Java Performance Measurement Framework
An Overview (and Status Report)

Lubomír Bulej

DISTRIBUTED SYSTEMS RESEARCH GROUP
http://dsrg.mff.cuni.cz

CHARLES UNIVERSITY PRAGUE
Faculty of Mathematics and Physics
Background: performance measurement

... in case of a library

- Create benchmark application(s) for typical use cases, measure the duration of actions and additional information such as e.g. amount of data sent over network, and store the measured data

... in case of an application

- Decide what needs to be measured, find spots in the application where to measure it, modify the application to perform the measurement, and the rest as above...
Goal: simplify the measurement

Application specific tasks
- Define and ensure generation of performance events
- Define data to be collected when events occur

Common, technical tasks
- Collect and store data associated with the events

Simplification through separation of concerns
- Application instrumentation
- Measurement configuration and control
- Data collection, storage, and transport
Solution: measurement framework

Allow to define performance events

- Specific, through manual instrumentation
  - Counting the number of iterations in selected loops
- Generic, through automatic instrumentation
  - Method invocations on component interfaces (even POJOs)

Allow to configure what data to collect

- Event-specific data
  - time stamps, iteration counts, sizes of arguments, ...
- Generic system-wide data
  - amount of data transferred over network, ...

Collect and store the data

- Without the user having to care about details
Prototype demo movie
Framework architecture overview

Performance Measurement Framework

Management
- Event Source Registration
- Experiment Configuration

Data Delivery
- Public interfaces
- Transport Methods
- Aggregation

Event Sources
- External
- Internal

Data Sampling

Data Buffering

Event Processing

Performance Data Access
Basic concepts: events

Event type

- interface with methods for group of related events

```java
public interface AtomicEventDelegate extends EventDelegate {
    void notify (long eventTime);
}

public interface EnterLeaveMethodEventDelegate extends EventDelegate {
    void notifyEnterMethod (long enterTime);
    void notifyLeaveMethod (long leaveTime);
}
```
Basic concepts: event sources

Event source

- Supports multiple event groups of various types
  - e.g. single event source could correspond to a single component interface, the event source providing events corresponding to entering and leaving each of the interface methods

- Individual event groups can be enabled/disabled
  - i.e. for particular method

- Event sources can be enabled/disabled
  - e.g. to reduce overhead if no events are enabled
public interface EventSource {
    String id ();
    List<Event> events ();

    void setTimerCounter (TimerCounter timer);
    void setEventDelegate (int eventIndex, EventDelegate delegate);

    void enable ();
    void disable ();
    boolean isEnabled ();

    void enableEvent (int eventIndex);
    void disableEvent (int eventIndex);
    boolean isEventEnabled (int eventIndex);
}
Normal event sources = external event sources

- Client needs to implement the event source interface and register instances with the framework
  - Typically provided by automatic instrumentation
  - Framework controls the external event source

```java
public void startFcPerformanceMonitor () {
    final List<ComponentInterface> interfaces =
        __getOrderedBusinessInterfaces (this.__component);
    this.__interfaceEventSources = __createInterfaceEventSources (interfaces);

    final EventRegistrar er = Jpmf.getEventRegistrar ();
    for (final InterfaceEventSource ies : this.__interfaceEventSources) {
        er.registerEventSource (ies);
    }
}
```
Basic concepts: event triggers

Event triggers = internal event sources

- Simplified internal implementation of event source interface, simple events only
  - Typically instantiated by manual instrumentation
  - Provides client with methods for generating events
  - Framework controls the internal event source

```java
public interface AtomicEventTrigger extends EventTrigger {
    void fire();
}

EventRegistrar er = Jpmf.getEventRegistrar();
AtomicEventTrigger trigger = er.createEventTrigger(
    AtomicEventTrigger.class, "class=Foo,event=bar");
trigger.fire();
```
Basic concepts: event delegates

Event delegates

- Callback references used by event sources to notify framework about events
  - Assigned to event sources by framework

- Collects generic performance data if configured so
  - Uses a preconfigured measurement context associated with a particular event group

- Submits event-specific and generic performance data for storage in a preallocated memory buffer
  - Policies for handling out-of-space situations (DROP, OVERWRITE, EXPAND)
Data delivery & management

Data delivery

- Periodically (or on demand) collects data from in-memory buffers and sends them over network or writes them to a file in a generic format
- Can (potentially) perform in-place aggregation of data to reduce the amount of data to store/send over network
  - Far future, still needs to be thought out to fit in

Management

- Event source registration and trigger creation (local)
- Browsing, controlling, and configuring events (remote)
  - i.e. assigning additional performance data to be collected on event occurrence
Framework architecture overview

Performance Measurement Framework

Management
- Event Source Registration
- Experiment Configuration

Data Delivery
- Public interfaces
- Transport Methods

Event Sources
- External
- Internal

Event Processing
- Performance Data Access
- Data Sampling
- Data Buffering

Aggregation
Performance data access

Lower-level subsystem to enable generic access to...

- Time sources
  - System.nanoTime(), simple global counter
  - native CPU-based counters (HPET, TSC), hrtime(), ...

