



A Concurrent and Distributed Analysis Framework for Kieker

— Joint Kieker / Palladio Days 2013 —

Nils Christian Ehmke, Jan Waller, and Wilhelm Hasselbring

Software Engineering Group
Kiel University, Germany

November 28, 2013 @ Karlsruhe



- 1 Introduction
- 2 Kieker
- 3 Development
- 4 Evaluation
- 5 Conclusion

Trace Analysis

- ▷ View on the dynamic architecture
- ▷ Supporting tool for SEs during maintenance tasks
- ▷ Useful for legacy **and** modern software systems

Types of Trace Analyses

- ▷ Offline (post-mortem)
- ▷ Online
 - ▷ Has to process a high amount of traces
 - ▷ Needs to be performed very fast
 - ▷ **ExplorViz**



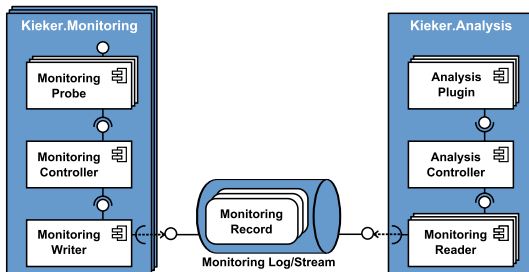
Kieker

- ▷ Supports monitoring and analysis
- ▷ Not designed for concurrent and distributed analysis networks
- ▷ Online trace analysis difficult to implement

- 1 Introduction
- 2 Kieker
- 3 Development
- 4 Evaluation
- 5 Conclusion

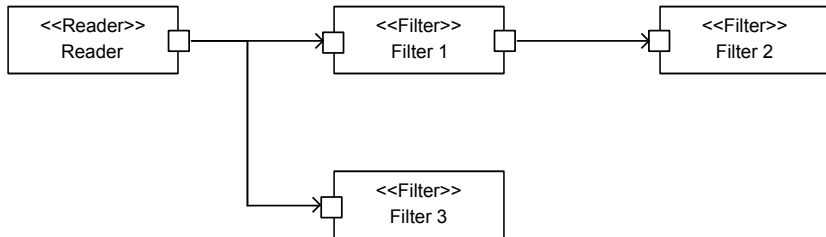
Introduction

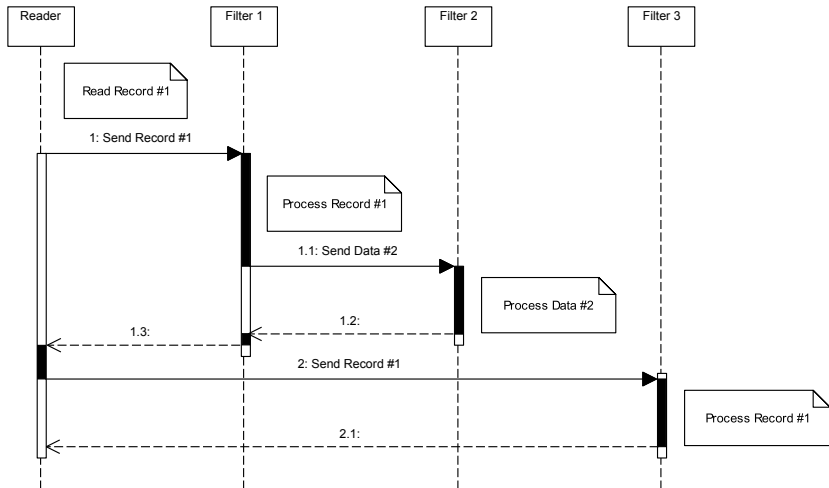
- ▷ Framework and tool collection
- ▷ Monitoring and analysis
 - ▷ Java, Visual Basic 6, COBOL, .Net, ...
- ▷ Developed at the Kiel University and the University of Stuttgart



Analysis Part

- ▷ Typical pipes and filters architecture
- ▷ Untypical repositories
- ▷ Analysis Controller as supervisor
- ▷ Plugins can use deliver method to send data



