exit(1)
93
94      if not silent:
95          print 'Read %d lines' % (lineno - 1)
96      return benchmarks
97
98  def read_measurement_metadata(mfile, combine_compatibles):
99      compatibles = odict()
100     measurement = None
101     line = None
102
103     i = 0
104     while line != '':
105         skipped = False
106         while line == "\n":
107             line = mfile.readline()
108             skipped = True
109
110         if skipped:
```

```
            if measurement:
                if 'tools' in measurement:
                    measurement['tool'] = measurement['tools']
                revision = measurement.get('code-revision')
                checksum = measurement.get('code-checksum')
                repetitions = measurement.get('repetitions')
                tool = measurement.get('tool')
                cpufreq = measurement.get('cpu-freq')
                benchmark_set = measurement.get('benchmark-set')
                substring_filter = measurement.get('substring-filter')
                if measurement.get('rounds') == None:
                    measurement['rounds'] = 1

                if revision and repetitions:
                    if combine_compatibles:
                        key = (revision, checksum, repetitions, tool, cpufreq,
                               benchmark_set, substring_filter)
                    else:
                        key = i
                        i += 1
                    if key not in compatibles:
                        compatibles[key] = []
                    compatibles[key].append(measurement)
            measurement = {}

        if line != None:
            splitted = line.split()
            if len(splitted) > 1:
                key = splitted[0].rstrip(':')
                val = ' '.join(splitted[1:])
                measurement[key] = val.strip()

        line = mfile.readline()

    return compatibles
```

/gnuplot.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
import uuid

INIT_PALETTE = """
# line styles for ColorBrewer Dark2
# for use with qualitative/categorical data
# provides 8 dark colors based on Set2
# compatible with gnuplot >=4.2
```

```
# author: Anna Schneider

# line styles
set style line 1 pt 7 lt 1 lc rgb '#1B9E77' # dark teal
set style line 2 pt 7 lt 1 lc rgb '#D95F02' # dark orange
set style line 3 pt 7 lt 1 lc rgb '#7570B3' # dark lilac
set style line 4 pt 7 lt 1 lc rgb '#000000' # black
set style line 5 pt 7 lt 1 lc rgb '#E7298A' # dark magenta
set style line 6 pt 7 lt 1 lc rgb '#66A61E' # dark lime green
set style line 7 pt 7 lt 1 lc rgb '#E6AB02' # dark banana
set style line 8 pt 7 lt 1 lc rgb '#A6761D' # dark tan
set style line 9 pt 7 lt 1 lc rgb '#666666' # dark gray
set style line 10 pt 7 lt 1 lc rgb '#1b70b3' # dark blue

set style line 11 pt 5 lt 2 lc rgb '#1B9E77' # dark teal
set style line 12 pt 5 lt 2 lc rgb '#D95F02' # dark orange
set style line 13 pt 5 lt 2 lc rgb '#7570B3' # dark lilac
set style line 14 pt 5 lt 2 lc rgb '#000000' # black
set style line 15 pt 5 lt 2 lc rgb '#E7298A' # dark magenta
set style line 16 pt 5 lt 2 lc rgb '#66A61E' # dark lime green
set style line 17 pt 5 lt 2 lc rgb '#E6AB02' # dark banana
set style line 18 pt 5 lt 2 lc rgb '#A6761D' # dark tan
set style line 19 pt 5 lt 2 lc rgb '#666666' # dark gray
set style line 20 pt 5 lt 2 lc rgb '#1b70b3' # dark blue


# palette
set palette maxcolors 8
set palette defined ( 0 '#1B9E77',\
                      1 '#D95F02',\
              2 '#7570B3',\
              3 '#E7298A',\
              4 '#66A61E',\
              5 '#E6AB02',\
              6 '#A6761D',\
              7 '#666666' )
"""

INIT_PLOTS_PDF = """
set terminal pdfcairo size 32cm,18cm {sizesuffix}
set size 1, 0.95
set output '{filename}'
"""

INIT_PLOTS_LATEX = """
set terminal epslatex input color \
header "\\\\caption{{{caption}}}\\\\label{{fig:{label}}}" {sizesuffix}
set pointsize 1.0
set format y "%4.2s%cs"
set output
"""

INIT_PLOTS_SVG = """
```

```
set terminal svg {sizesuffix}
set pointsize 1.0
set format y "%4.2s%cs"
set output
"""

INIT_PLOTS_COMMON = """
set grid
set xlabel "kutsuparametrien määrä"
"""

INIT_PLOT_LABEL_PDF = """
set label 1 "{bid}" at graph 0.01, graph 1.06
"""

TEMPLATES = {}
INIT_KEY = {}

TEMPLATES['binned_init'] = """
set title '{title}
binwidth={binwidth}
set boxwidth binwidth
set style fill solid 1.0
set xrange [{min_x}:{max_x}]
set yrange [0:{max_y}]
"""
# border lt -1
#bin(x,width)=width*floor(x/width) + width/2.0

TEMPLATES['binned_frame'] = """
#set label 2 "{datapoints}" at graph 0.8, graph 1.06
set bmargin 20
set tmargin 20
set rmargin 20
set lmargin 20
plot '-' using 1:2 notitle with boxes lt rgb "{color}"\n{values}\ne\n
#unset xlabel
#unset ylabel
#unset label 1
#unset title
unset xtics
unset ytics
"""

SET_TITLE_AND_PAGE_LABEL = """
set title '{title}'
set label 2 "{page}" at screen 0.9, screen 0.95
"""

INIT_KEY['simple_groups'] = """
set key {key_placement} box notitle width -3 height +1 vertical
"""

