Automatic Failure Diagnosis Support in Distributed Large-Scale Software Systems based on Timing Behavior Anomaly Correlation

Presentation at 13th European Conference on Software Maintenance and Reengineering

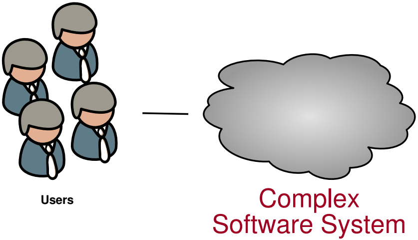
Nina Marwede[1], **Matthias Rohr**[1],
André van Hoorn[2], Wilhelm Hasselbring[3]

[1]BTC Business Technology Consulting AG, Germany
[2]Graduate School TrustSoft, University of Oldenburg, Germany
[3]Software Engineering Group, University of Kiel, Germany

Contact: matthias.rohr@btc-ag.com

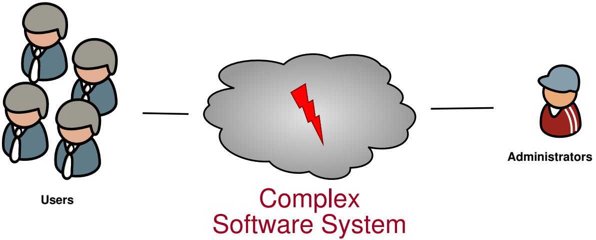
March 25, 2009

Motivation



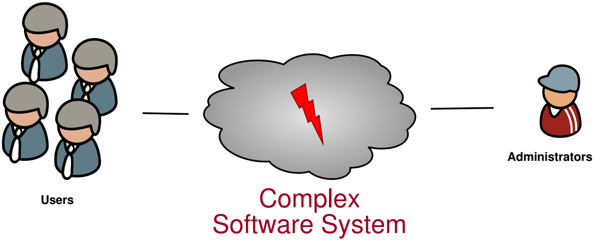
- Complex software systems are almost never free of faults.

Motivation



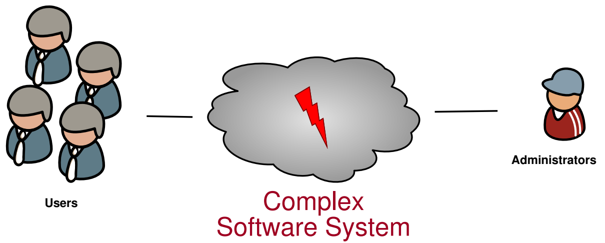
- Complex software systems are almost never free of faults.
- Software faults are a major cause for system failures [Küng and Krause, 2007; Gray, 1986]

Motivation



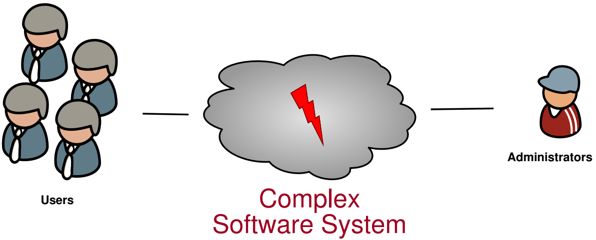
- Complex software systems are almost never free of faults.
- Software faults are a major cause for system failures [Küng and Krause, 2007; Gray, 1986]
- Manual failure diagnosis is time-consuming and error-prone.

Motivation



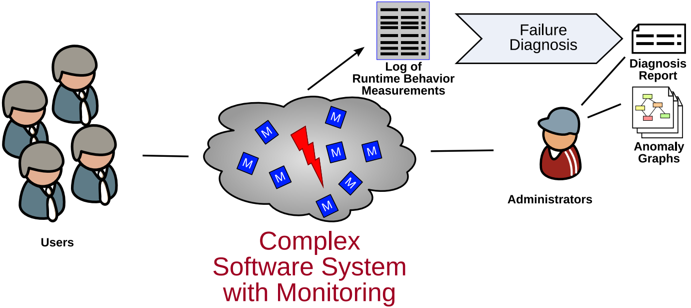
- Complex software systems are almost never free of faults.
- Software faults are a major cause for system failures [Küng and Krause, 2007; Gray, 1986]
- Manual failure diagnosis is time-consuming and error-prone.
  - Huge amount of program states (space and time) [Cleve and Zeller, 2005]
  - Temporal & spatial chasms between cause and symptom [Eisenstadt, 1997]
  - Many systems are not known completely by a single person
  - Some failure are hard to repeat – e.g., Heisenbugs

