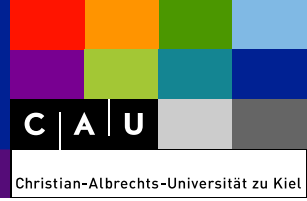


Assembling and Executing Kieker Analysis Configurations via Java API and Web UI

— Kieker Days 2012 —

Nils Christian Ehmke

Software Engineering Group
Kiel University, Germany

November 30, 2012 @ Kiel

Motivation: Pipes-and-Filters Analysis

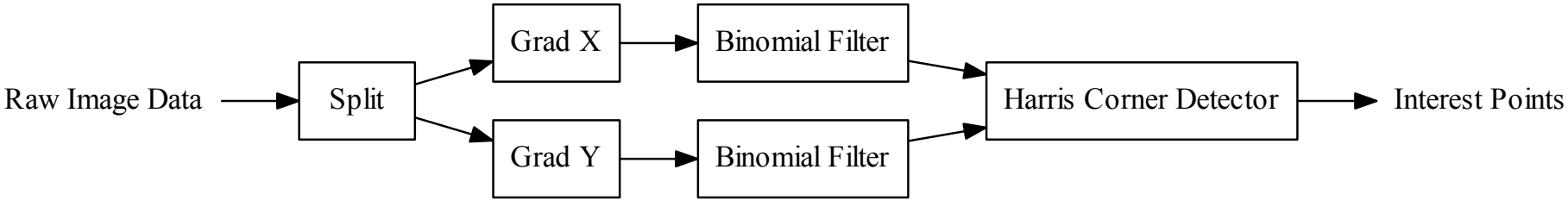


Figure : Interest Point Detection

Motivation: Pipes-and-Filters Analysis

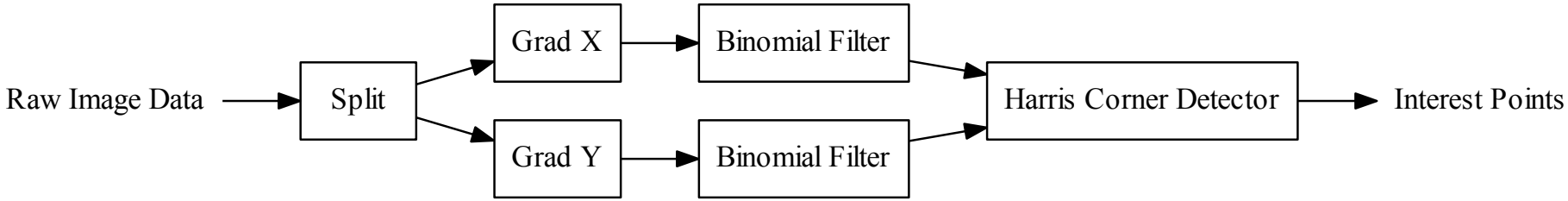


Figure : Interest Point Detection

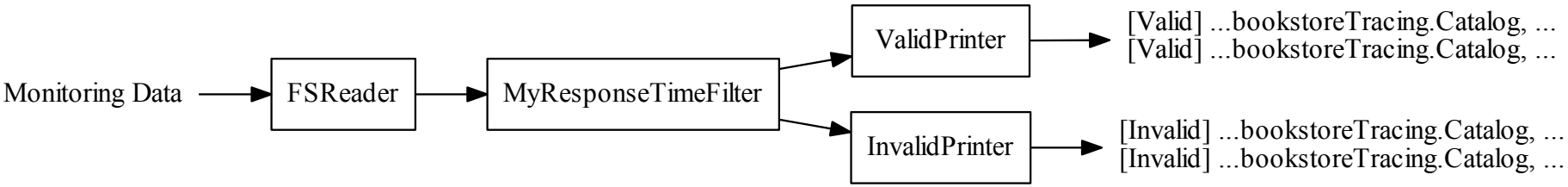


Figure : Kieker: Response time analysis

Question

Why do we need a pipes-and-filters API?