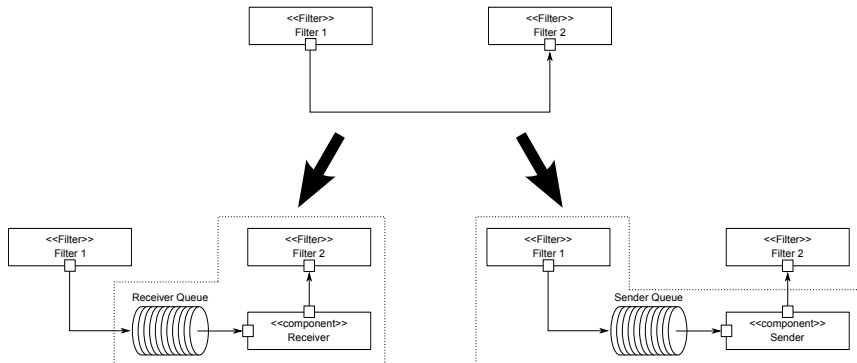


- Synchronous Message Call
- ←-- Return Call

- 1 Introduction
- 2 Kieker
- 3 Development**
- 4 Evaluation
- 5 Conclusion

Basic Design

- ▷ Ports can be asynchronous
- ▷ Use unbounded FIFO buffers between plugins



Data Forwarding

- ▶ Intercept asynchronous delivering
- ▶ Extend the deliver method

Sender and Receiver Threads

- ▶ Add a thread for each asynchronous port
- ▶ Thread can send or receive data

Analysis Shutdown

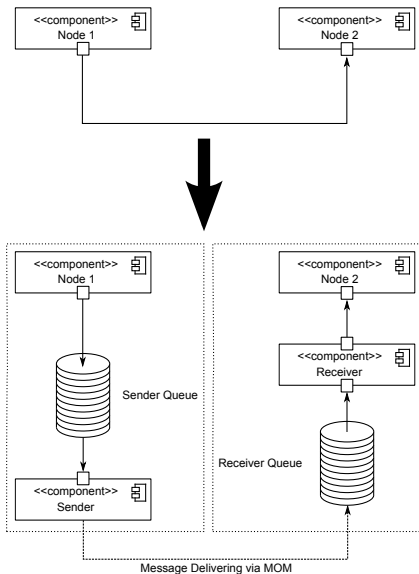
- ▶ Send initialization and termination signals through the network
- ▶ Termination is autonomous

```
IAnalysisController ac = new AnalysisController();  
...  
Configuration tfConfiguration = new Configuration();  
tfConfiguration.setProperty(  
    AbstractPlugin.CONFIG_ASYNC_INPUT_PORTS,  
    TeeFilter.INPUT_PORT_NAME_EVENTS);  
TeeFilter teeFilter = new TeeFilter(tfConfiguration, ac);  
...  
ac.run();
```

Basic Design

- ▷ Composite filters into analysis nodes
- ▷ Analysis can contain multiple nodes
- ▷ Nodes have one port of each kind
- ▷ Nodes can be distributed
- ▷ Usage of MOM for message routing

Development of the Distributed Part (cont'd)



Repositories

- ▷ Only theoretically considered
- ▷ Specify access to repositories via ports
- ▷ Cumbersome to implement in Kieker

Analysis Start and Shutdown

- ▷ Start nodes manually
- ▷ Send initialization and termination signals through the network
- ▷ Termination is autonomous

```
IAnalysisController ac = new AnalysisController();  
  
...  
  
Configuration nodeConfig = new Configuration();  
nodeConfig.setProperty(  
    AnalysisNode.CONFIG_PROPERTY_NAME_DISTRIBUTED, "true");  
nodeConfig.setProperty(  
    AnalysisNode.CONFIG_PROPERTY_NAME_NODE_NAME, "node2");  
AnalysisNode node2 = new AnalysisNode(nodeConfig, ac);  
  
AbstractPlugin tf = node2.createAndRegister(TeeFilter.class,  
    new Configuration());  
node2.connectWithInput(tf, TeeFilter.INPUT_PORT_NAME_EVENTS);  
  
...  
  
node2.connect("node1");  
  
...  
  
ac.run();
```


- 1 Introduction
- 2 Kieker
- 3 Development
- 4 Evaluation**
- 5 Conclusion

Statistical and Technical Methods

- ▷ Performed on various blade servers
- ▷ Each experiment uses up to six data sets
- ▷ Each experiment compares up to five different configurations
- ▷ Each configuration is executed ten times
- ▷ Five warm-up runs in the same JVM
- ▷ Cooldown time between the runs of the distributed experiments
- ▷ Student t-distribution
- ▷ Confidence intervals ($\alpha = 0.05$)

Experiments

- ▷ 2 x CPU and Memory/Swap records processing
- ▷ 2 x Trace Analysis
- ▷ 3 x Trace Reconstruction

Results

- ▷ Very high memory consumption
- ▷ Slightly positive speedup during some experiments
- ▷ Poor efficiency