Motivation



- Complex software systems are almost never free of faults.
- Software faults are a major cause for system failures [Küng and Krause, 2007; Gray, 1986]
- Manual failure diagnosis is time-consuming and error-prone.
- Most common failure diagnosis methods [Eisenstadt, 1997]:
  - Data-gathering (e.g., print-statements to source code, memory dumps)
  - Interactive execution using debugging tools

Motivation



Strategy to support failure diagnosis

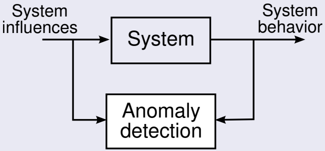
- Runtime behavior is indicative for failures and error-propagation.
- Automatic fault localization using anomaly detection on monitoring data.
- Analysis and visualization in the context of automatically derived architecture models.

Outline

Online failure diagnosis based on anomaly detection

Anomalies

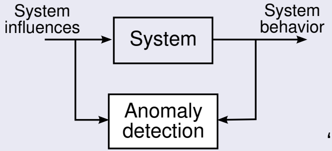
- Anomalies are deviations from normal system behavior.



Online failure diagnosis based on anomaly detection

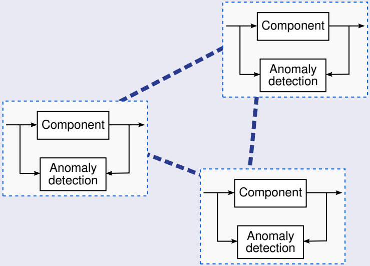
Anomalies

- Anomalies are deviations from normal system behavior.



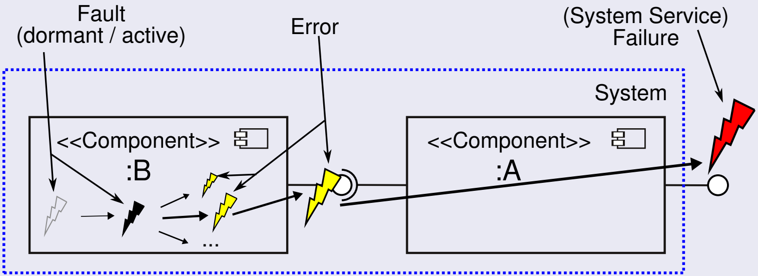
Fault localization activities

- Anomaly Detection
- Anomaly Correlation (often plain aggregation)
- Visualization and/or reporting



Propagation and Anomaly Detection

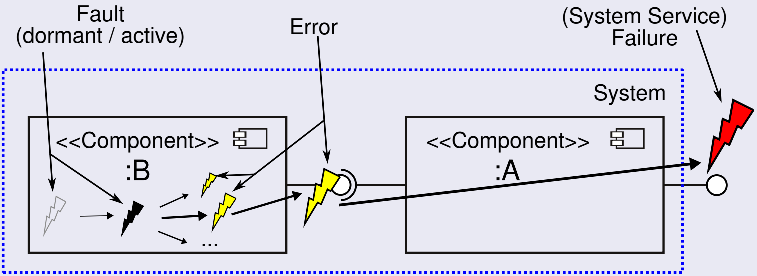
Error propagation



- Many errors propagate along *calling dependencies*.