Purpose

Question

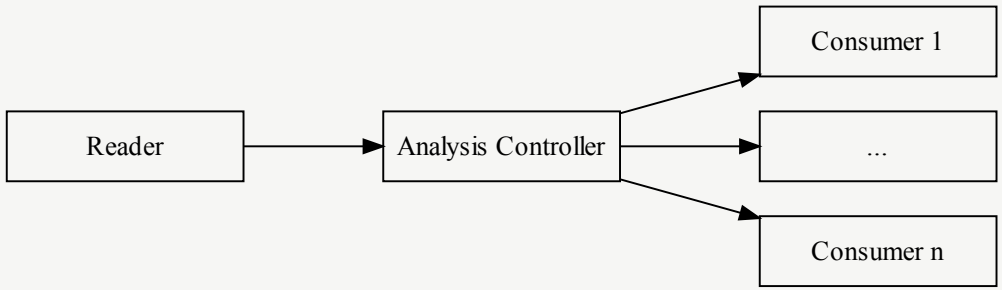
Why do we need a pipes-and-filters API?

Answer

- Assemble, save, load analysis configurations
- Creation of custom analyses
- Reusability of filters

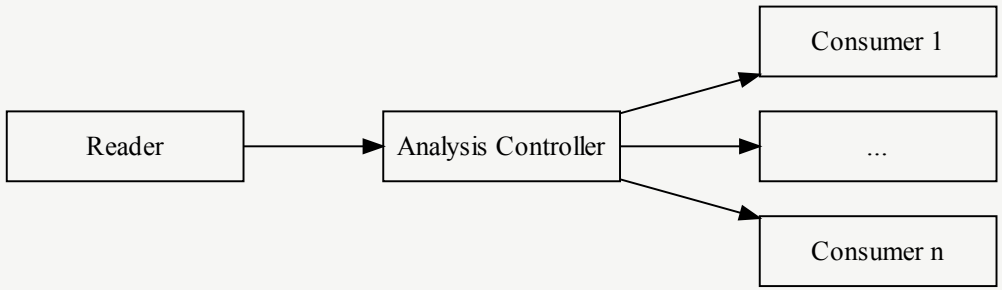
≤ Kieker 1.4

- Pipes and filters possible — very difficult to use
- Only programmatic assembly and execution



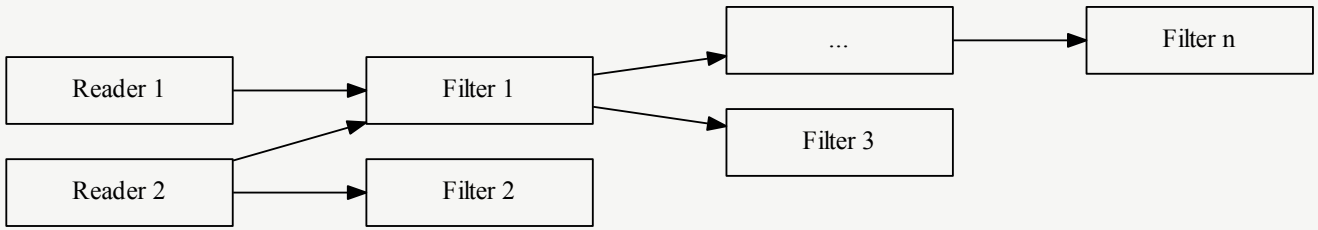
≤ Kieker 1.4

- Pipes and filters possible — very difficult to use
- Only programmatic assembly and execution



