Parfait

How it works + where to from here

(AKA "Through a glass, clearly")

Paul Cowan
Aconex

September 2010

Parfait

History of Parfait

- Originally developed as aconex-pcp-bridge
- Specifically for getting PCP values into a custom agent
- Expanded
- Improved
- Rewritten for MMV agent
- Open-sourced

The Basics

- Parfait has 3 main parts (for now):
  - Monitoring
  - DXM
  - Timing
  - Requests

Monitoring

- This is the ‘original’ PCP bridge metrics (heavily modified)
- Simple Java objects (MonitoredValues) which wrap a value (e.g. AtomicLong, String)
- MonitoredValues register themselves with a registry (container)

Monitoring

- When a value changes, a number of observers get told, and can output accordingly
  - PCP
  - JMX
  - Other?
- Very simple to use
- ‘Default registry’ (legacy concept)

Monitoring

- Also worth pointing out: PollingMonitoredValue
- This is used when the value is updated by something we don’t control, can’t ‘subscribe’
- Thread checks periodically for new value, updates Registry if changed

DXM

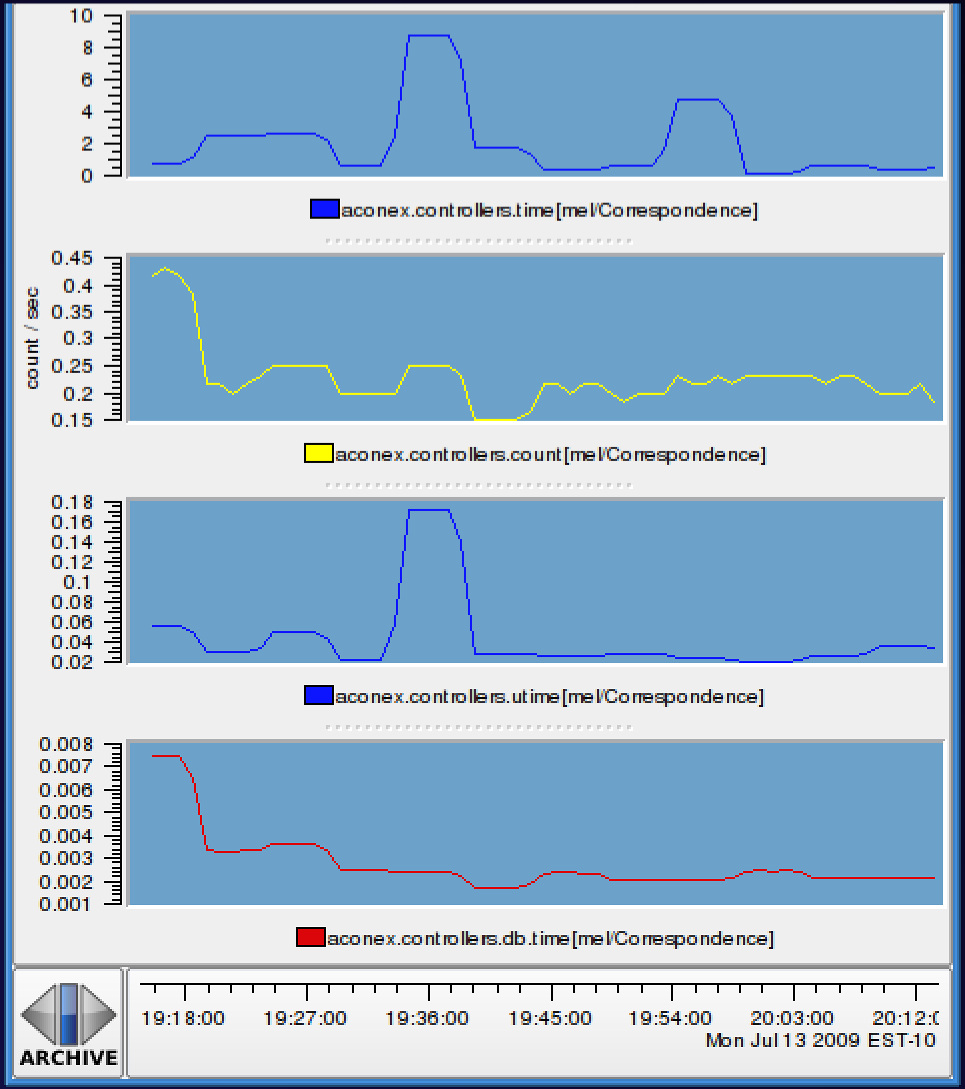
- This is the PCP output side of aconex-pcp-bridge
- Rewritten to use the new non-custom MMV PMDA
- Advantages:
  - flexible, standardised, less maintenance work
- Disadvantages
  - have to assign ID to each metric

