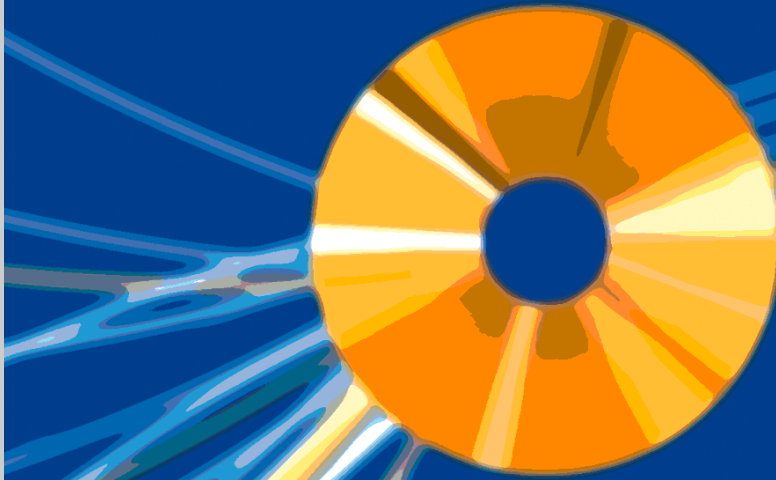


Passive Testing – A Solution to Testing Concurrent Systems



Andrej Pietschker, PhD

Siemens AG, CT SE 1

andrej.pietschker@siemens.com

ph: +49 (89) 636 55130



Software &
Engineering
Development
Techniques

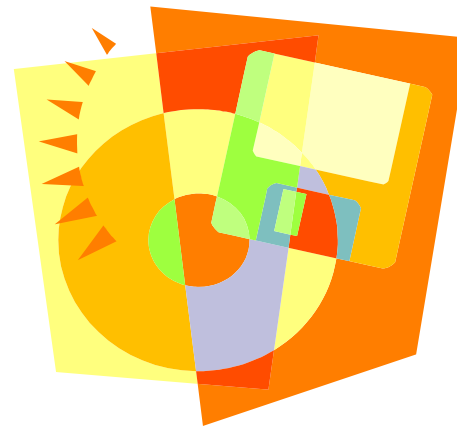
Presentation Overview

- **Motivation**
- **Light-Weight Approach to Passive Testing**
- **Model-based Approach to Passive Testing**
- **Conclusions**



Software &
Engineering
Development
Techniques

Motivation



Software &
Engineering
Development
Techniques

Fault Diagnosis in Distributed Systems

- **Fault diagnosis consists of the tasks**
 - Reproduce the failure observed, e.g., during testing
 - Examine the fault
 - Hypothesize about the error
- **Distributed, concurrent systems complicate matters**
 - Nondeterministic behavior
 - Reproducibility of failures difficult to achieve



Passive Testing

- **Tracing to support fault diagnosis**
 - Well-proven means for analyzing complex systems
 - Collects events from the running application
 - Usually large amount of data collected
- **Automatic offline analysis of traces**
 - Support of filtering, visualization, and details-on-demand
 - Flexible and easy usage of tools

⇒ Passive Testing

- *Active Testing*: Verification by means of interaction with SUT



Software &
Engineering
Development
Techniques

Light-Weight Approach to Passive Testing



Software &
Engineering
Development
Techniques

Information Seeking Mantra

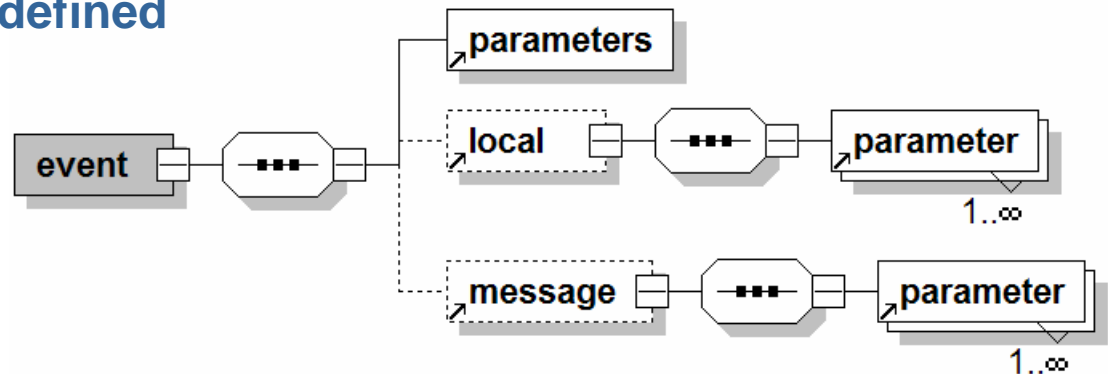
- **Passive testing approach applies concepts of**
 - Visualization
 - Trace data and analysis results are made accessible to the user
 - Different outputs supported, e.g. in SVG or HTML
 - Filtering
 - Reduces the size and information contained in a trace
 - Filters define operations over traces, e.g. selection, pruning
 - Details-on-Demand
 - Not all information should be present at all times
 - Displays further information when closer inspection is required