≥ Kieker 1.5

- Configurable and easy to use
- Easy extendible pipes and filters



The Meta Model Behind the P&F API

Christian-Albrechts-Universität zu Kiel

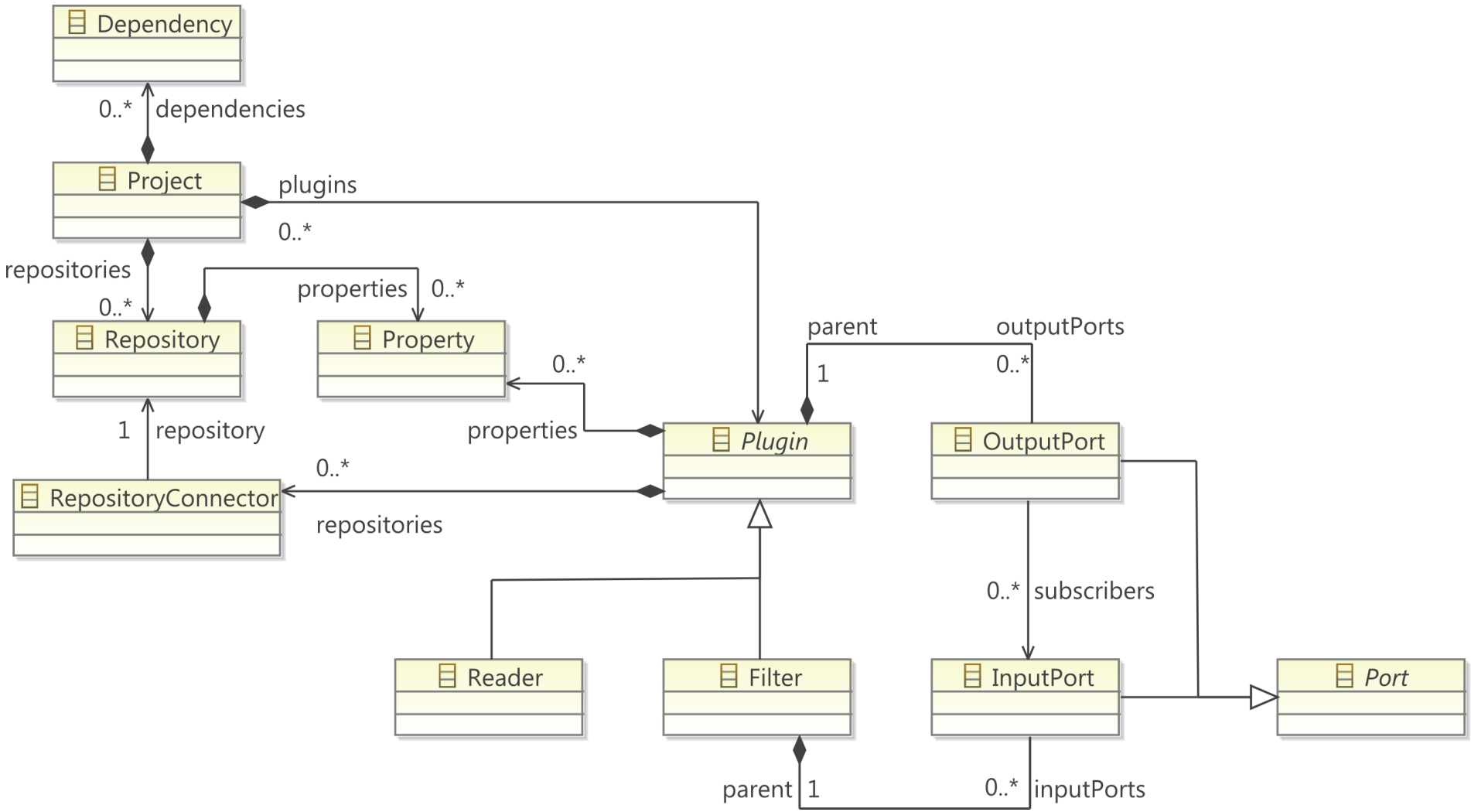


Figure : The Pipes-and-Filters Ecore Model (Simplified)

Question

How to use the pipes and filters API?

Question

How to use the pipes and filters API?

Answer

- Implement and execute in plain Java code
- Persist in kax files
- Implement and execute in a WebGUI
- (Implement own plugins)

P&F Assembly and Execution Method #1


```
// Create an analysis controller
AnalysisController analysisController = new AnalysisController();
```

P&F Assembly and Execution Method #1


```
// Create an analysis controller
AnalysisController analysisController = new AnalysisController();

// Programmatic assembly of pipes-and-filter configuration
Configuration config = new Configuration();
config.setProperty(...);
MyResponseTimePrinter printer = new MyResponseTimePrinter(config);
```

P&F Assembly and Execution Method #1


```
// Create an analysis controller
AnalysisController analysisController = new AnalysisController();

// Programmatic assembly of pipes-and-filter configuration
Configuration config = new Configuration();
config.setProperty(...);
MyResponseTimePrinter printer = new MyResponseTimePrinter(config);

analysisController.registerFilter(printer);
```

P&F Assembly and Execution Method #1


```
// Create an analysis controller
AnalysisController analysisController = new AnalysisController();

// Programmatic assembly of pipes-and-filter configuration
Configuration config = new Configuration();
config.setProperty(...);
MyResponseTimePrinter printer = new MyResponseTimePrinter(config);

analysisController.registerFilter(printer);
analysisController.connect(
   filter, MyResponseTimeFilter.OUTPUT_PORT_NAME_RT_VALID,
   printer, MyResponseTimeOutputPrinter.INPUT_PORT_NAME_EVENTS);
```