```

```
118 TEMPLATES['simple_groups'] = """
119 set ylabel "vasteaika {reps} toistolla"
120 set xlabel "{xlabel}"
121 plot for [I=2:{last_column}] '{filename}' index {index} \
122 using 1:I title columnhead with points ls I-1
123 """
124
125 TEMPLATES['fitted_lines'] = """
126 set ylabel "vasteaika {reps} toistolla"
127 set xlabel "{xlabel}"
128 plot for [I=2:{last_real_column}] '{filename}' index {index} using 1:I \
129 title columnhead with points ls I-1, \
130 for [I={first_fitted_column}:{last_column}] '{filename}' index {index} \
131 using 1:I notitle with lines ls I-{first_fitted_column}+1
132 """
133
134 TEMPLATES['named_columns'] = """
135 set yrange [0:*]
136 set xlabel "{xlabel}"
137 plot for [I=2:{last_column}] '{filename}' index {index} using I:xtic(1) \
138 title columnhead with linespoints
139 """
140
141 TEMPLATES['histogram'] = """
142 unset xlabel
143 unset ylabel
144 set y2label "vasteaika {reps} toistolla"
145 set size 1, 1
146 unset x2tics
147 #unset xtics
148 unset ytics
149
150 set y2tics format "%.00s%cs" rotate
151
152 set xtics out rotate
153 set key at graph 0.1, 0.9 width 2 height 8 notitle horizontal nobox samplen 0.2
154 set label 1 'C$\\rightarrow$Java' at graph 0.145, 0.78 left rotate by 90
155 set label 2 'Java$\\rightarrow$Java' at graph 0.205, 0.78 left rotate by 90
156 set style data histograms
157 set style histogram clustered
158
159 plot [] [0:*] for [I=2:{last_column}] '{filename}' index {index} \
160 using I:xtic(1) every ::1 title " " with histogram fillstyle solid 1.0 border lt -1
161 """
162
163 measurement_id = None
164 plot_directory = '/home/tituomin/gradu/paper/figures/plots'
165
166 def init(plotscript, filename, mid, output_type='pdf'):
167     global measurement_id, plot_directory
168     measurement_id = mid
169     if output_type == 'pdf':
170         plotscript.write(INIT_PLOTS_PDF.format(filename=filename))
```

```
171         plotscript.write(INIT_PLOT_LABEL_PDF.format(bid=measurement_id))
172     plotscript.write(INIT_PLOTS_COMMON)
173     plotscript.write(INIT_PALETTE)
174 
175 GROUPTITLES={
176     'direction': 'kutsusuunta',
177     'from': 'kieli'
178 }
179 
180 def output_plot(data_headers, data_rows, plotpath,
181                 plotscript, title, specs, style, page,
182                 identifier,
183                 xlabel, additional_data=None, output='pdf',
184                 key_placement="inside top left", reps='XXX-fixme-XXX'):
185     global plot_directory
186     template = TEMPLATES[style]
187 
188     rowlen = len(data_rows[0]) - 1
189     size = 'normal'
190     if style == 'fitted_lines':
191         rowlen /= 2
192     if (page > 51 and rowlen > 7) or rowlen > 10:
193         size = 'tall'
194     if rowlen < 15:
195         size = 'normal'
196     if identifier in [
197         'basic-call-all-types-j-j-fit',
198         'basic-call-all-types-c-c-fit',
199         'variable-argument-size-j-c',
200         'special-calls-arrayelements-c-j-fit',
201         'special-calls-arrayregion-c-j-fit']:
202         size = 'tall'
203 
204     if output in ['latex', 'svg']:
205         if output == 'latex':
206             init_tmpl = INIT_PLOTS_LATEX
207             file_suffix = 'tex'
208         elif output == 'svg':
209             init_tmpl = INIT_PLOTS_SVG
210             file_suffix = 'svg'
211         sizesuffix=''
212         if size == 'tall':
213             if output == 'svg':
214                 sizesuffix = 'size 1000,800'
215             else:
216                 sizesuffix="size 15cm,13cm"
217         else:
218             if output == 'svg':
219                 sizesuffix="size 1000,600"
220             else:
221                 sizesuffix="size 15cm,10cm"
222         plotscript.write(
223             init_tmpl.format(
```

```
                caption=title,
                label=identifier,
                sizesuffix=sizesuffix))
        if specs['variable'] == 'dynamic_size':
            plotscript.write("set xrange [0:512]\n")
            plotscript.write("set xtics 0, 64\n")
            plotscript.write("set format x \"%6.sB\"\n")
        else:
            plotscript.write("unset xtics\n")
            plotscript.write("set xtics autofreq\n")
            plotscript.write("set xrange [*:*]\n")
            plotscript.write("set format x \"%6.s\"\n")

        if size == 'tall':
            if identifier in ['special-calls-arrayelements-c-j-fit',
                              'special-calls-arrayregion-c-j-fit']:
                plotscript.write(
                    "set tmargin at screen 0.8\nset key above box "
                    "horizontal maxrows 8 maxcols 4 samplen 1 "
                    "spacing .5 font \",4\"\n");
            else:
                plotscript.write(
                    "set tmargin at screen 0.85\n"
                    "set key above nobox horizontal\n");
        else:
            plotscript.write("set tmargin at screen 0.95\n")
        plotscript.write("set output '{}'".format(
            os.path.join(plot_directory,
                         "plot-{}-{}.{}".format(
                             measurement_id, identifier, file_suffix))))

    if plotpath:
        # external data
        filename = os.path.join(plotpath, "plot-" + str(uuid.uuid4()) + ".data")
        plotdata = open(filename, 'w')
        specs['convert_to_seconds'] = False # (output == 'latex')
        if output == 'latex':
            specs['tinylabels'] = True
        if output == 'svg':
            specs['scriptlabels'] = True
        plotdata.write(print_benchmarks(data_headers, data_rows, title,
                                        **specs))

    miny = 0
    for row in data_rows:
        for cell in row[1:]:
            if cell < miny:
                miny = cell
    if miny == None:
        miny = '*'

    if output == 'pdf':
        plotscript.write(SET_TITLE_AND_PAGE_LABEL.format(page=identifier,
```

```
                                                        title=title))

    if style == 'binned':
        plotscript.write(template.format(
           title = title, page = identifier, filename = filename, index = 0,
            last_column = len(data_rows[0]),
           xlabel = xlabel, miny=miny, **additional_data))

    elif style == 'fitted_lines':
        length = len(data_headers) - 1
        last_real_column = 1 + length / 2
        first_fitted_column = last_real_column + 1
        plotscript.write(template.format(
           title = title, reps = reps, page = identifier, filename = filename,
            index = 0, last_column = len(data_rows[0]),
           xlabel = xlabel, miny=miny, last_real_column=last_real_column,
            first_fitted_column=first_fitted_column))

    elif style == 'simple_groups':
        grouptitle = GROUPTITLES.get(specs['group'], 'group')
        if key_placement is None:
            plotscript.write("\nunset key\n")
        elif size != 'tall':
            plotscript.write(INIT_KEY[style].format(
                key_placement=key_placement))

        plotscript.write(template.format(
            title = title, reps = reps, page = identifier, filename = filename,
            index = 0, last_column = len(data_rows[0]),
            xlabel = xlabel, miny=miny, grouptitle=grouptitle))

    else:
        grouptitle = GROUPTITLES.get(specs['group'], 'group')
        plotscript.write(template.format(
            title = title, page = identifier, filename = filename, index = 0,
            last_column = len(data_rows[0]),
            key_placement = key_placement, xlabel = xlabel, reps=reps,
            miny=miny, grouptitle=grouptitle))


def print_benchmarks(data_headers, data_rows, title, group=None, variable=None,
                     measure=None, convert_to_seconds=False, tinylabels=False,
                     scriptlabels=False):
    result = '#{0}\n'.format(title)
    if group and variable and measure:
        result = '#measure:{m} variable:{v} group:{g}'.format(
            m=measure, v=variable, g=group)

    prefix = ""
    suffix = ""
    if tinylabels:
        prefix = "\\\\tiny "
    elif scriptlabels:
```

```
        prefix = "\\\\tiny{"
        suffix = "}"
    result = " ".join([format_value("{}{}{}".format(prefix, k, suffix))
                       for k in data_headers])
    result += '\n'

    for row in data_rows:
        results = []
        for i, v in enumerate(row):
            convert = convert_to_seconds and i > 0
            results.append(format_value(v, convert_to_seconds=convert))
        result += ' '.join(results) + '\n'
    result += '\n\n'

    return result

def format_value(value, convert_to_seconds=False):
    if value == None:
        return "-500"
    if type(value) == str:
        return '"{0}"'.format(value)
    if type(value) == int:
        strval = str(value)
        if convert_to_seconds == False:
            return strval
        strval = strval.zfill(10)
        strlen = len(strval)
        return "{}.{}".format(
            strval[0:strlen-9],
            strval[strlen-9:])

    return str(value)

def hex_color_gradient(start, end, point):
    # start, end are tuples with r,g,b values (integer)
    # point is a point between 0 (start) and 1000 (end)
    return "#" + "".join(
        "{:0>2X}".format(
            int(start[i] +
                ((end[i] - start[i]) * (float(point)))))
        for i in range(0,3))
```