Software &
Engineering
Development
Techniques

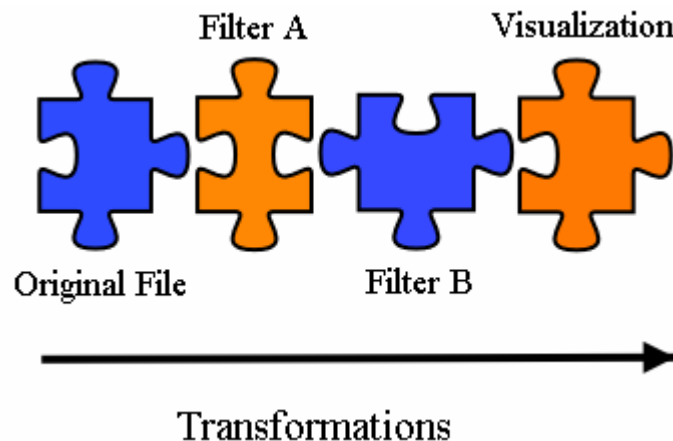
Structure of a Standardized Trace Format

- Trace contains the following event types
 - Registration and unregistration events of
 - Devices
 - Processes and/or threads
 - Objects
 - Communication events
 - Send and receive of messages
 - Local events
 - Variable assignments
 - Assertions
- Event structure defined in XML schema



Trace Transformations

- Trace present as XML-file (converted from other format)
- Output of a filter is another trace in XML format
- Filters can be combined
 - Supports reuse of filters
 - Filters are not necessarily commutative: $f \circ g \neq g \circ f$
- Visualization component applied if needed



Software &
Engineering
Development
Techniques

Example: Overview

- **Automotive application**
 - Driver information and navigation system
 - Embedded component-based software development project
 - System was designed with tracing already in mind!
- **System failure**
 - System slowed down unexpectedly when running on target hardware
 - Task: Analyze the cause of the slow-down!
- **Original trace contained more than 1 million events**
- **Analysis steps**
 - Identify threads with a large running time
 - Visualize the communication of these threads
- **Analysis result**
 - Slow-down caused by permanent polling of a device that was not operational



Example: Performance Analysis in HTML

Analysis for Trace:

Date	Name	Version
Fri Jul 06 14:52:40 GMT+02:00 2001	mmidis.log	0.2
Legend	fatal	attention
	thread never used	no unregistration event found; times represent last usage