DXM

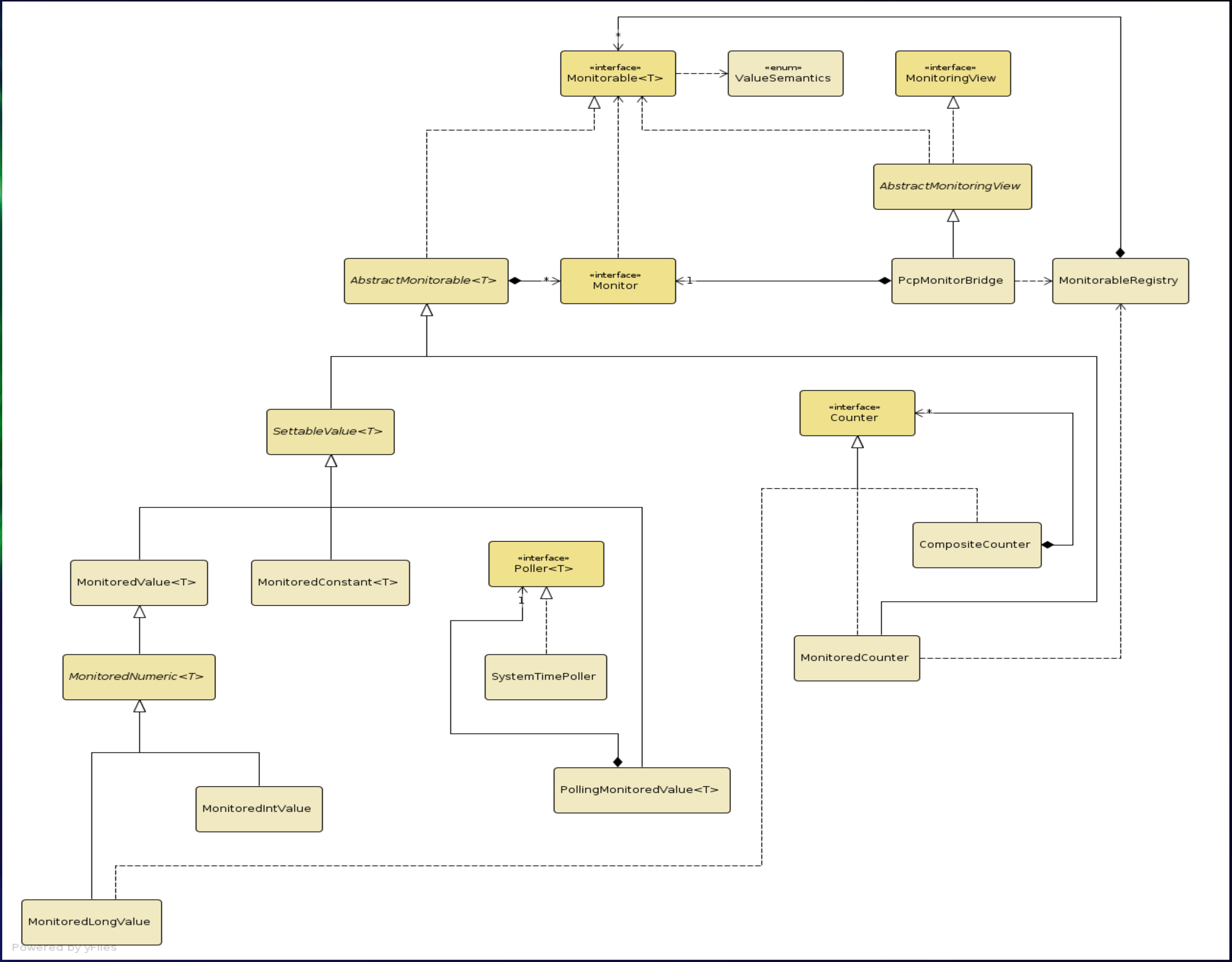
- Map metrics names to ‘pseudo-PCP’ names, e.g.:
  - aconex.controllers.time.blah → aconex.controllers[mel/blah].time
- Placement of brackets is significant (determines PCP domains)

Monitoring + DXM

(Pretty graph time!)