Propagation and Anomaly Detection

Error propagation

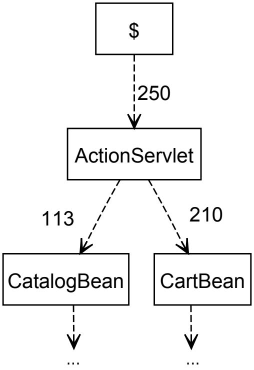


- Many errors propagate along *calling dependencies*.

Anomaly correlation

- Anomalies propagate as well - compensating analysis is required.
- Some approaches analyze anomalies in context of *calling dependency graphs*.

Dependency Graphs

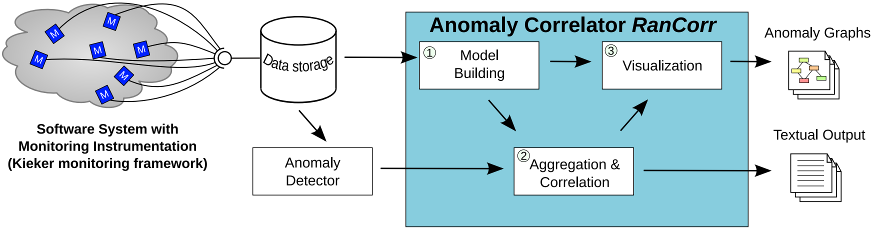


Calling Dependency Graphs

- Nodes: E.g., Operations, Components, Deployment contexts, Virtual Machines
- Directed edges represent call actions
- Weights quantify call frequencies

Contents

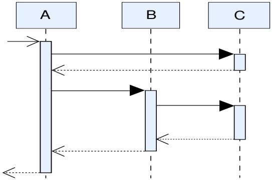
Overview



M
Data storage
Software System with
Monitoring Instrumentation
(Kieker monitoring framework)
Anomaly Detector
Anomaly Correlator *RanCorr*
① Model Building
② Aggregation & Correlation
③ Visualization
Anomaly Graphs
Textual Output

Input Data

1. Calling dependencies between operations

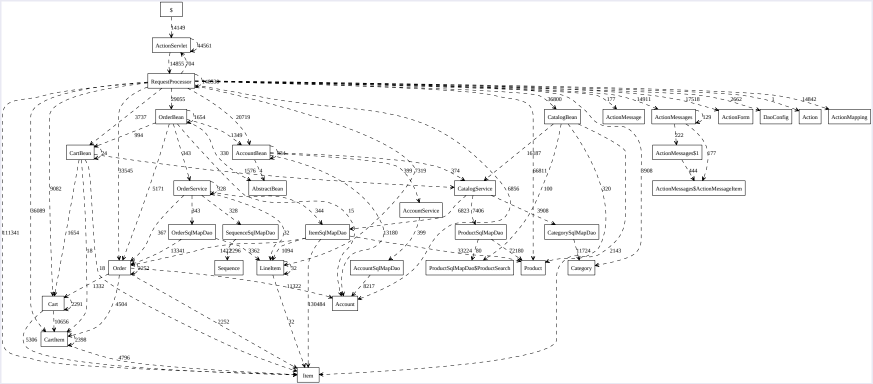


2. Anomalies scores provided by a timing behavior anomaly detector

Comp	VM	Start	RT	Anomaly
...				
A	X	0001	8	0.6
C	Y	0002	1	–0.2
B	X	0004	4	0.9
C	Y	0006	2	0.3
...				

Architectural model creation

Calling Dependency Graph (class granularity) for iBatis JPetStore



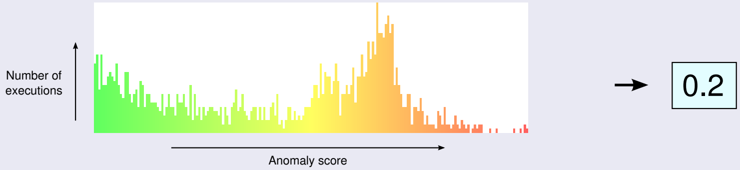
Two alternative methods for creating the CDG:

- **Analysis of monitoring data**
- Static (source code) analysis

Aggregation and integration into the architectural model

Approach

- Each architectural element's anomaly scores are aggregated into a single value
  - Several metrics explored (mean, median, power mean, ...)