Analysis Results

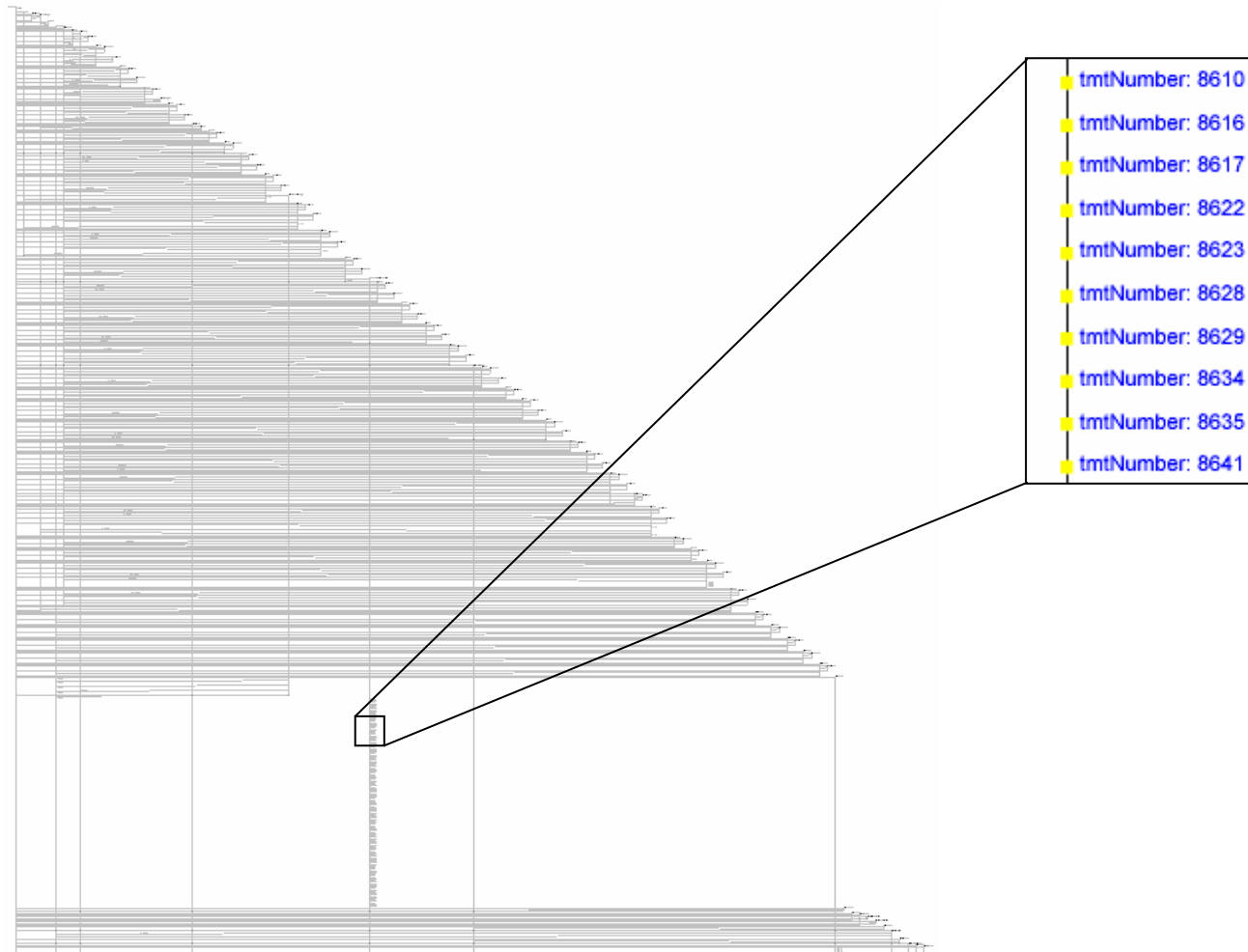
Results listed per thread

Thread-id	start time	end time	running time
1879048213	215895430	429823775	213928345
1879048214	217016280	434248769	217232489
1879048215	219756404	761841	761841
1879048216	220248567	417222381	196973814
1879048217	223307934	143855	143855
1879048218	223646869	223773846	126977
1879048219	225412332	402261	402261
1879048220	226134681	114494	114494
1879048221	226272623	242954	242954
1879048222	228729452	63014	63014
1879048223	228802288	41660	41660
1879048224	230796709	230981664	184955
1879048225	234883405	43210	43210



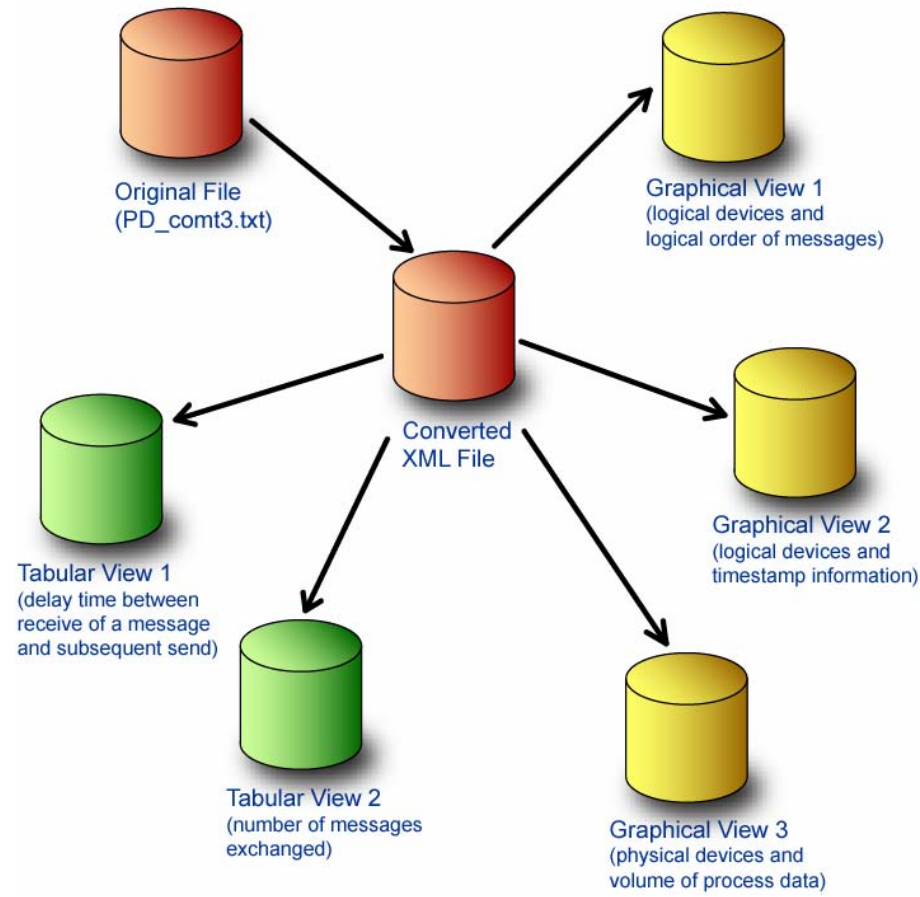
Software &
Engineering
Development
Techniques