How it hangs together



Timing

- Logs the resources consumed by a *request* (an individual user action)
- Relies on a single request being thread-bound (and threads being used exclusively)

Timing

- Basically need a Map<Thread, Value>
- Take the value for a Thread at the start, and at the end
- Delta is the ‘cost’ of that request

Timing

- Deltas can be output a number of ways:
  - Normal metrics
    - Per ‘event’
    - Total
  - Logs (Log4jSink)
  - HBase (HBaseSink) – in progress!

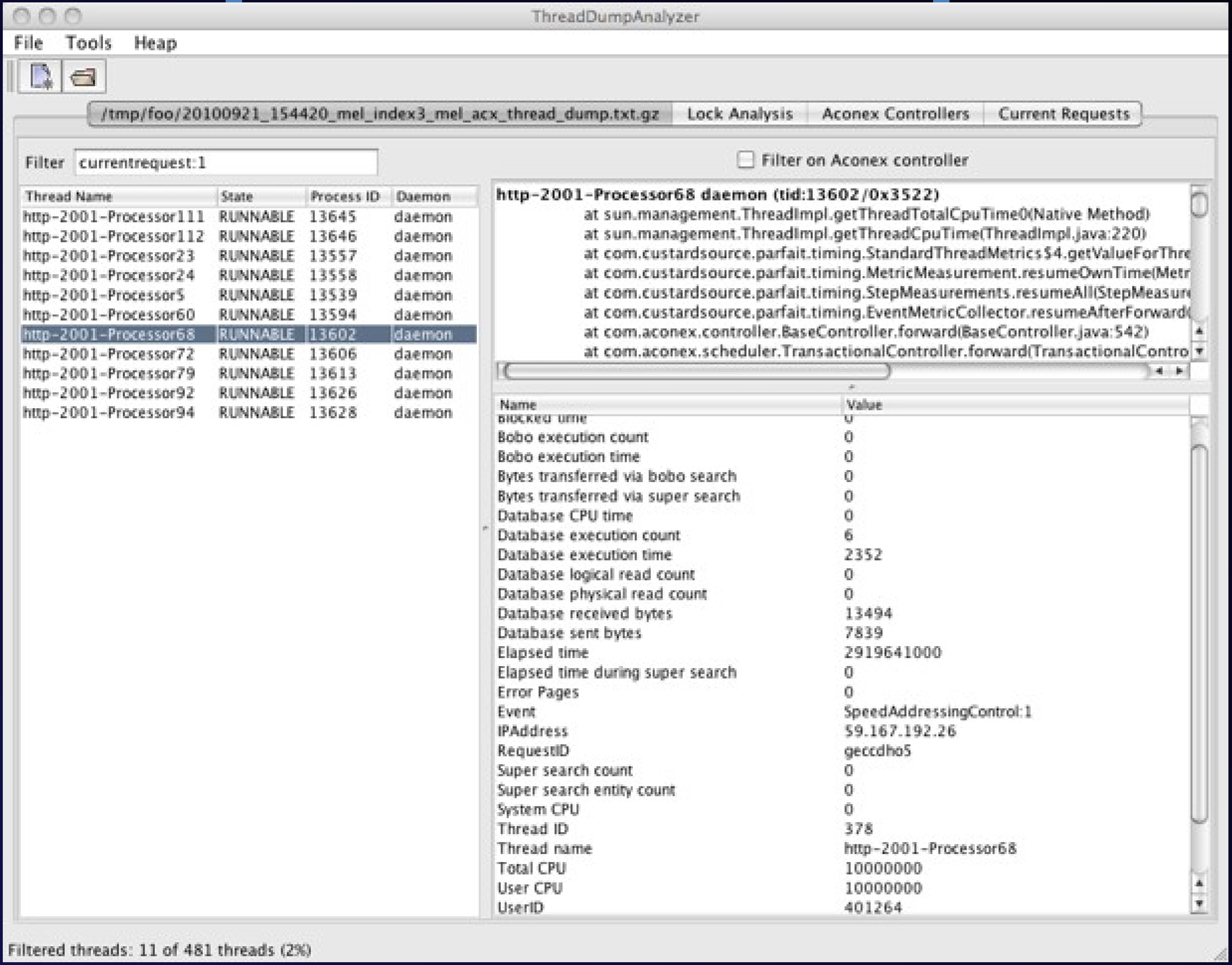
Timing: Example

```
[2010-09-22 15:02:13,466 INFO ][ait.timing.Log4jSink][http-
8080-Processor3 gedq93kl][192.168.7.132][20][] Top
taskssummaryfeatures:tasks    taskssummaryfeatures:tasks
      Elapsed time: own 380.146316 ms, total 380.14688
ms  Total CPU: own 150.0 ms, total 150.0 ms      User CPU: own
140.0 ms, total 140.0 ms  System CPU: own 10.0 ms, total 10.0
ms  Blocked count: own 40, total 40      Blocked time: own 22 ms,
total 22 ms   Wait count: own 2, total 2  Wait time: own 8 ms,
total 8 ms     Database execution time: own 57 ms, total 57 ms
      Database execution count: own 11, total 11
Database logical read count: own 0, total 0  Database physical
read count: own 0, total 0  Database CPU time: own 0 ms, total 0
ms  Database received bytes: own 26188 By, total 26188
By  Database sent bytes: own 24868 By, total 24868 By  Error
Pages: own 0, total 0  Bobo execution time: own 40.742124
ms, total 40.742124 ms      Bobo execution count: own 2, total
2     Bytes transferred via bobo search: own 0 By, total 0 By
Super search entity count: own 0, total 0      Super search count:
own 0, total 0      Bytes transferred via super search: own 0 By,
total 0 By     Elapsed time during super search: own 0 ms, total 0
ms
```

