SPASS-meter Quick Introduction for SEI

Distributions

Available at http://projects.sse.uni-hildesheim.de/SPASS-meter/

- Version 0.5: Basic functionality based on synchronous internal event processing
- Version 0.6: Asynchronous internal event processing, several performance optimizations done, currently memory unallocation partly disabled

Android Build Process

This section we briefly describe the integration into the Android build process. Currently, Android apps need to be instrumented statically as (at least until Android version 2.3) does not support dynamic instrumentation. The `spass-meter-ant.jar` provides the tools for performing a static instrumentation, in particular a specific ANT task. However, we started our work on the Android build process before ANT supported task extension so that we rebuild the Android compile steps. It might be that this is easier in your specific environment.
Basically, I would recommend packaging all Java classes (before calling the `dex` tool) into one JAR and statically instrument that jar before turning it into an APK file. Using the SPASS-meter ANT integration this can be achieved as shown below:

```
<spassInstrumenter
     classpathref="${classpath.instrumentation}"
     in="${app.jar}"
     out="${instrumented.dir}"
     params="${spass.args}" />
```

Here, `classpath.instrumentation` shall contain the other JAR files in the SPASS-meter distribution for Android, `app.jar` the bundled App and instrumented.dir the output directory for the instrumented version of `app.jar`. `spass.args` contains the command line arguments for the SPASS-meter instrumenter, which will be described below.

Global SPASS-meter arguments

The configuration of SPASS-meter is done on two levels, a) on the global level specifying data which applies to the entire instrumentation process and b) on the level of Java elements to be instrumented. While we describe the configuration on the global level in this section, we will describe the Java element level in the next two sections (using an external XML configuration file or Java source code annotations).
The following (selected) global configuration options are supported by SPASS-meter:

- `logLevel` specifies one of the Java log levels (`FINE, FINER, FINEST, INFO, OFF, SEVERE, WARNING`). Example `logLevel=OFF`.
- `out` the output file for a post-mortem summary file. There are integrations with JMX (not available on Android) or OW2 wildcat for providing standardized access to the monitored values at runtime but we shall discuss this in more detail for your case. Example `out=/mnt/sdcard/tmp/meter.log`.
- configDetect a specific parameter for DSPL experiments, please use the value `false`, i.e. `configDetect=false` in your case.
- `instrumentJavaLib` enables or disables the instrumentation of the Java library in the static instrumentation case. However, currently the built-in Android library cannot be

instrumented due to security restrictions. Please use `instrumentJavaLib=false` in your case.

- `registerThreads` enables or disables explicit registration of threads with the native library. In case of Android this must be `true`.
- `memoryAccountingType` defines how memory accounting shall happen. In order to simplify the configuration, please use `memoryAccountingType=CREATION` for the moment.

Note: All these global settings can also be used for dynamic instrumentation as parameters to the Java instrumentation agent (JVM parameter `–javaagent spass-meter-ia.jar=<`*`global parameter list`*`>`).

XML-Configuration of the System Under Monitoring

The individual elements of a Java program to be monitored can be specified in an external XML file or using source code annotations. Basic concept here is the so called *monitoring group*, a virtual group of java elements (methods or classes) which shall be monitored with the same monitoring semantics. The monitored values for all members of a monitoring group are aggregated. In this section we briefly describe the XML configuration of a System under monitoring, i.e. how to define the monitoring groups.

The following XML fragment shows how to configure a monitoring group called MyGroup for the class my.package.Main. For demonstration purpose, the `equals` method in that class is excluded from monitoring and the main method is explicitly marked as start and end point of the monitoring (we used `startSystem` at `onCreate` and `endSystem` when the App definitively ends).

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://sse.uni-hildesheim.de/instrumentation">
  <namespace name=”my”>
    <namespace name=”package”>
      <module name=”Main”>
        <monitor id=”MyGroup”/>
        <behavior signature=“equals(java.lang.Object)“>
          <excludeFromMonitoring/>
        </behavior>
        <behavior signature=“main(java.lang.String[])“>
          <endSystem/>
          <startSystem/>
        </behavior>
      </module>
    </namespace>
  </namespace>
</configuration>
```

There are several ways of influencing the resources to be accounted. In particular, the global options (to be mentioned in the XML `configuration` element) `accountableResources` which defines the resource to be monitored at all (default is `ALL`, a selection of `CPU_TIME`, `FILE_IO`, `NET_IO` and `MEMORY` may be given) and `sumResources`, i.e. the specific resources to be accounted for the entire Java program. Also for each individual monitoring group, an attribute resources with the specific resources may be specified (`accountableResources` is used if none is specified).

Annotation-based configuration

The XML-based configuration is beneficial, if the source code of the system under monitoring must not be changed or is not available. Otherwise, all options expressed in the XML configuration file can also be stated directly at the element using SPASS-meter annotations. Regarding the XML configuration example shown above, this would be

`@monitor(id=”MyGroup”)` at `my.package.Main`
`@excludeFromMonitoring` at the `equals` method of `my.package.Main`
`@endSystem` at the `main` method of `my.package.Main`
`@startSystem` at the `main` method of `my.package.Main`