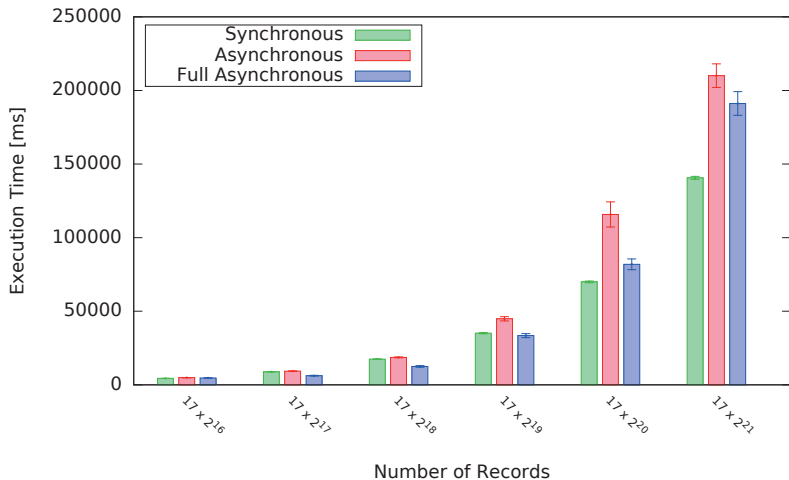


Figure: Results from a CPU and Memory/Swap record processing experiment

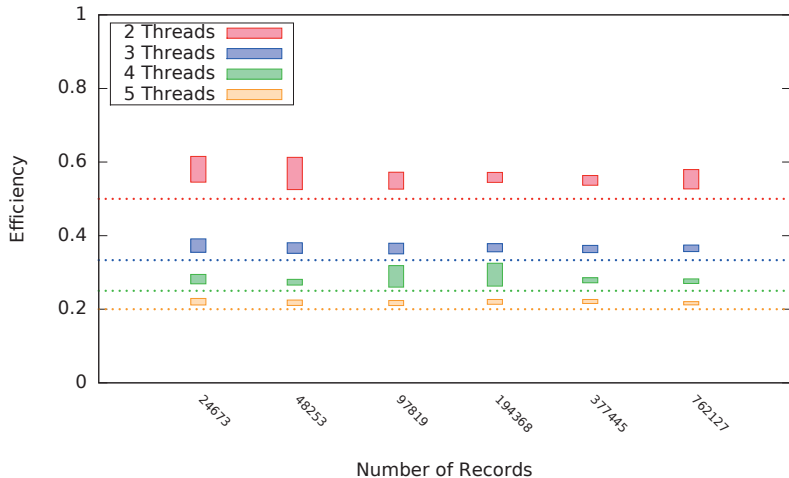


Figure: Results from a trace reconstruction experiment

Experiments

- ▷ 2 x CPU and Memory/Swap records processing
- ▷ 2 x Trace Analysis
- ▷ 2 x Trace Reconstruction

Results

- ▷ Very high memory consumption within the working nodes
- ▷ Negative speedup during all experiments
- ▷ Similar order of magnitude during most of the experiments

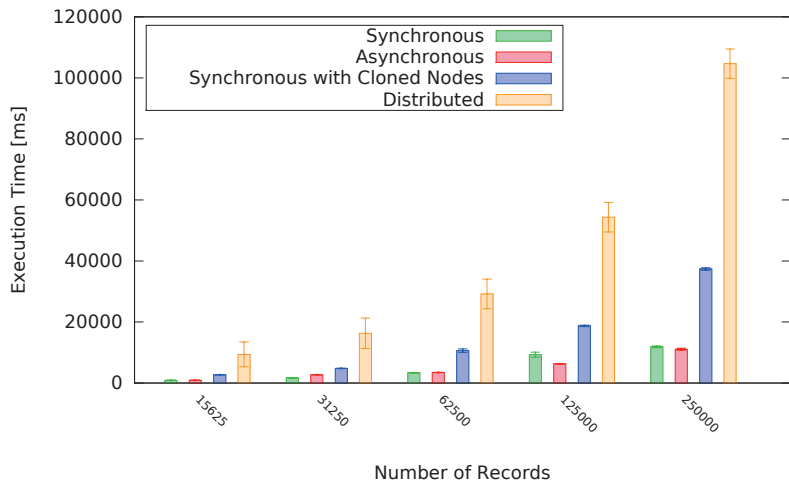


Figure: Results from a trace analysis experiment

- 1 Introduction
- 2 Kieker
- 3 Development
- 4 Evaluation
- 5 Conclusion**

Summary

- ▷ Presented an approach for concurrent and distributed analyses
- ▷ Measured the approach with various lab experiments

Results

- ▷ No speedup in common analysis networks
- ▷ High memory consumption
- ▷ Could theoretically be used for very specific analyses

Outlook

- ▷ Suitable data structure for the buffers
- ▷ Use bounded buffers
- ▷ Execute each filter in an own thread
- ▷ Rework termination sequence

- ▷ Decentralized MOM solution
- ▷ Batching of messages

Recommendation

- ▷ Carry on with research in the direction of a concurrent framework