Requests

- As well as snapshotting requests after completion, for many metrics we can see meaningful ‘in-progress’ values
- Simple JMX bean which ‘walks’ in-progress requests
- Tie in with ThreadContext (MDC abstraction)
  - Include UserID
  - ThreadID

Requests - Example


ThreadDumpAnalyzer
File
Tools
Heap
/tmp/foo/20100921_154420_mel_index3_mel_acx_thread_dump.txt.gz
Lock Analysis
Aconex Controllers
Current Requests
Filter on Aconex controller

Filter currentrequest:1

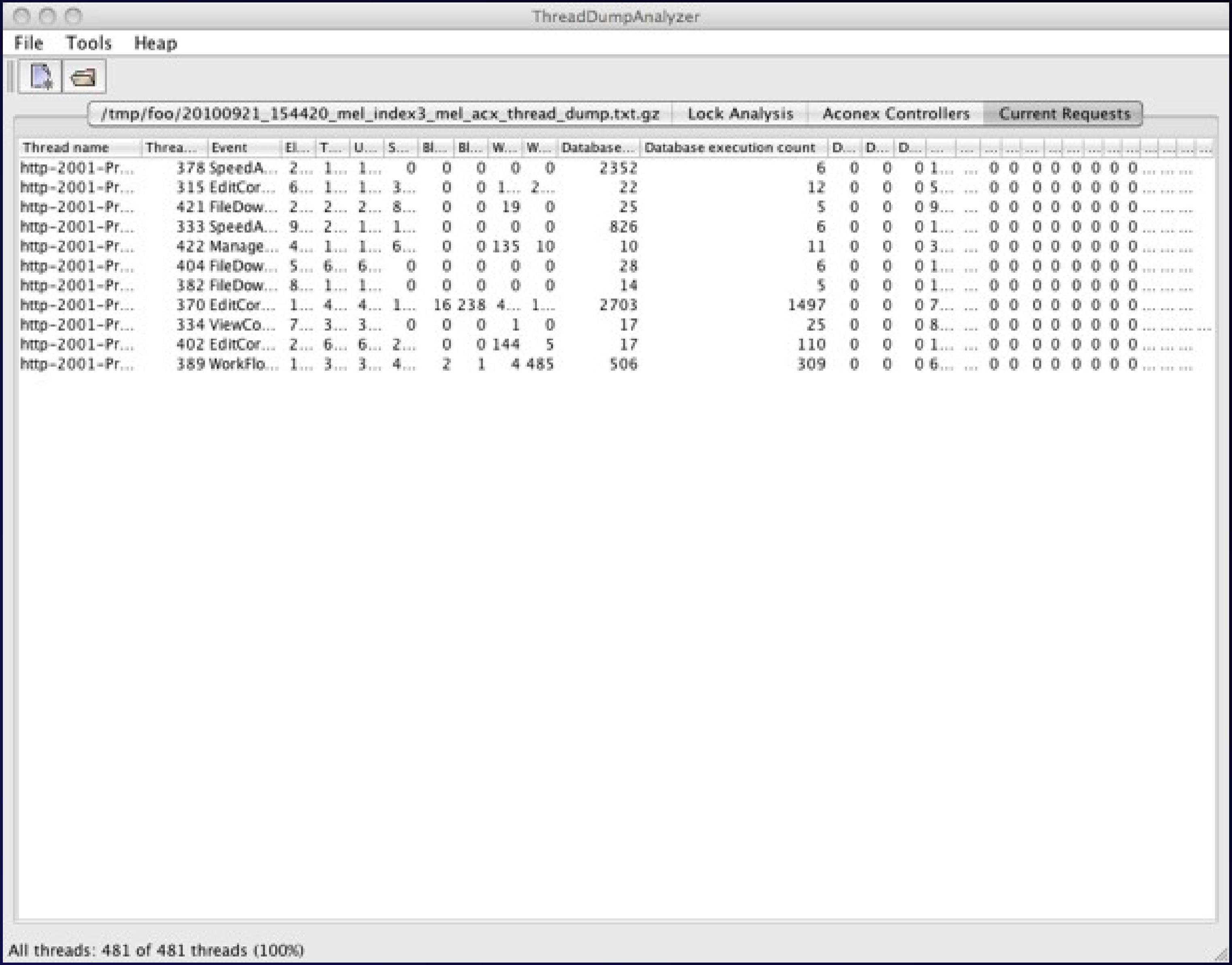
Thread Name	State	Process ID	Daemon
http-2001-Processor111	RUNNABLE	13645	daemon
http-2001-Processor112	RUNNABLE	13646	daemon
http-2001-Processor23	RUNNABLE	13557	daemon
http-2001-Processor24	RUNNABLE	13558	daemon
http-2001-Processor5	RUNNABLE	13539	daemon
http-2001-Processor60	RUNNABLE	13594	daemon
http-2001-Processor68	RUNNABLE	13602	daemon
http-2001-Processor72	RUNNABLE	13606	daemon
http-2001-Processor79	RUNNABLE	13613	daemon
http-2001-Processor92	RUNNABLE	13626	daemon
http-2001-Processor94	RUNNABLE	13628	daemon

```
http-2001-Processor68 daemon (tid:13602/0x3522)
        at sun.management.ThreadImpl.getThreadTotalCpuTime0(Native Method)
        at sun.management.ThreadImpl.getThreadCpuTime(ThreadImpl.java:220)
        at com.custardsource.parfait.timing.StandardThreadMetrics$4.getValueForThre
        at com.custardsource.parfait.timing.MetricMeasurement.resumeOwnTime(Metr
        at com.custardsource.parfait.timing.StepMeasurements.resumeAll(StepMeasure
        at com.custardsource.parfait.timing.EventMetricCollector.resumeAfterForward(
        at com.aconex.controller.BaseController.forward(BaseController.java:542)
        at com.aconex.scheduler.TransactionalController.forward(TransactionalContro
```

Name	Value
Blocked time	0
Bobo execution count	0
Bobo execution time	0
Bytes transferred via bobo search	0
Bytes transferred via super search	0
Database CPU time	0
Database execution count	6
Database execution time	2352
Database logical read count	0
Database physical read count	0
Database received bytes	13494
Database sent bytes	7839
Elapsed time	2919641000
Elapsed time during super search	0
Error Pages	0
Event	SpeedAddressingControl:1
IPAddress	59.167.192.26
RequestID	geccdho5
Super search count	0
Super search entity count	0
System CPU	0
Thread ID	378
Thread name	http-2001-Processor68
Total CPU	10000000
User CPU	10000000
UserID	401264

Filtered threads: 11 of 481 threads (2%)