/plot_data.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from collections import OrderedDict as odict
from itertools import groupby
```

```
from subprocess import call
from sys import argv
import functools
import pprint
import re
import os
import sys
import shutil
import uuid

import glob
import zipfile

import numpy
from numpy import array

from jni_types import primitive_type_definitions
from jni_types import object_type_definitions, array_types
from datafiles import read_datafiles, read_measurement_metadata
import analysis
from analysis import linear_fit, estimate_measuring_overhead
import gnuplot
import textualtable

FNULL = None

primitive_types = [
    t['java']
    for t in primitive_type_definitions
]

reference_types = [
    t['java']
    for t in array_types.itervalues()
]

reference_types.extend([
    t['java']
    for t in object_type_definitions
])

types = reference_types + primitive_types

plot_axes = {
    'description': 'operaatioiden määrä',
    'parameter_count': 'kutsuparametrien määrä',
    'dynamic_size': 'kohteen koko',
    'direction': 'kutsusuunta',
    'id': 'nimi'
}
pp = pprint.PrettyPrinter(depth=10, indent=4)

debugdata = open('/tmp/debug.txt', 'w')
```

```
59
60 def format_direction(fr, to, latex):
61     if fr == 'J':
62         fr = 'Java'
63     if to == 'J':
64         to = 'Java'
65     if latex:
66         SEPARATOR = '$\\\\rightarrow$'
67     else:
68         SEPARATOR = ' > '
69     return "%s%s%s" % (fr, SEPARATOR, to)
70
71 DIRECTIONS = [('C', 'J'), ('J', 'C'), ('J', 'J'), ('C', 'C')]
72
73 def preprocess_benchmarks(benchmarks, global_values, latex=None):
74     # For allocating benchmarks, the repetition count for individual benchmarks
75     # come from the datafile. For non-allocating, it is a global value.
76     keys = set([key for b in benchmarks for key in b.keys()])
77     if 'repetitions' in keys:
78         benchmarks = [b for b in benchmarks if b['repetitions'] is not None]
79     for b in benchmarks:
80         add_derived_values(b, latex=latex)
81         add_global_values(b, global_values)
82     return benchmarks
83
84 def add_derived_values(benchmark, latex=None):
85     if benchmark.get('response_time_millis') != None:
86         benchmark['response_time'] = benchmark.get('response_time_millis')
87         benchmark['time_unit'] = 'milliseconds'
88         del benchmark['response_time_millis']
89     if benchmark.get('dynamic_size') == None:
90         benchmark['dynamic_variation'] = 0
91         benchmark['dynamic_size'] = 0
92     else:
93         benchmark['dynamic_variation'] = 1
94     if benchmark['no'] == -1:
95         # Custom benchmark, do some name mapping:
96         bid = benchmark['id']
97         rename = True
98         if bid == 'CopyUnicode':
99             bid = 'GetStringRegion'
100        elif bid == 'CopyUTF':
101            bid = 'GetStringRegionUTF'
102        elif bid == 'StringLength':
103            bid = 'GetStringLength'
104        elif bid == 'StringLengthUTF':
105            bid = 'GetStringUTFLength'
106        elif bid == 'ReadUnicode':
107            bid = 'ReadString'
108        elif bid == 'ReadUnicodeCritical':
109            bid = 'ReadStringCritical'
110        elif bid == 'ReadUTF':
111            bid = 'ReadStringUTF'
```

```
112         elif bid == 'ReadUtf':
113             bid = 'ReadStringUTF'
114         elif bid == 'ReadObjectArrayElement':
115             bid = 'GetObjectArrayElement'
116         elif bid == 'WriteObjectArrayElement':
117             bid = 'SetObjectArrayElement'
118         else:
119             rename = False
120         if rename:
121             benchmark['id'] = bid
122 
123     single_type = None
124     if (benchmark.get('parameter_count') == 0):
125         single_type = 'any'
126     elif (benchmark.get('parameter_type_count') == 1):
127         for tp in types:
128             if benchmark.get('parameter_type_{t}_count'.format(t=tp)) != None:
129                 single_type = tp
130                 break
131     benchmark['direction'] = format_direction(
132         benchmark['from'], benchmark['to'], latex)
133     benchmark['single_type'] = single_type
134     if 'Nio' in benchmark['id']:
135         benchmark['nio'] = True
136     else:
137         benchmark['nio'] = False
138 
139 def add_global_values(benchmark, global_values):
140     for key, val in global_values.iteritems():
141         if key not in benchmark or benchmark[key] == None:
142             benchmark[key] = val
143         elif key == 'multiplier' and benchmark[key] != None:
144             benchmark[key] *= val
145 
146 
147 def extract_data(benchmarks,
148                  group=None, variable=None, measure=None,
149                  min_series_length=2, sort=None, min_series_width=None):
150 
151     # info == extra metadata not to be analyzed
152     info = ['no', 'from', 'to', 'lineno', 'start', 'end']
153 
154     if 'class' in benchmarks[0]:
155         info.append('class')
156     if 'description' in benchmarks[0]:
157         info.append('description')
158     if re.match('parameter_type_.+count', variable):
159         info.append('parameter_count')
160     if variable != 'id':
161         info.append('id')
162 
163     # note: all the benchmarks have the same keyset
164     all_keys = set(benchmarks[0].keys())
```

```
165
166     # the actual keys of interest must have the least weight in sorting
167     sort_last = [group, variable, measure] + info
168     controlled_variables = all_keys - set(sort_last)
169     sorted_keys = list(controlled_variables) + sort_last
170
171     sorted_benchmarks = sorted(
172         benchmarks,
173         cmp=functools.partial(comp_function, sorted_keys))
174
175     # 1. group benchmarks into a multi-dimensional list
176     #    with the following structure:
177     #    - compatible-measurements (controlled variables are equal)
178     #      - plots (list of individual data series ie. plots)
179     #        - multiple measurements ()
180     benchmarks = group_by_keys(sorted_benchmarks, controlled_variables)
181     for i, x in enumerate(benchmarks):
182         benchmarks[i] = group_by_keys(x, [group])
183         for j, y in enumerate(benchmarks[i]):
184             benchmarks[i][j] = group_by_keys(y, [variable])
185
186     # 2. statistically combine multiple measurements
187     # for the exact same benchmark and parameters,
188     # and store information about the roles of keys
189
190     for i, compatibles in enumerate(benchmarks):
191         for j, plotgroups in enumerate(compatibles):
192             for k, measured_values in enumerate(plotgroups):
193
194                 plotgroups[k] = aggregate_measurements(
195                     measured_values, measure, stat_fun=min)
196
197             compatibles[j] = odict(
198                 (benchmark[variable], {
199                     'fixed': dict(
200                         (key, benchmark[key]) for key in controlled_variables),
201                     'info': dict((key, benchmark[key]) for key in info),
202                     'variable': variable,
203                     'measure': measure,
204                     'group': group,
205                     variable: benchmark[variable],
206                     measure: benchmark[measure],
207                     group: benchmark[group]
208                 }) for benchmark in plotgroups)
209
210         benchmarks[i] = odict(
211             sorted(((bms.values()[0][group], bms)
212                     for bms in benchmarks[i]),
213                    key=lambda x: x[0]))
214
215     return [x for x in benchmarks
216             if len((x.values())[0]) >= min_series_length]
217
```

