Kieker 1.5

Application Performance Monitoring and Dynamic Software Analysis

Kieker Meeting

Jan Waller — 16.02.2012

Overview of some new Features

Finished Features:

#317 Kieker Logging / Remove commons.logging

#319 Registry to assign unique IDs to Object

#323 Binary FS Writer and Reader

#329 PrintStream Writer

#335 Static factory for records

#409 New Record types for Event based monitoring

Work in Progress:

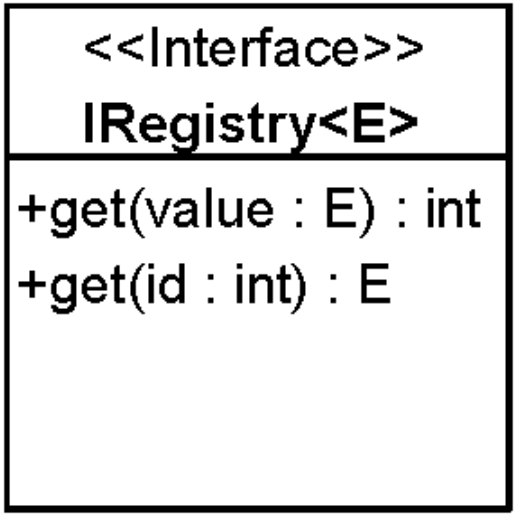
#330 Pipe&Filter Framework

Kieker Logging

- Removed the dependency on commons.logging
- Provides a simple interface for logging
- Four log levels
  - debug, info, warn, error
- Actual logging is handled by libraries
  - JDK1.4 logging (default)
  - commons.logging (if present)
  - easy to add support for other logging libraries

Registry to Assign Unique IDs to Objects

- Simplified and specialized **ConcurrentHashMap**
- Only one method to access
  - if value not yet present
    - assign unique id (int)
    - add to internal hashmap (<String, int>)
    - [new HashRecord(id, value)]
  - return unqiue id



- Highly concurrent access is possible
- Currently used to cache Strings for Writers

Classes: Registry.java, IRegistry.java, RegistryController.java, IRegistryController.java, HashRecord.java

Binary FS Writer and Reader

- Simple comparison:

	AsyncFSWriter	AsyncBinaryFSWriter
kieker.map	275 bytes	692 bytes
kieker-….(dat/bin)	2.385 bytes	1.442 bytes
total	2.660 bytes	2.134 bytes

```
$0=kieker.common.record.flow.Trace
$1=kieker.common.record.flow.BeforeOperationEvent
$2=kieker.common.record.flow.CallOperationEvent
$3=kieker.common.record.flow.AfterOperationEvent
$4=kieker.common.record.flow.ConstructionEvent
$5=kieker.common.record.flow.SplitEvent
```

```
$0=kieker.common.record.flow.Trace
$1=kieker.common.record.flow.BeforeOperationEvent
$2=public static void kiekerFlow.TestFlow.main(java.lang.String[])
$3=kieker.common.record.flow.CallOperationEvent
$4=public kiekerFlow.TestFlow()
$5=kieker.common.record.flow.AfterOperationEvent
$6=kieker.common.record.flow.ConstructionEvent
$7=kiekerFlow.TestFlow
$8=Thread[Thread-3,5,main]
$9=kieker.common.record.flow.SplitEvent
$10=public synchronized void kiekerFlow.TestFlow.start()
$11=Thread[Thread-4,5,main]
$12=public void kiekerFlow.TestFlow.test()
$13=public void kiekerFlow.TestFlow.setA(int)
$14=public int kiekerFlow.TestFlow.getA()
$15=public void kiekerFlow.TestFlow.run()
```

- Uses Registry
- More efficient writer (?)
- Smaller logs
- Less fault tolerant (!)