Requests - Example



ThreadDumpAnalyzer

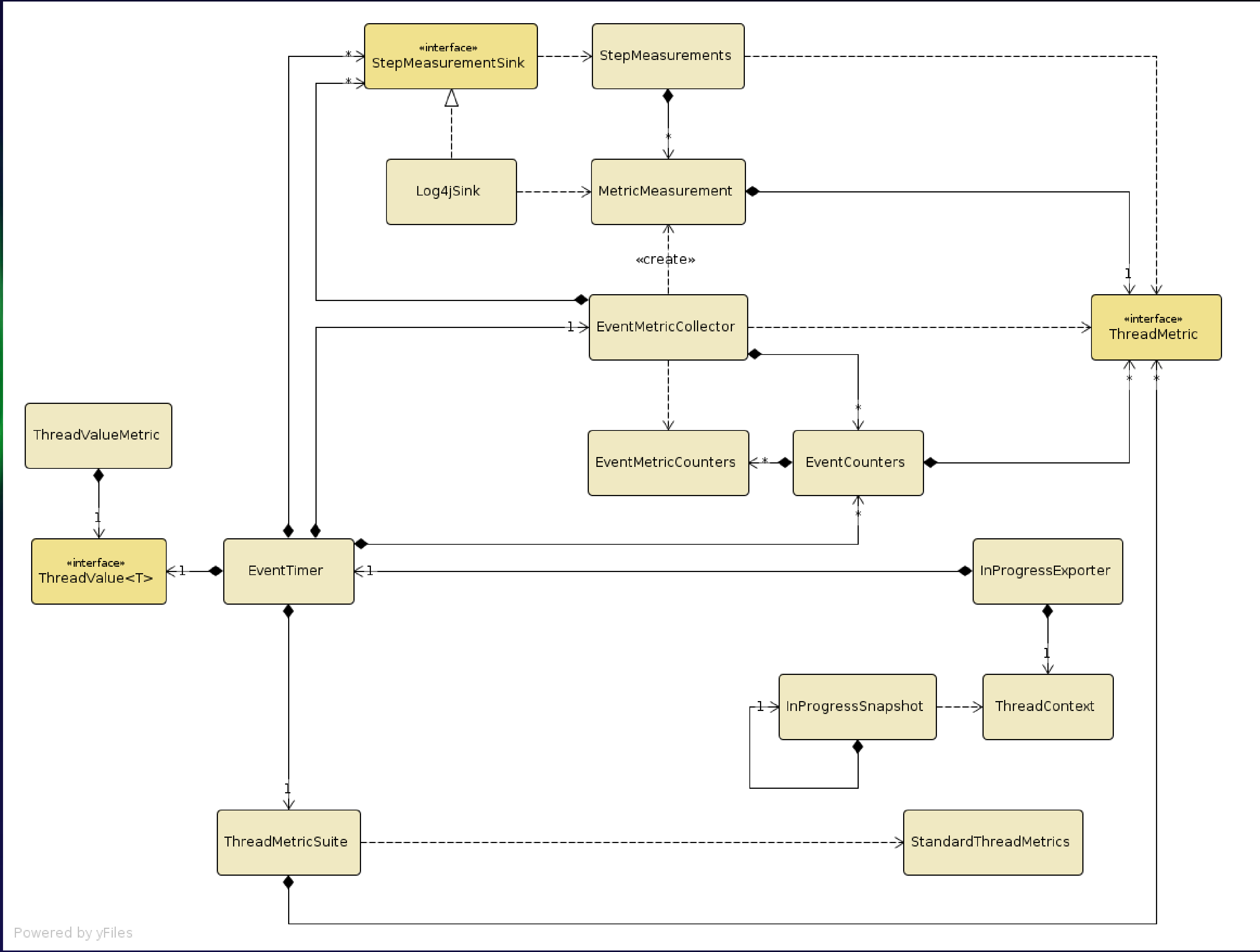
File Tools Heap

/tmp/foo/20100921_154420_mel_index3_mel_acx_thread_dump.txt.gz | Lock Analysis | Aconex Controllers | Current Requests