```
218
219 def group_by_keys(sorted_benchmarks, keyset):
220     return [
221         list(y) for x, y in groupby(
222             sorted_benchmarks,
223             key=lambda b: [b[k] for k in keyset])]
224
225
226 def aggregate_measurements(benchmarks, measure, stat_fun=min):
227     values = []
228     benchmark = None
229     for benchmark in benchmarks:
230         values.append(benchmark[measure])
231
232     benchmark[measure] = stat_fun(values)
233
234     if len(values) != benchmark['multiplier']:
235         print ("Error: expecting", benchmark['multiplier'],
236                "measurements, got", len(values))
237         debugdata.write(pp.pformat(list(benchmarks)))
238         exit(1)
239
240     return benchmark
241
242
243 def comp_function(keys, left, right):
244     for key in keys:
245         if key not in left and key not in right:
246             continue
247         l, r = left[key], right[key]
248         if l < r:
249             return -1
250         if l > r:
251             return 1
252     return 0
253
254
255 def without(keys, d):
256     if keys == None:
257         return d
258     return dict(((key, val) for key, val in d.iteritems() if key not in keys))
259
260
261 def plot(
262         benchmarks, gnuplot_script, plotpath, metadata_file,
263         keys_to_remove=None, select_predicate=None,
264         group=None, variable=None, measure=None,
265         title=None, style=None, min_series_width=1,
266         key_placement='inside top left',
267         identifier=None,
268         revision=None, checksum=None, output='pdf'):
269
270     if len(benchmarks) > 0 and benchmarks[0].get('is_allocating'):
```

```
271         identifier += '-alloc'
272     if len(benchmarks) > 0:
273         reps = benchmarks[0].get('repetitions')
274 
275     filtered_benchmarks = [
276         without(keys_to_remove, x)
277         for x in benchmarks
278         if select_predicate(x)]
279 
280     variables = set([benchmark[variable] for benchmark in filtered_benchmarks])
281 
282     if len(variables) < 2:
283         print 'Skipping plot without enough data variables', title
284         return
285 
286     if len(filtered_benchmarks) == 0:
287         print 'Error, no benchmarks for', title
288         exit(1)
289 
290     print 'Plotting', title
291 
292     specs = {
293         'group': group,
294         'variable': variable,
295         'measure': measure}
296 
297     data = extract_data(filtered_benchmarks, **specs)
298 
299     index = -1
300 
301     data_len = len([s for s in data if len(s.keys()) >= min_series_width])
302     for series in data:
303         if len(series.keys()) < min_series_width:
304             # there are not enough groups to display
305             continue
306         index += 1
307 
308         plot.page += 1
309         axes_label = plot_axes.get(variable, '<unknown variable>')
310 
311         headers, rows = make_table(
312             series, group, variable, measure, axes_label)
313 
314         assert identifier is not None
315         id_suffix = ""
316         if data_len > 1:
317             id_suffix = "-{}".format(index)
318 
319         gnuplot.output_plot(
320             headers, rows, plotpath, gnuplot_script,
321             title, specs, style, plot.page, identifier + id_suffix, axes_label,
322             output=output, key_placement=key_placement, reps=reps
323         )
```

```
324
325         metadata_file.write("\n\n{0}\n{1}\n\n".format(
326             title, identifier + id_suffix))
327
328         keyvalpairs = series.values()[0].values()[0]['fixed'].items() + [
329             ('variable', axes_label),
330             ('measure', measure),
331             ('grouping', group)]
332
333         for k, v in keyvalpairs:
334             if v != None:
335                 metadata_file.write("{k:<25} {v}\n".format(k=k, v=v))
336
337         metadata_file.write(
338             "\n" + textualtable.make_textual_table(headers, rows))
339
340         id_headers, id_rows = make_table(
341             series, group, variable, 'class', axes_label)
342
343         def make_id(variable_value, item, variable):
344             ret = "/".join([revision, item or '-'])
345             if variable == 'dynamic_size':
346                 ret += "/" + str(variable_value)
347             return ret
348
349         id_rows = [
350             [row[0]] +
351             [make_id(row[0], item, variable) for item in row[1:]]
352             for row in id_rows]
353
354         ttable = textualtable.make_textual_table(id_headers, id_rows)
355         metadata_file.write("\n" + ttable)
356
357         if variable != 'direction' and variable != 'id':
358             x, polys, residuals = linear_fit(rows)
359
360             fitted_curves = []
361             for i, xval in enumerate(x):
362                 current = [xval]
363                 current.extend(rows[i][1:])
364                 current.extend([numpy.polyval(polys[j], xval)
365                                 for j in range(0, len(rows[i]) - 1)])
366                 fitted_curves.append(current)
367
368             plot.page += 1
369             gnuplot.output_plot(
370                 headers + headers[1:], fitted_curves, plotpath, gnuplot_script,
371                 title, specs, 'fitted_lines', plot.page, identifier +
372                 id_suffix + '-fit', axes_label, output=output, reps=reps)
373
374             def simplified_function(poly):
375                 return "{:.3g} * x {:+.3g}".format(poly[0], poly[1])
376             metadata_file.write(
```

```
                "\npolynomial:\n" + textualtable.make_vertical_textual_table(
                    headers[1:], [map(simplified_function, polys)]))
            metadata_file.write(
                "\nresiduals:\n" + textualtable.make_vertical_textual_table(
                    headers[1:], [residuals]))
            metadata_file.write(
                "\nslope:\n" + textualtable.make_vertical_textual_table(
                    headers[1:], [map(lambda p: p[0], polys)]))
            metadata_file.write(
                "\nintercept:\n" + textualtable.make_vertical_textual_table(
                    headers[1:], [map(lambda p: p[1], polys)]))
    return data

plot.page = 0

def convert_to_seconds(value):
    if type(value) == int:
        strval = str(value)
        if convert_to_seconds == False:
            return strval
        strval = strval.zfill(10)
        strlen = len(strval)
        return float("{}.{}".format(
            strval[0:strlen-9],
            strval[strlen-9:]))
    return value

def make_table(series, group, variable, measure, axes_label):
    all_benchmark_variables_set = set()
    for bm_list in series.itervalues():
        all_benchmark_variables_set.update(bm_list.keys())

    all_benchmark_variables = sorted(list(all_benchmark_variables_set))

    rows = []

    headers = (
        [axes_label] +
        [k for k in series.iterkeys()]
    )

    for v in all_benchmark_variables:
        row = []
        row.append(v)
        for key, grp in series.iteritems():
            val = grp.get(v, {}).get(measure, None)
            if val is None:
                val = grp.get(v, {}).get('info', {}).get(measure, None)
            if measure == 'response_time':
                val = convert_to_seconds(val)
            row.append(val)
        rows.append(row)

```