P&F Assembly and Execution Method #1


```
// Create an analysis controller
AnalysisController analysisController = new AnalysisController();

// Programmatic assembly of pipes-and-filter configuration
Configuration config = new Configuration();
config.setProperty(...);
MyResponseTimePrinter printer = new MyResponseTimePrinter(config);

analysisController.registerFilter(printer);
analysisController.connect(
   filter, MyResponseTimeFilter.OUTPUT_PORT_NAME_RT_VALID,
   printer, MyResponseTimeOutputPrinter.INPUT_PORT_NAME_EVENTS);

// Starting the analysis
analysisController.run();
```

P&F Assembly and Execution Method #1

Note: We can also save the analysis for later

```
// Save the analysis in a kax file
analysisController.saveToFile(new File("analysis.kax"));
```

The resulting kax file (XMI format)

```
...
<plugins xsi:type="Filter" name="Valid Printer" classname="...">
  <properties name="validOutput" value="true"/>
  <inputPorts name="newEvent"/>
</plugins>
...
<plugins xsi:type="Filter" name="MyResponseTimeFilter"
         classname="...">
  <properties name="thresholdNanos" value="1900000"/>
  <outputPorts name="validResponseTimes"
               subscribers="//@plugins.4/@inputPorts.0"/>
  <outputPorts name="invalidResponseTimes"
               subscribers="//@plugins.3/@inputPorts.0"/>
  <inputPorts name="newResponseTime"/>
</plugins>
```

P&F Assembly and Execution Method #2

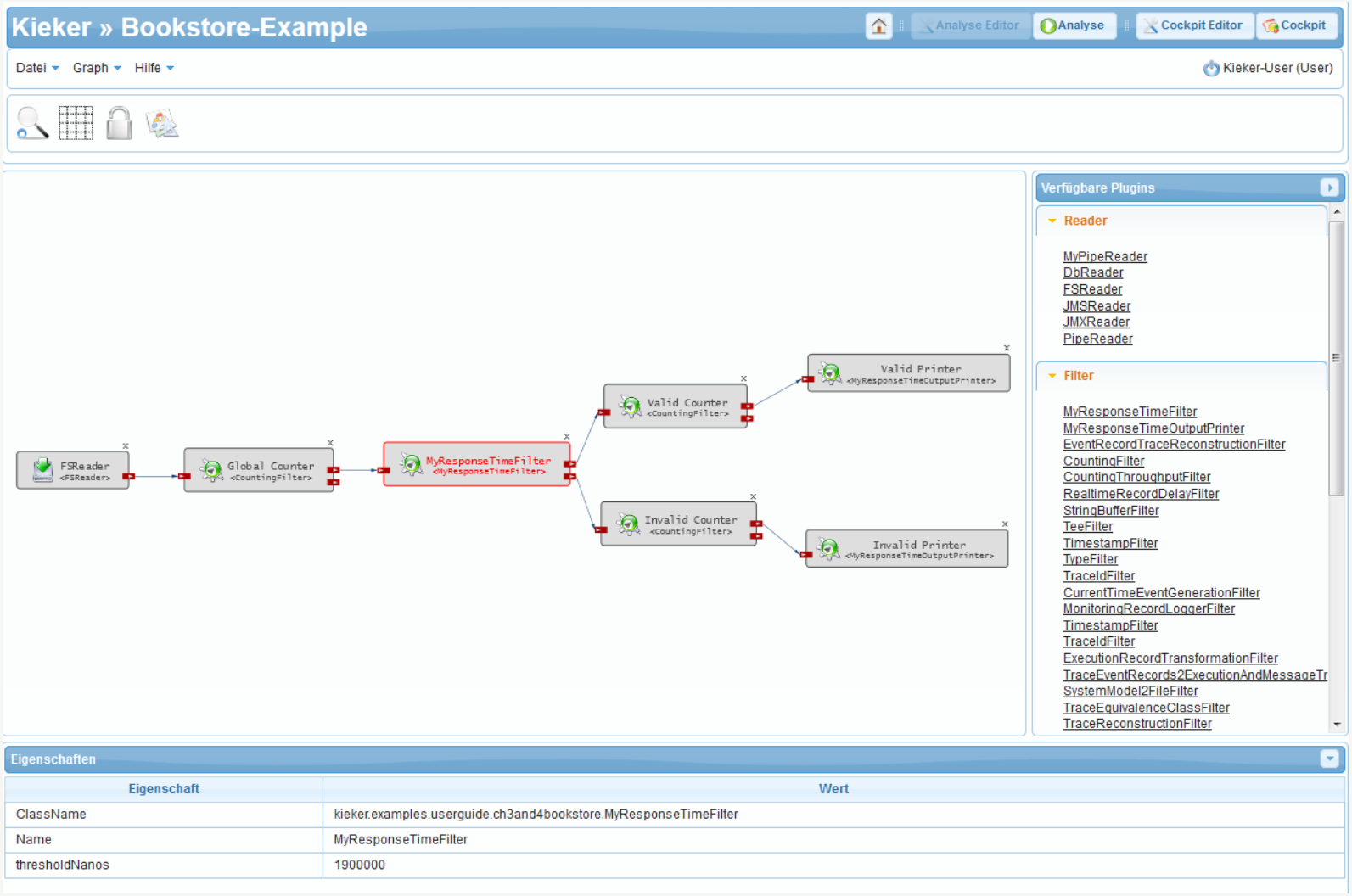
Running the analysis using the Java API

```
new AnalysisController(new File("analysis.kax")).run();
```