Thread name	Threa...	Event	El...	T...	U...	S...	Bl...	Bl...	W...	W...	Database...	Database execution count	D...	D...	D...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
http-2001-Pr...	378	SpeedA...	2...	1...	1...	0	0	0	0	0	2352	6	0	0	0	1...	...	0	0	0	0	0	0	0	0	...	...	...	
http-2001-Pr...	315	EditCor...	6...	1...	1...	3...	0	0	1...	2...	22	12	0	0	0	5...	...	0	0	0	0	0	0	0	0	...	...	...	
http-2001-Pr...	421	FileDow...	2...	2...	2...	8...	0	0	19	0	25	5	0	0	0	9...	...	0	0	0	0	0	0	0	0	...	...	...	
http-2001-Pr...	333	SpeedA...	9...	2...	1...	1...	0	0	0	0	826	6	0	0	0	1...	...	0	0	0	0	0	0	0	0	...	...	...	
http-2001-Pr...	422	Manage...	4...	1...	1...	6...	0	0	135	10	10	11	0	0	0	3...	...	0	0	0	0	0	0	0	0	...	...	...	
http-2001-Pr...	404	FileDow...	5...	6...	6...	0	0	0	0	0	28	6	0	0	0	1...	...	0	0	0	0	0	0	0	0	...	...	...	
http-2001-Pr...	382	FileDow...	8...	1...	1...	0	0	0	0	0	14	5	0	0	0	1...	...	0	0	0	0	0	0	0	0	...	...	...	
http-2001-Pr...	370	EditCor...	1...	4...	4...	1...	16	238	4...	1...	2703	1497	0	0	0	7...	...	0	0	0	0	0	0	0	0	...	...	...	
http-2001-Pr...	334	ViewCo...	7...	3...	3...	0	0	0	1	0	17	25	0	0	0	8...	...	0	0	0	0	0	0	0	0	...	...	...	...
http-2001-Pr...	402	EditCor...	2...	6...	6...	2...	0	0	144	5	17	110	0	0	0	1...	...	0	0	0	0	0	0	0	0	...	...	...	
http-2001-Pr...	389	WorkFlo...	1...	3...	3...	4...	2	1	4	485	506	309	0	0	0	6...	...	0	0	0	0	0	0	0	0	...	...	...	

All threads: 481 of 481 threads (100%)

How it hangs together



Where do we use it?

- Instrument the app itself (business actions) with metrics
- Instrument third-party libraries (notably JDBC driver) for metrics/timings
- Generate timings for inter-process events (supersearch, bobo)

How to use: Metrics

- Adding a new metric is trivial:

```
public class FileIndexer {
 private final MonitoredLongValue done =
   new MonitoredLongValue(
       “aconex.indexes.time”,
       “Time spend indexing”,
       MonitorableRegistry.DEFAULT_REGISTRY,
            // injection = better!
       0L,  // initial value
       SI.NANO(SI.SECOND));
```

Add a line to pcp-metric-ids.txt:

```
   aconex.indexes.time  670
```

And use it!

```
   done.inc(timeSpent);
```

How to use: Timing

- Adding a new measurement needs a new ThreadMetric
- Easiest to use ThreadCounter (glorified ThreadLocal) and ThreadValueMetric:

```
public class CoolThing {
 public final ThreadCounter coolThingsDone =
   new ThreadMapCounter();

 public void doCoolThing(...) {
   coolThingsDone.inc();
 }
}
```

How to use: Timing

- Then just add it to the ThreadMetricSuite
- e.g. AconexMetricSuite

-
  ```
controllerSuite.addMetric(
   new ThreadValueMetric("Cool things",
     Unit.ONE, "things.done.cool",
     "The number of cool things done",
     coolThing.getCoolThingsDone());
);
```

Will automatically appear for all controllers

The Magic Sauce

- EventTimer has a bunch of metrics, PCP prefix, etc
- Wired together by SpringEventTimerInjector:
  - Finds all Spring beans which use an interface
  - Tells the EventTimer about them
  - Injects the timer into the bean
  - Bean can now start/stop timing, with a ‘tag’
  -

Where to?
(what are the grand plans?)

- Timing becoming ‘1st-class’ citizen
- Multi-thread support
- Outputs: JMX (++), Hbase, RabbitMQ (Rocksteady)?
- Inputs: AOP, Hibernate
- Distributed (à la Dapper)?

Staying Involved

- Project uses Mercurial now (easy to branch/contribute)
- Releases happen to central (much simpler to manage)
- Adding others (psmith?) to repo uploaders
- Follow the Google Code project!
  - Mailing lists: -user and -dev
  - Watch commits
- Use, contribute, keep in touch!