```
430     if variable == 'id':
431         rows = sorted(rows, key=lambda x: x[1] or -1)
432 
433     return headers, rows
434 
435 
436 def binned_value(minimum, width, value):
437     return width * (int(value - minimum) / int(width)) + minimum
438 
439 
440 def plot_distributions(
441         all_benchmarks, output, plotpath, gnuplotcommands, bid,
442         metadata_file, plot_type=None, latex=None, **kwargs):
443 
444     output_type = 'screen'
445     if plot_type != 'animate':
446         output_type = 'pdf'
447 
448     gnuplot.init(gnuplotcommands, output, bid, output_type=output_type)
449     measure = 'response_time'
450 
451     keyset = set(all_benchmarks[0].keys()) - \
452         set([measure, 'lineno', 'start', 'end'])
453     comparison_function = functools.partial(comp_function, keyset)
454     sorted_benchmarks = sorted(all_benchmarks, cmp=comparison_function)
455 
456     for group in group_by_keys(sorted_benchmarks, keyset):
457         if plot_type != None:
458             keyf = lambda x: x['lineno']
459         else:
460             keyf = lambda x: x[measure]
461 
462         frame_count = 1
463         if plot_type != None:
464             frame_count = 256
465 
466         current_frame = frame_count
467         all_values = [b[measure] for b in sorted(group, key=keyf)]
468         while current_frame > 0:
469 
470             if current_frame == frame_count:
471                 frame_ratio = 1
472             else:
473                 frame_ratio = float(current_frame) / frame_count
474             values = array(all_values[0:int(frame_ratio * len(all_values))])
475 
476             bin_width = 500
477             min_x = numpy.amin(all_values)
478             max_x = numpy.amax(all_values)
479 
480             bin_no = (max_x - min_x) / bin_width
481 
482             hgram, bin_edges = numpy.histogram(values, bins=max(bin_no, 10))
```

```
483
484             mode = bin_edges[numpy.argmax(hgram)]
485             min_x = mode - 100000
486             max_x = mode + 100000
487
488             if current_frame == frame_count:
489                 metadata_file.write(
490                     'Direction {0}\n'.format(group[0]['direction']))
491
492                 gnuplotcommands.write(
493                     gnuplot.templates['binned_init'].format(
494                         title='%s %s' % (group[0]['id'], group[
495                                          0]['direction']),
496                         binwidth=bin_edges[1] - bin_edges[0],
497                         min_x=min_x, max_x=max_x,
498                         max_y=numpy.max(hgram)))
499
500                 if plot_type == 'animate':
501                     gnuplotcommands.write('pause -1\n')
502
503                 elif plot_type == 'gradient':
504                     gnuplotcommands.write("set multiplot\n")
505
506             current_frame -= 1
507
508             if plot_type == None:
509                 gnuplotcommands.write(
510                     gnuplot.templates['binned_frame'].format(
511                         datapoints='', color='#000033',
512                         values='\n'.join(['{} {} {}'.format(val, count, val)
513                                           for val, count in zip(
514                                                   bin_edges, hgram)])))
515
516             elif plot_type == 'gradient':
517                 gnuplotcommands.write(
518                     gnuplot.templates['binned_frame'].format(
519                         datapoints='',
520                         color=gnuplot.hex_color_gradient(
521                             (125, 0, 0), (255, 255, 0), 1 - frame_ratio),
522                         values='\n'.join(['{} {} {}'.format(val, count, val)
523                                           for val, count in zip(
524                                                   bin_edges, hgram)])))
525
526         gnuplotcommands.write("set xtics\n")
527         gnuplotcommands.write("set ytics\n")
528
529
530 def plot_benchmarks(
531         all_benchmarks, output, plotpath, gnuplotcommands, bid, metadata_file,
532         plot_type=None, revision=None, checksum=None, latex=None):
533
534     output_type = 'pdf'
535     if latex == 'plotlatex':
```

```
536         output_type = 'latex'
537     elif latex == 'plotsvg':
538         output_type = 'svg'
539 
540     gnuplot.init(gnuplotcommands, output, bid, output_type=output_type)
541 
542     type_counts = ["parameter_type_{t}_count".format(t=tp) for tp in types]
543     keys_to_remove = type_counts[:]
544     keys_to_remove.extend(
545         ['parameter_type_count', 'single_type', 'dynamic_variation'])
546 
547     benchmarks = [bm for bm in all_benchmarks if bm['no'] != -1]
548     defaults = [benchmarks, gnuplotcommands, plotpath]
549 
550     overhead_estimates = {}
551     overhead_benchmarks = [
552         bm for bm in all_benchmarks
553         if bm['no'] == -1 and 'Overhead' in bm ['id']]
554     for loop_type in ['AllocOverhead', 'NormalOverhead']:
555         for from_lang in ['C', 'J']:
556             language_name = from_lang
557             if language_name == 'J': language_name = 'Java'
558             overhead_estimates[from_lang] = {}
559             overhead_data = plot(
560                 overhead_benchmarks, gnuplotcommands, plotpath, metadata_file,
561                 style='simple_groups',
562                 key_placement=None,
563                 title='Mittauksen perusrasite ({})'.format(language_name),
564                 identifier='{}-{}'.format(loop_type.lower(), from_lang.lower()),
565                 keys_to_remove=[],
566                 select_predicate=(
567                         lambda x: x['from'] == from_lang and loop_type in x['id']),
568                 group='from',
569                 measure='response_time',
570                 variable='description',
571                 revision=revision,
572                 checksum=checksum,
573                 output=output_type)
574 
575             if overhead_data == None:
576                 continue
577             if len(overhead_data) > 1:
578                 print ('Error, more loop types than expected.',
579                        len(overhead_data))
580                 exit(1)
581 
582             series = overhead_data[0]
583             headers, rows = make_table(series,
584                                        'from',
585                                        'description',
586                                        'response_time',
587                                        'workload')
588             est = estimate_measuring_overhead(rows[1:])
```