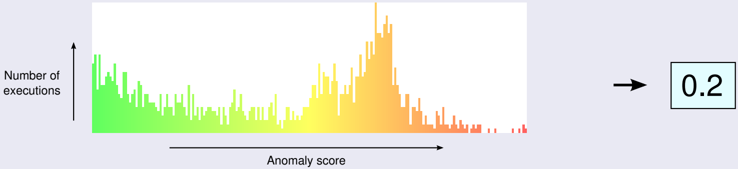


- The aggregation reduces the complexity for the correlation activity

Aggregation and integration into the architectural model

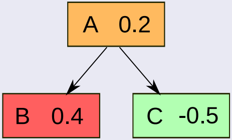
Approach

- Each architectural element's anomaly scores are aggregated into a single value
  - Several metrics explored (mean, median, power mean, ...)



- The aggregation reduces the complexity for the correlation activity

Example result: Three operations with assigned anomaly scores

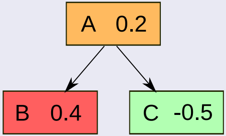


Correlation of anomaly ratings

Approach

- Rules are applied that recompute an elements anomaly score in the context of its callers and callees
    - Similar approach to cellular automaton
- The rules encapsulate error and anomaly propagation knowledge

Example scenario: Is A's anomaly score just the result of a fault in B?

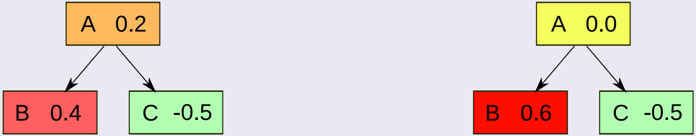


Correlation of anomaly ratings

Approach

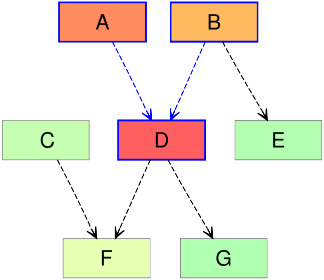
- Rules are applied that recompute an elements anomaly score in the context of its callers and callees
  - Similar approach to cellular automaton
- The rules encapsulate error and anomaly propagation knowledge

Example scenario: Is A's anomaly score just the result of a fault in B?



Rules

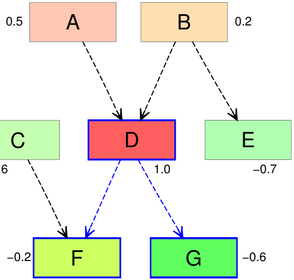
- Rule 1:
  Mean of anomaly ratings of directly connected **callers** . . .
  relatively high? $\Rightarrow$ Increase rating



Rules

- Rule 1:
  Mean of anomaly ratings of directly connected **callers** . . .
  relatively high? $\Rightarrow$ Increase rating

- Rule 2:
  Maximum of anomaly ratings of directly connected **callees** . . .
  relative high? $\Rightarrow$ Decrease rating



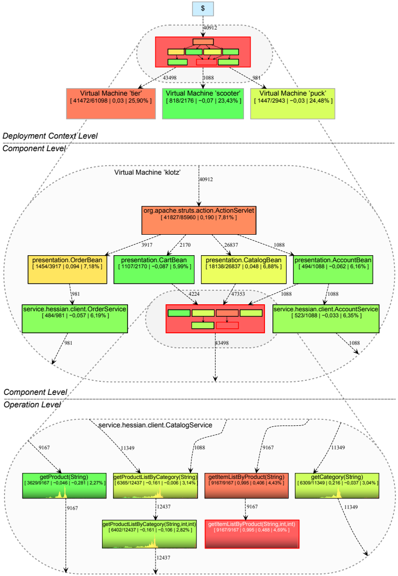
Rules

- Rule 1:
  Mean of anomaly ratings of directly connected **callers** . . .
  relatively high? $\Rightarrow$ Increase rating
- Rule 2:
  Maximum of anomaly ratings of directly connected **callees** . . .
  relative high? $\Rightarrow$ Decrease rating
- Additional rules:
  - Consideration of call frequencies (edges in CDG)
  - Transitive closure of callers
  - Transitive closure of callees