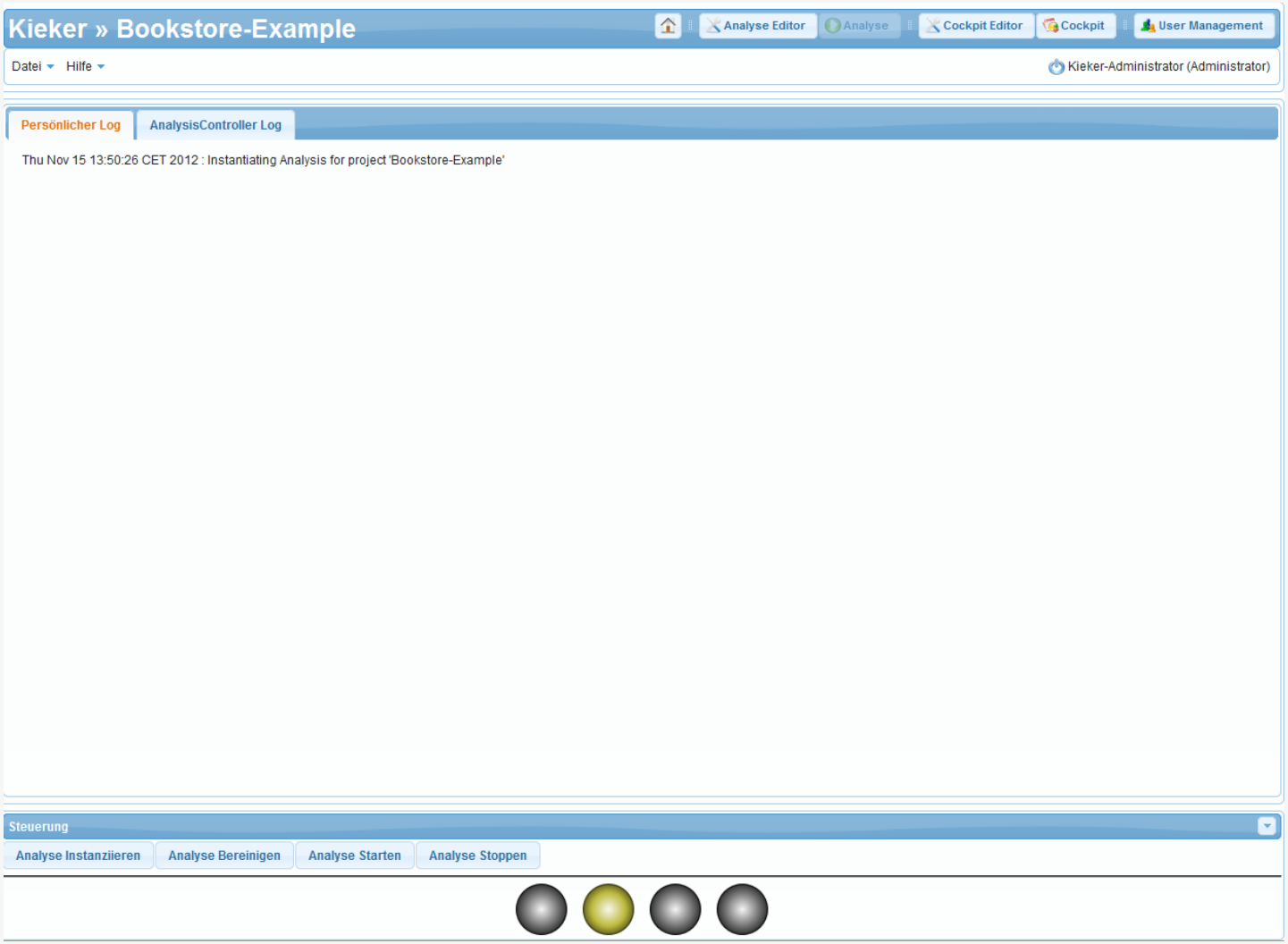
Running the analysis via the Kax-Runner

```
kax-run.bat/sh -i analysis.kax
```