```
589             overhead_estimates[from_lang][loop_type] = est[0]
590             metadata_file.write('Overhead ' + from_lang + ' ' + str(est[0]))
591
592     for i, ptype in enumerate(types):
593         plot(
594             benchmarks, gnuplotcommands, plotpath, metadata_file,
595             title='{}-tyyppiset kutsuparametrit'.format(ptype),
596             identifier='basic-call-{}'.format(ptype),
597             style='simple_groups',
598             keys_to_remove=(
599                 keys_to_remove +
600                 ['dynamic_size'] +
601                 ['has_reference_types']),
602             select_predicate=lambda x: (
603                 x['single_type'] in [ptype, 'any'] and
604                 x['dynamic_size'] == 0),
605             group='direction',
606             variable='parameter_count',
607             measure='response_time',
608             revision=revision, checksum=checksum, output=output_type)
609
610     for fr, to in DIRECTIONS:
611         direction = format_direction(fr, to, latex)
612         plot(
613             benchmarks, gnuplotcommands, plotpath, metadata_file,
614             title='Vaihteleva argumentin koko kutsusuunnassa ' + direction,
615             identifier='variable-argument-size-{}-{}'.format(fr.lower(),
616                                                              to.lower()),
617             style='simple_groups',
618             keys_to_remove=type_counts,
619             select_predicate=(
620                 lambda x: (
621                     x['direction'] == direction and
622                     x['has_reference_types'] == 1 and
623                     x['single_type'] in reference_types and
624                     x['parameter_count'] == 1)),
625             group='single_type',
626             variable='dynamic_size',
627             measure='response_time',
628             revision=revision, checksum=checksum, output=output_type)
629
630     for fr, to in DIRECTIONS:
631         direction = format_direction(fr, to, latex)
632         plot(
633             benchmarks, gnuplotcommands, plotpath, metadata_file,
634             title='Vaihteleva paluuarvon koko kutsusuunnassa ' + direction,
635             identifier='variable-return-value-size-{}-{}'.format(fr.lower(),
636                                                                  to.lower()),
637             style='simple_groups',
638             keys_to_remove=type_counts,
639             select_predicate=(
640                 lambda x: x['has_reference_types'] == 1
641                 and x['direction'] == direction
```

```
642                 and x['return_type'] != 'void'),
643             group='return_type',
644             variable='dynamic_size',
645             measure='response_time',
646             revision=revision, checksum=checksum, output=output_type)
647
648     keys_to_remove = type_counts[:]
649     keys_to_remove.append('has_reference_types')
650     keys_to_remove.append('dynamic_variation')
651
652     for fr, to in DIRECTIONS:
653         direction = format_direction(fr, to, latex)
654         plot(
655             benchmarks, gnuplotcommands, plotpath, metadata_file,
656             style='simple_groups',
657             title='Parametrityyppien vertailu ' + direction,
658             identifier='basic-call-all-types-{}-{}'.format(fr.lower(),
659                                                            to.lower()),
660             keys_to_remove=keys_to_remove,
661             select_predicate=(
662                 lambda x: x['direction'] == direction),
663             group='single_type',
664             variable='parameter_count',
665             measure='response_time',
666             revision=revision, checksum=checksum, output=output_type)
667
668     plot(
669         benchmarks, gnuplotcommands, plotpath, metadata_file,
670         style='named_columns',
671         title='Paluuarvon tyypit',
672         identifier='return-value-types',
673         keys_to_remove=['has_reference_types', 'dynamic_variation'],
674         select_predicate=(
675             lambda x: x['dynamic_size'] == 0 and
676             x['return_type'] != 'void'),
677         group='return_type',
678         measure='response_time',
679         variable='direction',
680         min_series_width=2,
681         revision=revision, checksum=checksum, output=output_type)
682
683     def utf(b):
684         return 'UTF' in b['id'] or 'Utf' in b['id']
685
686     filters = {
687         'utf': utf,
688         'arrayregion': lambda x: 'ArrayRegion' in x['id'],
689         'bytebufferview': lambda x: 'ByteBufferView' in x['id'],
690         'unicode': lambda b: not utf(b) and 'String' in b['id'],
691         'arrayelements': (lambda x:
692                           'ArrayElements' in x['id'] or
693                           'ArrayLength' in x['id'] or
694                           'ReadPrimitive' in x['id']),
```

```
695     }
696     def uncategorized(x):
697         for f in filters.values():
698             if f(x):
699                 return False
700         return True
701 
702     benchmarks = {}
703     for key, f in filters.iteritems():
704         benchmarks[key] = [
705             bm for bm in all_benchmarks
706             if bm['no'] == -1 and f(bm)]
707 
708     benchmarks['uncategorized'] = [
709         bm for bm in all_benchmarks
710         if bm['no'] == -1 and 'Overhead' not in bm['id'] and uncategorized(bm)]
711 
712     custom_benchmarks = benchmarks['uncategorized']
713 
714     for fr, to in DIRECTIONS:
715         direction = format_direction(fr, to, latex)
716         plot(
717             custom_benchmarks, gnuplotcommands, plotpath, metadata_file,
718             style='simple_groups',
719             title='Erityiskutsut suunnassa ' + direction,
720             identifier='special-calls-{}-{}'.format(fr.lower(), to.lower()),
721             select_predicate=(
722                 lambda x: (x['direction'] == direction and
723                            x['dynamic_variation'] == 1)),
724             group='id',
725             measure='response_time',
726             variable='dynamic_size',
727             revision=revision, checksum=checksum, output=output_type)
728 
729         plot(
730             benchmarks['arrayregion'], gnuplotcommands, plotpath,
731             metadata_file,
732             style='simple_groups',
733             title='Erityiskutsut suunnassa ' + direction,
734             identifier='special-calls-arrayregion-{}-{}'.format(fr.lower(),
735                                                                 to.lower()),
736             select_predicate=(
737                 lambda x: (x['direction'] == direction and
738                            x['dynamic_variation'] == 1)),
739             group='id',
740             measure='response_time',
741             variable='dynamic_size',
742             revision=revision, checksum=checksum, output=output_type)
743 
744         plot(
745             benchmarks['arrayelements'], gnuplotcommands, plotpath,
746             metadata_file,
747             style='simple_groups',
```

```
748             title='Erityiskutsut suunnassa ' + direction,
749             identifier='special-calls-arrayelements-{}-{}'.format(fr.lower(),
750                                                                   to.lower()),
751             select_predicate=(
752                 lambda x: (x['direction'] == direction and
753                            x['dynamic_variation'] == 1)),
754             group='id',
755             measure='response_time',
756             variable='dynamic_size',
757             revision=revision, checksum=checksum, output=output_type)
758
759         plot(
760             benchmarks['utf'], gnuplotcommands, plotpath, metadata_file,
761             style='simple_groups',
762             title='UTF-merkkijonot suunnassa ' + direction,
763             identifier='special-calls-utf-{}-{}'.format(fr.lower(),
764                                                         to.lower()),
765             select_predicate=(
766                 lambda x: (x['direction'] == direction and
767                            x['dynamic_variation'] == 1)),
768             group='id',
769             measure='response_time',
770             variable='dynamic_size',
771             revision=revision, checksum=checksum, output=output_type)
772
773         plot(
774             benchmarks['unicode'], gnuplotcommands, plotpath, metadata_file,
775             style='simple_groups',
776             key_placement='inside bottom left',
777             title='Unicode-merkkijonot suunnassa ' + direction,
778             identifier='special-calls-unicode-{}-{}'.format(fr.lower(),
779                                                             to.lower()),
780             select_predicate=(
781                 lambda x: (x['direction'] == direction and
782                            x['dynamic_variation'] == 1)),
783             group='id',
784             measure='response_time',
785             variable='dynamic_size',
786             revision=revision, checksum=checksum, output=output_type)
787
788         plot(
789             benchmarks['bytebufferview'], gnuplotcommands, plotpath,
790             metadata_file,
791             style='simple_groups',
792             title='Erityiskutsut suunnassa ' + direction,
793             identifier='special-calls-bytebufferview-{}-{}'.format(fr.lower(),
794                                                                    to.lower()),
795             select_predicate=(
796                 lambda x: (x['direction'] == direction and
797                            x['dynamic_variation'] == 1 and
798                            'Bulk' not in x['id'])),
799             group='id',
800             measure='response_time',
```