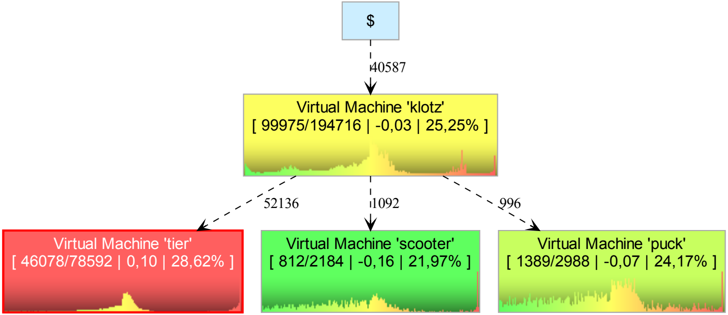
Visualization - Three visualization granularity levels

Granularity levels:

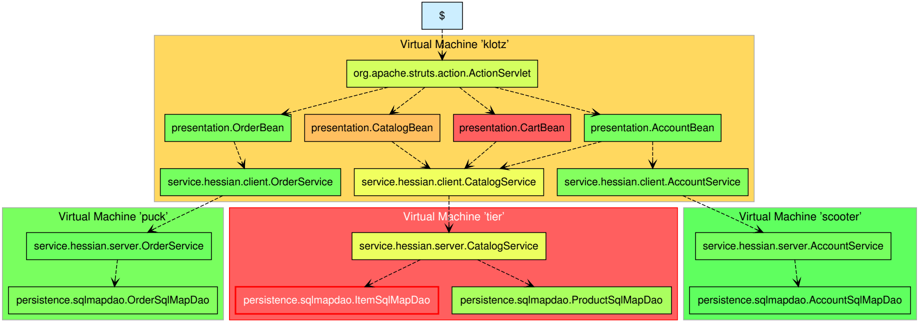
- Deployment context level / Virtual Machine level
- Component level
- Operation level



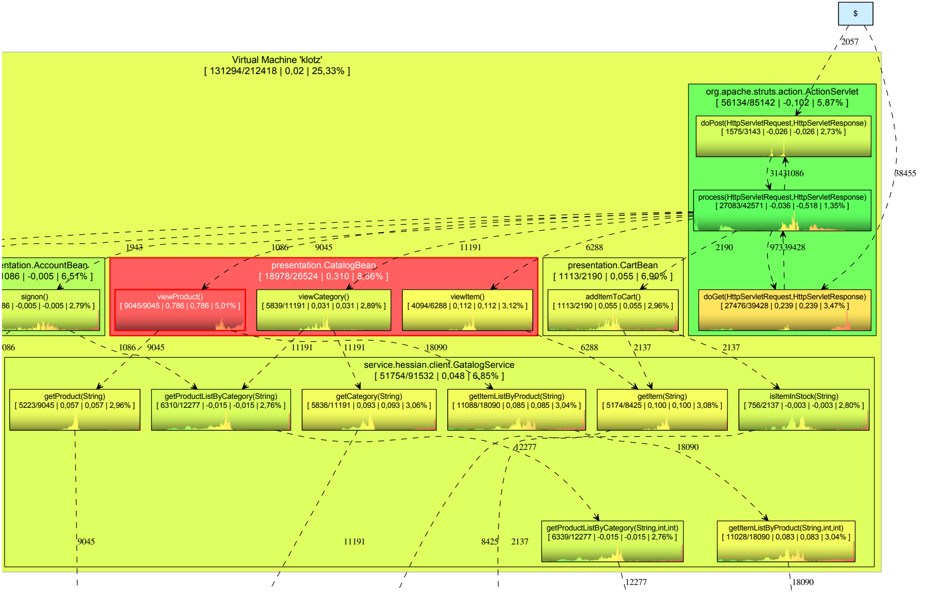
Visualization - Deployment context / Virtual Machine level



Component level



Operation level



Contents

Goals & Metrics

Goals

- Proof of concept
- Quantitative evaluation
- Visualization evaluation

Metrics

- Accuracy:
  Are injected faults accurately localized?
- Clearness:
  Are the results clearly (sufficient contrast) ranked?