Assembling the analysis in the WebGUI



Running the analysis in the WebGUI



Implement Own Plugins


```
@Plugin(
  name = "Response time filter",
  outputPorts = {
    @OutputPort(name = "out", eventTypes = {MyResponseTimeRecord.class})},
  configuration = {
    @Property(name = "thresholdNanos", defaultValue = "1000000")})
public class MyResponseTimeFilter extends AbstractFilterPlugin {

  @InputPort(
    name = "newResponseTime", eventTypes = {MyResponseTimeRecord.class})
  public void newResponseTime(final MyResponseTimeRecord rtRecord) {
      ...
      super.deliver("out", rtRecord);
  }
}
```

Initial (coarse) specification / project goals

- Web application for the analysis
- Tool to assemble and execute an analysis

Initial (coarse) specification / project goals

- Web application for the analysis
- Tool to assemble and execute an analysis

Current Specification

- Multi-user web application with integrated user management
- Management for the analysis projects
- Visual analysis editor
- Cockpit editor with cockpit / dashboard

Current State

- Is currently developed in an iterative way
- Project management implemented
- User management implemented
- Visual analysis editor
- Execution of analyses possible

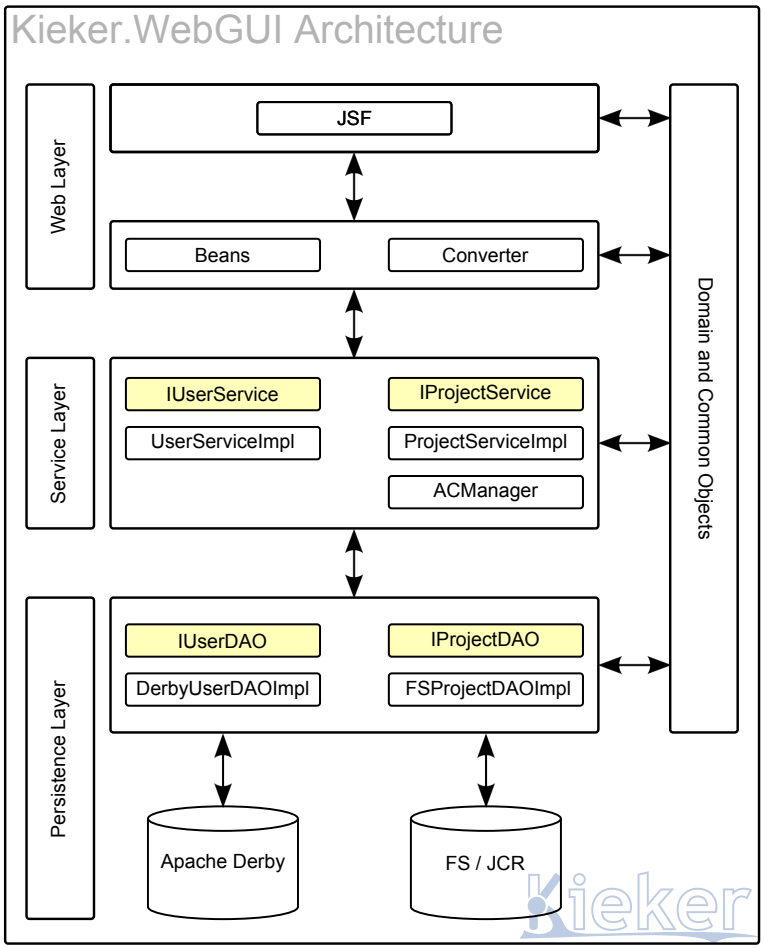


Figure : Current Architecture

Live Demonstration

Summary

- Kieker's pipes-and-filters API
- Different assembly and execution methods
- Kieker.WebGUI: Goals and Demo

Summary

- Kieker's pipes-and-filters API
- Different assembly and execution methods
- Kieker.WebGUI: Goals and Demo

Next Steps

- Visual and interactive cockpit
- Refinement and refactoring of existing code
- Improve configuration possibilities
- User manual and screencast
- Textual DSL for kax files

Kieker.WebGUI (Beta) included since Kieker 1.6

`http://kieker-monitoring.net`