```
801             variable='dynamic_size',
802             revision=revision, checksum=checksum, output=output_type)
803 
804         plot(
805             benchmarks['bytebufferview'], gnuplotcommands, plotpath,
806             metadata_file,
807             style='simple_groups',
808             title='Erityiskutsut suunnassa ' + direction,
809             identifier='special-calls-bulk-bytebufferview-{}-{}'.format(
810                 fr.lower(), to.lower()),
811             select_predicate=(
812                 lambda x: (x['direction'] == direction and
813                            x['dynamic_variation'] == 1 and
814                            'Bulk' in x['id'])),
815             group='id',
816             measure='response_time',
817             variable='dynamic_size',
818             revision=revision, checksum=checksum, output=output_type)
819 
820     plot(
821         custom_benchmarks, gnuplotcommands, plotpath, metadata_file,
822         style='histogram',
823         title='Erityiskutsujen vertailu eri kutsusuunnissa',
824         identifier='special-calls-non-dynamic',
825         select_predicate=(
826             lambda x: (
827                 x['dynamic_variation'] == 0 and
828                 'Field' in x['id'])),
829         group='direction',
830         measure='response_time',
831         variable='id',
832         revision=revision, checksum=checksum, output=output_type)
833 
834 
835 MEASUREMENT_FILE = 'measurements.txt'
836 DEVICE_PATH = '/sdcard/results'
837 PLOTPATH = '/tmp'
838 TOOL_NAMESPACE = 'fi.helsinki.cs.tituomin.nativebenchmark.measuringtool'
839 
840 
841 def sync_measurements(dev_path, host_path, filename, update=True):
842     old_path = host_path + '/' + filename
843     tmp_path = '/tmp/' + filename
844     if not update and os.path.exists(old_path):
845         print 'No sync necessary'
846         return
847 
848     kwargs = {}
849     if FNULL is not None:
850         kwargs['stdout'] = FNULL
851         kwargs['stderr'] = FNULL
852 
853     try:
```

```
854         success = call(['adb', 'pull',
855                         dev_path + '/' + filename,
856                         tmp_path], **kwargs)
857     except OSError:
858         success = -1
859     if success == 0:
860         if os.path.exists(old_path):
861             size_new = os.path.getsize(tmp_path)
862             size_old = os.path.getsize(old_path)
863             if size_new < size_old:
864                 print ("Warning: new file contains less data than "
865                        "the old. Aborting.")
866                 exit(2)
867         shutil.move(tmp_path, old_path)
868 
869     else:
870         print "Could not get new measurements, continuing with old."
871 
872 def render_perf_reports_for_measurement(identifier, measurements,
873                                         measurement_path, output_path,
874                                         output_command=False):
875     path = identifier.split("/")
876     if len(path) < 2:
877         print 'Invalid identifier {}'.format(identifier)
878         exit(1)
879     if len(path) == 3:
880         revision, class_, dynamic_size = path
881     elif len(path) == 2:
882         revision, class_ = path
883         dynamic_size = None
884 
885     def match_measurement(measurement):
886         m = measurement[0]
887         return (m.get('code-revision') == revision and
888                 m.get('tool') == 'LinuxPerfRecordTool')
889 
890     def match_measurement_run(m):
891         if m.get('class').lower() != class_.lower():
892             return False
893         if dynamic_size and m.get('dynamic_size') != int(dynamic_size):
894             return False
895         if 'Filename' not in m or m['Filename'] is None:
896             return False
897         return True
898 
899     datafiles = []
900     for measurement in filter(match_measurement, measurements):
901         mid = measurement[0].get('id')
902         zpath = os.path.join(measurement_path, 'perfdata-{}.zip'.format(mid))
903         try:
904             measurement_zipfile = zipfile.ZipFile(zpath, 'r')
905             datafiles.append({
906                 'zip': measurement_zipfile,
```

```
                'zip_path': zpath,
                'mid': mid,
                'csv': measurement_zipfile.open(
                    '{0}/benchmarks-{0}.csv'.format(mid))
            })
        except zipfile.BadZipfile:
            print 'Bad zip file %s' % zpath
        except IOError as e:
            print 'Problem with zip file %s' % zpath
            print e

    benchmarks = []
    for df in datafiles:
        benchmarks.append({
            'zip': df['zip'],
            'mid': df['mid'],
            'metadata': read_datafiles([df['csv']], silent=output_command)
        })

    matching_benchmarks = []
    for bm in benchmarks:
        for row in bm['metadata']:
            if match_measurement_run(row):
                matching_benchmarks.append({
                    'zip': bm['zip'],
                    'mid': bm['mid'],
                    'filename': row['Filename']
                })

    for record in matching_benchmarks:
        perf_file = record['zip'].extract('{}/{}'.format(record['mid'],
                                                         record['filename']),
                                          '/tmp')
        try:
            command_parts = [
                "perf report",
                "-i {}",
                "--header",
                "--symfs=/home/tituomin/droid-symbols",
                "--kallsyms=/home/tituomin/droid/linux-kernel/kallsyms"
            ]
            command_parts.extend([
                "-g graph,0,caller",
                "--stdio",
                "| c++filt",
                ">/tmp/out.txt"
            ])
            command = " ".join(command_parts).format(perf_file)
            if output_command:
                print command
                exit(0)
            else:
                call([command], shell=True)
```

```
960         except OSError as e:
961             print e.filename, e.message, e.args
962 
963     for f in datafiles:
964         f['zip'].close()
965     print "Profile for identifier", identifier
966     with open('/tmp/out.txt', 'r') as f:
967         print f.read()
968     exit(0)
969 
970 if __name__ == '__main__':
971     if len(argv) < 4 or len(argv) > 6:
972         print argv[0]
973         print ("\n    Usage: %s input_path output_path "
974                "limit [pdfviewer] [separate]\n").format(argv[0])
975         exit(1)
976 
977     FNULL = open(os.devnull, 'w')
978 
979     method = argv[0]
980     measurement_path = os.path.normpath(argv[1])
981     output_path = argv[2]
982 
983     if 'plotlatex' in method:
984         latex = 'plotlatex'
985         method = 'curves'
986     elif 'plotsvg' in method:
987         latex = 'plotsvg'
988         method = 'curves'
989     else:
990         latex = None
991 
992     output_command = False
993     if len(argv) > 5:
994         if argv[5] == 'show-command':
995             output_command = True
996 
997     limit = argv[3]
998     if len(argv) > 4:
999         pdfviewer = argv[4]
1000    else:
1001        pdfviewer = None
1002 
1003    if len(argv) == 6:
1004        group = (not argv[5] == "separate")
1005    else:
1006        group = True
1007 
1008    if output_command:
1009        system_stdout = sys.stdout
1010        system_stderr = sys.stderr
1011        sys.stdout = FNULL
1012        sys.stderr = FNULL
```