Classes: AsyncBinaryFsWriter.java, BinaryFsWriterThread.java

PrintStream Writer

- A simple **writer** to STDOUT or STDERR
- Useful in *debugging* or *development*
- Example output:

```
Trace: 31;0;1;0;-1
BeforeOperationEvent: 48;48;0;0;public static void kiekerFlow.TestFlow.main(java.lang.String[])
CallOperationEvent: 48;48;0;1;public static void kiekerFlow.TestFlow.main(java.lang.String[]);public kiekerFlow.TestFlow()
BeforeOperationEvent: 48;48;0;2;public kiekerFlow.TestFlow()
AfterOperationEvent: 48;48;0;3;public kiekerFlow.TestFlow()
ConstructionEvent: 48;48;kiekerFlow.TestFlow;Thread[Thread-2,5,main]
SplitEvent: 64;64;0;4
CallOperationEvent: 64;64;0;5;public static void kiekerFlow.TestFlow.main(java.lang.String[]);public synchronized void kiekerFlow.TestFlow.start()
CallOperationEvent: 64;64;0;6;public static void kiekerFlow.TestFlow.main(java.lang.String[]);public kiekerFlow.TestFlow()
BeforeOperationEvent: 64;64;0;7;public kiekerFlow.TestFlow()
AfterOperationEvent: 64;64;0;8;public kiekerFlow.TestFlow()
ConstructionEvent: 64;64;kiekerFlow.TestFlow;Thread[Thread-3,5,main]
SplitEvent: 64;64;0;9
CallOperationEvent: 64;64;0;10;public static void kiekerFlow.TestFlow.main(java.lang.String[]);public synchronized void kiekerFlow.TestFlow.start()
CallOperationEvent: 64;64;0;11;public static void kiekerFlow.TestFlow.main(java.lang.String[]);public void kiekerFlow.TestFlow.test()
BeforeOperationEvent: 64;64;0;12;public void kiekerFlow.TestFlow.test()
AfterOperationEvent: 64;64;0;13;public void kiekerFlow.TestFlow.test()
CallOperationEvent: 64;64;0;14;public static void kiekerFlow.TestFlow.main(java.lang.String[]);public void kiekerFlow.TestFlow.setA(int)
```

Classes: PrintStreamWriter.java

Static Factory for Records

Motivation:

- Enable MonitoringRecords with final fields

Problem:

- Readers rely on
  - an empty constructor
  - an initFromArray(Object[]) method

Solution:

- Provide static factory methods
  - we can't enforce these methods!
    - ➢ marker Interface, if implemented we assume present