- (Performance) data sources
  - Operating system specific performance data access, e.g. Windows performance objects, Linux /proc & /sys & syscalls, Solaris kstat
  - Potentially also Java-specific and application's own data
  - Performance counters/gauges represented as sensors
Basic concepts: sensors & sensor instances

Sensor

- generic representation of performance data
  - amount of free memory in the system, number of packets sent over interface, number of interrupts in the system, processor idle time, ...
- describes the values it provides
  - kind (gauge/counter), data type (int/long/...)
- multiple occurrences of a sensor in the system = instances
  - three sensor kinds: singleton, static, dynamic

Public interface

- URI-like names (sensor:linux.procfs/system.stat/cpu.nice#0)
- sensor descriptor interface
public abstract class Sensor implements Identifiable {
    public final String id ();

    public final SensorKind sensorKind ();
    public final ValueType valueType ();
    public final ValueKind valueKind ();

    public final Probe probe ();

    ...
    public final int instanceCount ();
    public final Iterable <String> instances ();
    public final boolean hasInstance (final String instanceId);
    ...
    // plus inheritance interface towards subclasses
}

**Basic concepts: probes & data sources**

**Probe**
- internal entity responsible for individual sensors
- uses Java or binds to native code to obtain sensor values
  - must provide probe specific context for requested sensors, see measurement context

**Data source**
- aggregates similar probes into a data source
- manages data source name space for sensors
  - must be able to find internal sensor representation for the given name
- plugged into system through Java ServiceLoader, looking for classes implementing DataSourceProvider interface
Measurement context

- Represents a set of sensor instances to get readings from
  - aggregates contexts from probes responsible for sensors

- Decouples configuration from measurement
  - allows to avoid disruptive operations (e.g. memory allocation, opening files) during measurement and data access

- Decouples data sampling from data decoding phase
  - allows to get snapshot of raw sensor values in performance event context and decode them later, asynchronously

- Data provided in generic way through value handles (references to elements of shared arrays)
  - value handles can be obtained prior to measurement, which can speed up value extraction after decoding
public interface MeasurementContext {
    MeasurementResult prepare ();
    MeasurementResult sample ();
    void decode ();
    MeasurementResult update ();

    int readingCount ();
    int readingIndexFor (String sensor);
    SensorReading [] readings ();
    SensorReading readingAt (int index);
    SensorReading readingFor (String sensor);

    long samplingStartTime ();
    long samplingFinishTime ();

    MeasurementContext duplicate ();
    void destroy ();
}
Controller ctl = Jpda.getController ();
MeasurementContext mc = ctl.createMeasurementContext (  
    // multiple-instance sensors  
    "linux.procfs/system.stat/cpu.user",
    "linux.procfs/system.stat/cpu.nice",
    "linux.procfs/system.stat/intr",

    // singleton sensors  
    "linux.procfs/system.stat/procs_blocked"
);

SensorReading niceReading = mc.readingFor ("linux.procfs/system.stat/cpu.nice");
ValueHandle cpu0Nice = niceReading.valueHandleFor ("0");
mc.update ();
System.out.println (cpu0Nice.longValue ());
JPMF status

Basic management
- registering event sources and event triggers works
- event delegates and triggers for basic event types work
  - i.e. deliver events from application to the framework
- additional event types may be needed for collecting loop counts (me)

Application instrumentation
- automatic instrumentation for Fractal
- Itemis showcase instrumented manually using triggers
- work in progress on automatic byte code instrumentation (Lukáš)
Measurement configuration and control

- support for file-based framework configuration
  - XML file with properties, JMX ObjectName patterns
- event sources and events can be enabled/disabled
  - JConsole or a custom JMX-based command line utility
- association of sensors with events not yet supported (me)
  - only event specific data collected, such as time stamps
JPMF status

Measurement configuration and control:

- support for file-based framework configuration
  - XML file with properties, JMX ObjectName patterns
- event sources and events can be enabled/disabled
  - JConsole or a custom JMX-based command line utility
- association of sensors with events not yet supported (me)
  - only event specific data collected

```xml
<source name="*">
  <property name="timeSource">java.nano</property>
  <property name="enableJMX">true</property>
  <property name="enabled">false</property>
</source>

<event name="*">
  <!-- event controller -->
  <property name="enabled">true</property>

  <!-- buffer -->
  <property name="capacity">1000</property>
  <property name="overflowPolicy">DROP</property>
  <property name="skipCount">0</property>

  <!-- measurement context pool -->
  <property name="threadCount">1</property>

  <!-- measurement context -->
  <property name="sensors">
    linux.procfs/cpu.stat/user#0
    linux.procfs/cpu.stat/system#0
    linux.procfs/net/bytes_sent#eth0
  </property>
</event>

<transport name="default">
  <property name="outputFile">/dev/null</property>
  <property name="events">
    class=* 
  </property>
</transport>
```
JPMF status

Storage and transport

- event specific data stored to memory buffers
- work in progress on storing data to disk, including generic sensor data (Petko)
  - will need management support for creating and configuring transport entities when available (me)
## JPDA status

### Management
- loads and initializes time and data sources
- browsing of available sensors not yet supported (me)

### Measurement and data access
- works – data get from probes to sensor readings
- measurement context needs cloning support to enable construction of measurement context pools for multi-threaded operation (me)

### Data sources
- currently only 1 test data source with 1 probe (/proc/stat)
  - Windows Performance Objects support not yet merged
  - support for more sensors as needed (...)

Conclusion

It's not vaporware :-)  
- it's just quite complex and the progress is slow  
- ~21k non-empty text lines, ~10k physical LOC

Populating Itemis showcase model with data  
- necessary functionality expected to be available by end of June 2009  
  (December 2009)  
  end of next week (hopefully... Petko?)