```
    sync_measurements(DEVICE_PATH, measurement_path, MEASUREMENT_FILE)

    f = open(os.path.join(measurement_path, MEASUREMENT_FILE))

    try:
        measurements = read_measurement_metadata(f, group)
    finally:
        f.close()

    limited_measurements = (
        filter(lambda x: int(x[0].get('repetitions', 0)) >= int(limit),
               measurements.values()))

    # ID = revision/checksum/class[/dynamic_size]
    if 'perf_select' in method:
        identifier = argv[4]
        if output_command:
            sys.stdout = system_stdout
            sys.stderr = system_stderr
            FNULL.close()
        render_perf_reports_for_measurement(
            identifier, limited_measurements, measurement_path,
            output_path, output_command=output_command)
        exit(0)

    csv_files = set()
    for f in glob.iglob(measurement_path + '/benchmarks-*.csv'):
        try:
            csv_files.add(f.split('.csv')[0].split('benchmarks-')[1])
        except IndexError:
            pass

    if len(limited_measurements) > 20:
        i = len(limited_measurements) - 20 + 1
        splice = limited_measurements[-20:]
    else:
        i = 1
        splice = limited_measurements

    print "\nAvailable compatible measurements. Choose one"
    for m in splice:
        b = m[0]
        warning = ""
        if int(b.get('rounds')) == 0:
            warning = " <---- WARNING INCOMPLETE MEASUREMENT"
        print """
    [{idx}]:     total measurements: {num}
                           local: {local}
                    repetitions: {reps}
                    description: {desc}
                         rounds: {rounds}{warning}
                             id: {mid}
```

```
                        checksum: {ck}
                        revision: {rev}
                            tool: {tool}
                             cpu: {freq} KHz
                             set: {bset}
                          filter: {sfilter}
                           dates: {first} -
                                  {last}
    """.format(
            local=b.get('id') in csv_files,
            num=len(m),
            mid=b.get('id'),
            idx=i,
            warning=warning,
            last=m[-1]['end'],
            rounds=reduce(lambda x, y: y + x, [int(b['rounds']) for b in m]),
            reps=b.get('repetitions'),
            ck=b.get('code-checksum'),
            rev=b.get('code-revision'),
            tool=b.get('tool'),
            freq=b.get('cpu-freq'),
            bset=b.get('benchmark-set'),
            desc=b.get('description'),
            sfilter=b.get('substring-filter'),
            first=b.get('start')
        )

        i += 1

    try:
        response = raw_input("Choose set 1-{last} >> ".format(last=i - 1))
    except EOFError:
        print 'Exiting.'
        exit(1)

    benchmark_group = limited_measurements[int(response) - 1]

    filenames = []
    ids = []
    multiplier = 0
    for measurement in benchmark_group:
        if 'LinuxPerfRecordTool' in measurement['tool']:
            basename = "perfdata-{n}.zip"
        else:
            basename = "benchmarks-{n}.csv"
        filenames.append(
            basename.format(n=measurement['id']))
        if 'logfile' in measurement:
            filenames.append(measurement['logfile'])
        ids.append(measurement['id'])
        multiplier += int(measurement['rounds'])

    files = []
```

```
    for filename in filenames:
        sync_measurements(DEVICE_PATH, measurement_path,
                          filename, update=False)
        if filename not in [m.get('logfile') for m in benchmark_group]:
            files.append(open(os.path.join(measurement_path, filename)))

    first_measurement = benchmark_group[0]

    global_values = {
        'repetitions': first_measurement['repetitions'],
        'is_allocating': first_measurement['benchmark-set'] == 'ALLOC',
        'multiplier': multiplier
    }

    perf = False
    if 'LinuxPerfRecordTool' in first_measurement['tool']:
        print 'Perf data downloaded.'
        perf = True
    if not perf:
        try:
            benchmarks = read_datafiles(files)

        finally:
            for f in files:
                f.close()

        benchmark_group_id = os.getenv('PLOT_ID', str(uuid.uuid4()))
        plot_prefix = 'plot-{0}'.format(benchmark_group_id)

        if latex is not None:
            output_filename = os.path.join(output_path, plot_prefix)
        else:
            output_filename = os.path.join(output_path, plot_prefix + '.pdf')
        plot_filename = plot_prefix + '.gp'

        plotfile = open(os.path.join(output_path, plot_filename), 'w')
        metadata_file = open(os.path.join(
            output_path, plot_prefix + '-metadata.txt'), 'w')

        measurement_ids = " ".join(ids)
        metadata_file.write("-*- mode: perf-report; -*-\n\n")
        metadata_file.write("id: {0}\n".format(benchmark_group_id))
        metadata_file.write("measurements: {0}\n".format(measurement_ids))

        benchmarks = preprocess_benchmarks(benchmarks, global_values,
                                           latex=latex)

        animate = False
        if pdfviewer == 'anim':
            plot_type = 'animate'
            pdfviewer = None
        elif pdfviewer == 'gradient':
            plot_type = 'gradient'
```

```
            pdfviewer = None
        else:
            plot_type = None

    if 'curves' in method:
        function = plot_benchmarks
    elif 'distributions' in method:
        function = plot_distributions
    if perf or not function:
        exit(0)

    function(
        benchmarks,
        output_filename,
        PLOTPATH,
        plotfile,
        benchmark_group_id,
        metadata_file,
        plot_type=plot_type,
        revision=first_measurement['code-revision'],
        checksum=first_measurement['code-checksum'],
        latex=latex)

    plotfile.flush()
    plotfile.close()
    if plot_type == 'animate':
        print "Press enter to start animation."
    call(["gnuplot", plotfile.name])
    if pdfviewer:
        call([pdfviewer, str(output_filename)])
    print "Final plot",
    if 'animate' != plot_type:
        print str(output_filename)
    else:
        print str(plot_filename)
    print(benchmark_group_id)
    exit(0)
```

/textualtable.py

```
#/usr/bin/python

def make_textual_table(headers, rows):
    result = ""
    max_widths = []

    for x in headers:
        max_widths.append(len(str(x)))

```

```
    for row in rows:
        for i, x in enumerate(row):
            l = len(str(x))
            if max_widths[i] < l:
                max_widths[i] = l

    row_format = ["{{:>{w}}}   ".format(w=w) for w in max_widths]
    row_format = "".join(row_format) + "\n"

    result += row_format.format(*headers)
    for row in rows:
        result += row_format.format(*row)
    return result

def make_vertical_textual_table(headers, elements):
    result = ""
    max_width = max((len(x) for x in headers))

    header_format = "{{:>{w}}}".format(w=max_width)

    for i in range(0, len(headers)):
        result += header_format.format(headers[i])
        for group in elements:
            result += "    "
            result += str(group[i])
        result += "\n"

    return result
```