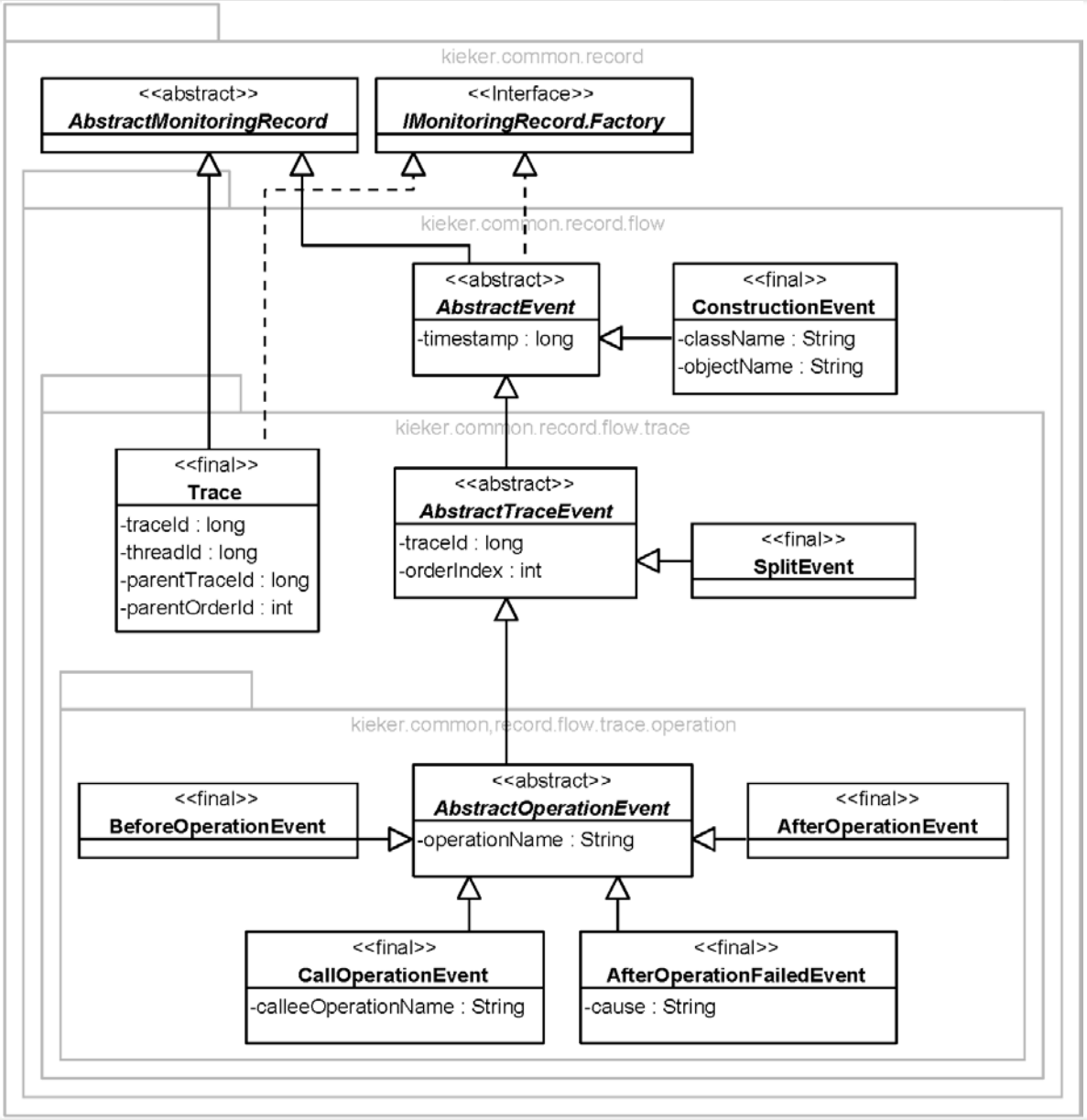
Example (BooleanRecord)


```java
package test;

import kieker.common.record.AbstractMonitoringRecord;

public final class BooleanRecordClassic extends AbstractMonitoringRecord {
    private static final long serialVersionUID = 6117847069615403290L;

    private boolean value;

    public BooleanRecordClassic() {
        // nothing to do
    }

    public BooleanRecordClassic(final boolean value) {
        this.value = value;
    }

    @Override
    public Object[] toArray() {
        return new Object[] { this.value };
    }

    @Override
    public void initFromArray(final Object[] values) {
        AbstractMonitoringRecord.checkArray(values, new Class<?>[] { boolean.class, });
        this.value = (Boolean) values[0];
    }

    @Override
    public Class<?>[] getValueTypes() {
        return new Class<?>[] { boolean.class, };
    }

    public boolean isValue() {
        return this.value;
    }

    public void setValue(final boolean value) {
        this.value = value;
    }
}
```