Experiment Setup

- Distributed variant of iBATIS JPetStore (5 nodes)
- 34 operations are instrumented with monitoring probes
- Workload generation
  - Probabilistic user behavior
- Fault injection
  - Programming faults
  - Database connection slowdown
  - Hard disk misconfiguration
  - Resource intensive concurrent processes
  - CPU throttling

Results: Experiment statistics and fault localization quality

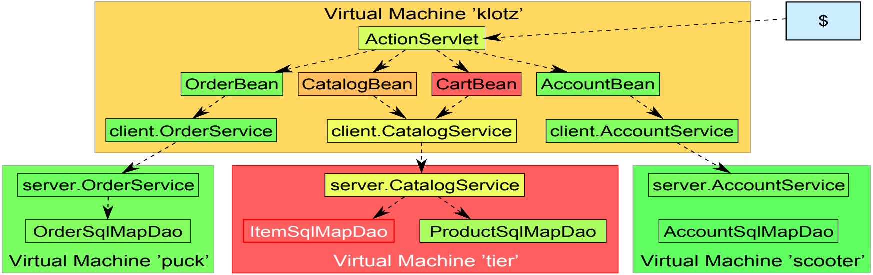
Experiment statistics

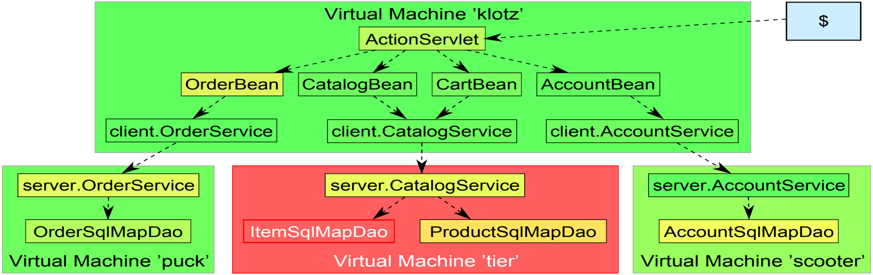
- 42 experiment scenarios
- 20 hours total experiment time
- 16 million monitored executions
- 100 MB data per experiment run

Fault localization quality (Accuracy and Clearness)

Scenario	Injection	"Trivial"	"Simple"	"Advanced"
No. 1	Progr. fault	+	+	+
No. 2	Progr. fault	+	+	++
No. 3	Progr. fault	-	-	+
No. 4	DB slowdown	+	++	++
No. 5	DB slowdown	o	+	++
Averages		3.4	3.8	4.6

Visualization Clearness: No correlation vs. our approach





Contents

Issues

- Number of monitoring points:
  - Too less: Architecture and its dependencies not discovered
  - Too many: Large overhead
  - Trade-off: Major component services and entry points
- Monitoring overhead:
  - Overhead approx. few microseconds/observation
- Maintainability:
  - Approach automatically adapts to architectural changes
  - Non-intrusive monitoring instrumentation
- Anomaly detector requirements:
  - False alarms (false positives) can be tolerated if equally distributed over the architecture
- Computational requirements:
  - 35.000 executions/sec on 1.5 GHz Desktop

Summary & Conclusions

Summary

- New approach for failure diagnosis (focus on correlation and visualization)
- Evaluation of accuracy and clearness of correlation algorithms
- Case study with distributed web-application, fault injection, and probabilistic workload

Conclusions

- Good chance of localizing the fault
- Large system parts are declared of *not* being a fault's cause
- Approaches without correlation show a fault's effect, not its origin
- Multi-granularity visualization even for small systems required

Questions?