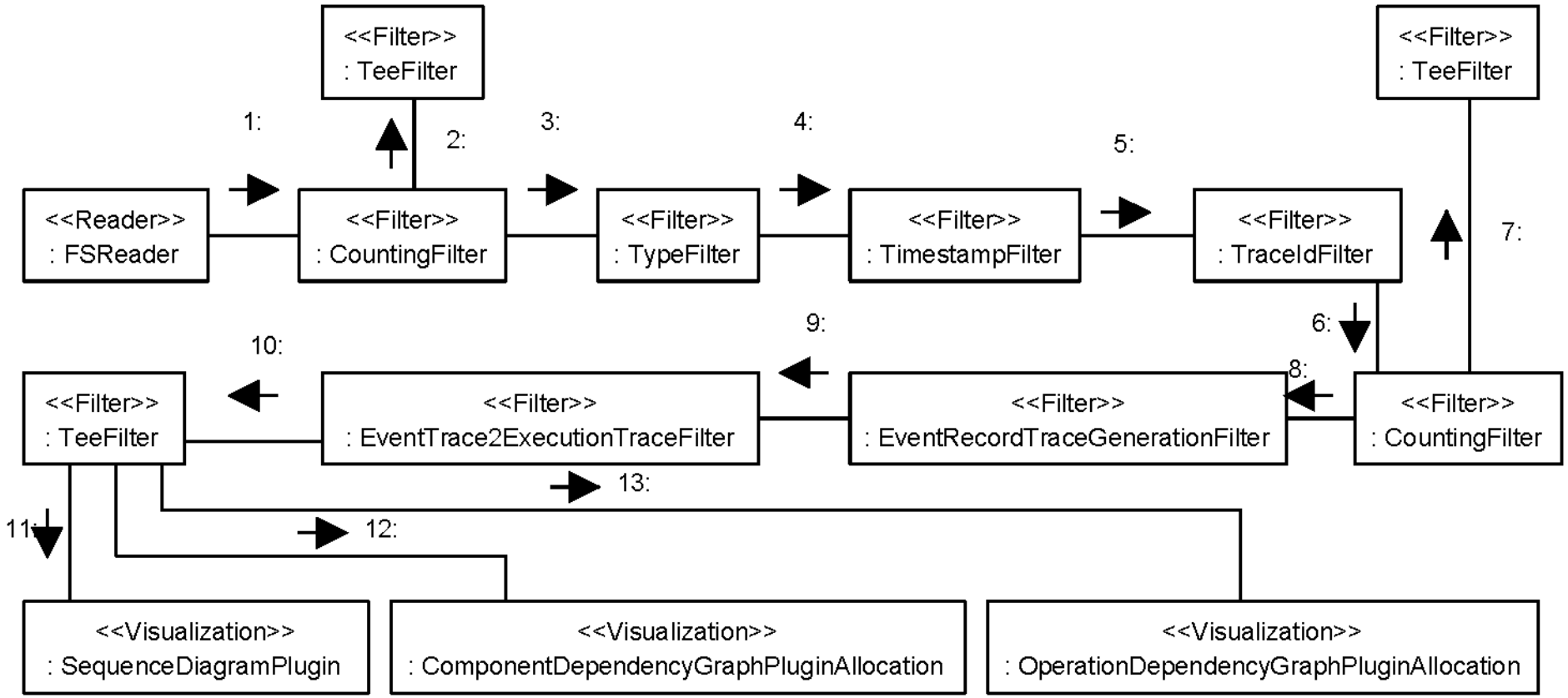
Example (BooleanRecord)


```java
package test;

import kieker.common.record.AbstractMonitoringRecord;
import kieker.common.record.IMonitoringRecord;

public final class BooleanRecordFactory extends AbstractMonitoringRecord implements IMonitoringRecord.Factory {
    private static final long serialVersionUID = -7611090038031964250L;

    private static final Class<?>[] TYPES = new Class<?>[] { boolean.class, };

    private final boolean value;

    public BooleanRecordFactory(final boolean value) {
        this.value = value;
    }

    public BooleanRecordFactory(final Object[] values) {
        AbstractMonitoringRecord.checkArray(values, BooleanRecordFactory.TYPES);
        this.value = (Boolean) values[0];
    }

    @Override
    public Object[] toArray() {
        return new Object[] { this.value };
    }

    @Override
    @Deprecated
    public void initFromArray(final Object[] values) {
        throw new UnsupportedOperationException();
    }

    @Override
    public Class<?>[] getValueTypes() {
        return BooleanRecordFactory.TYPES.clone();
    }

    public boolean isValue() {
        return this.value;
    }
}
```

Event Based Monitoring



Pipe&Filter Framework




Kieker