Bibliography

Holger Cleve and Andreas Zeller. Locating causes of program failures. In *Proceedings of the 27th International Conference on Software Engineering (ICSE'05)*, pages 342–351. ACM Press, May 2005. ISBN 1595939632.

Marc Eisenstadt. My hairiest bug war stories. *Commun. ACM*, 40(4):30–37, 1997. ISSN 0001-0782. doi:10.1145/248448.248456.

Simon Giesecke, Matthias Rohr, and Wilhelm Hasselbring. Software-Betriebs-Leitstände für Unternehmensanwendungslandschaften. In *Proceedings of the Workshop "Software-Leitstände: Integrierte Werkzeuge zur Softwarequalitätssicherung"*, volume P-94 of *Lecture Notes in Informatics*, pages 110–117. Gesellschaft für Informatik, October 2006. ISBN 978-3-88579-188-1.

Jim Gray. Why do computers stop and what can be done about it? In *Proceedings of Symposium on Reliability in Distributed Software and Database Systems (SRDS-5)*, pages 3–12. IEEE, 1986.

Peter Küng and Heinrich Krause. Why do software applications fail and what can software engineers do about it? a case study. In *Proceedings IRMA Conference: Managing Worldwide Operations and Communications with Information Technology*, pages 319–322. IGI Publishing, 2007. ISBN 978-1-59904-929-8.

Matthias Rohr, André van Hoorn, Jasminka Matevska, Nils Sommer, Lena Stoever, Simon Giesecke, and Wilhelm Hasselbring. Kieker: Continuous monitoring and on demand visualization of Java software behavior. In *Proceedings of the IASTED International Conference on Software Engineering 2008*, pages 80–85. ACTA Press, February 2008. ISBN 978-0-88986-715-4.

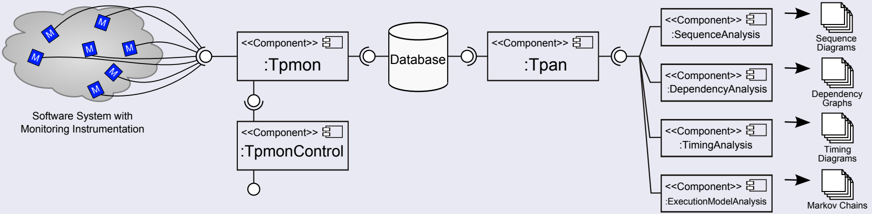
Characteristics monitored by online fault localization approaches

- Hardware platform (e.g., CPU, Network, Memory)
- Operating system and middleware (e.g., OS Services, Application server, Virtual machine)
- Internal software application behavior:
  - Operation execution sequences (Control flow)
  - Response times (end-to-end and of single software operations)
  - Application output

Characteristics monitored by online fault localization approaches

- Hardware platform (e.g., CPU, Network, Memory)
- Operating system and middleware (e.g., OS Services, Application server, Virtual machine)
- Internal software application behavior:
  - Operation execution sequences (Control flow)
  - Response times (end-to-end and of single software operations)
  - Application output

Monitoring Framework Kieker [Rohr et al., 2008]



Open and Future Work

Future work:

- Field studies on accuracy and clearness
- Evaluation of the visualization method (field or lab study)
  - Are three architectural levels better than two or four?
- Evaluation of the complete approach
  - Whats the benefit in terms of repair time reduction?
- Continuous analysis and visualization
  ("Leitstand" / "Cockpit") [Giesecke et al., 2006]

Fault Injection

1. Programming faults
    - Duplicated code execution
    - Empty DB query result set
2. Database connection slowdown
    - `Thread.sleep(10)`
3. Hard disk misconfiguration
    - `hdparm -X mdma1 /dev/hda`
4. Resource intensive concurrent processes
5. CPU throttling
    - Simulation of a broken CPU cooling system