Example: Graphical Trace Visualization in SVG



Software &
Engineering
Development
Techniques

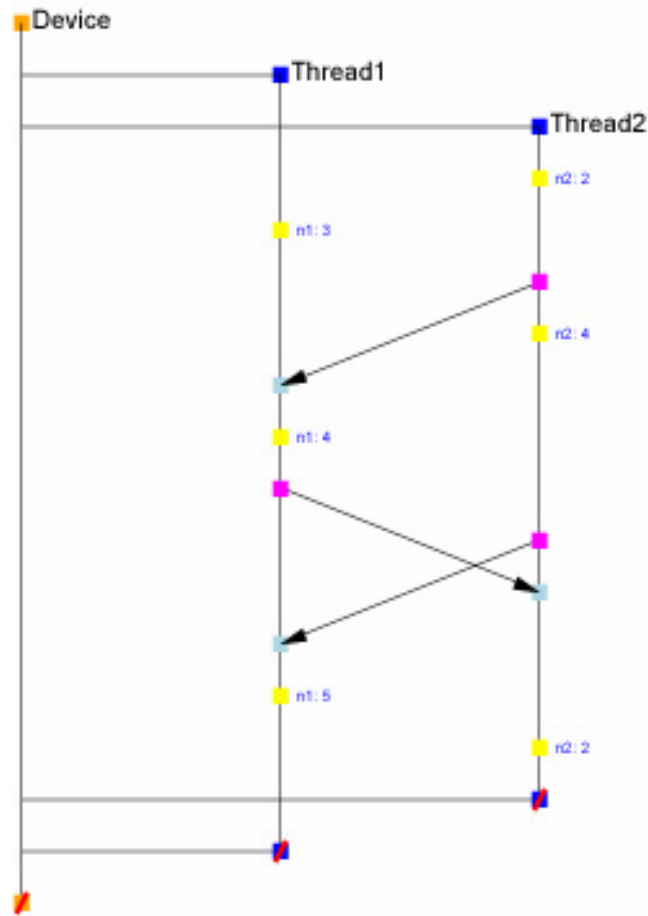
More Examples



Software &
Engineering
Development
Techniques

Taking Concurrency into Account

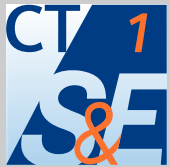
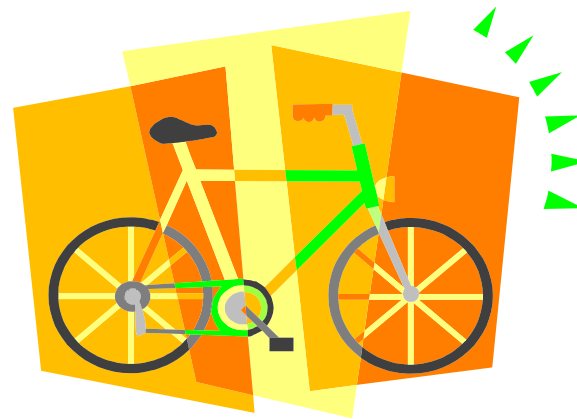
Message Sequence Chart.



Linear Trace File.

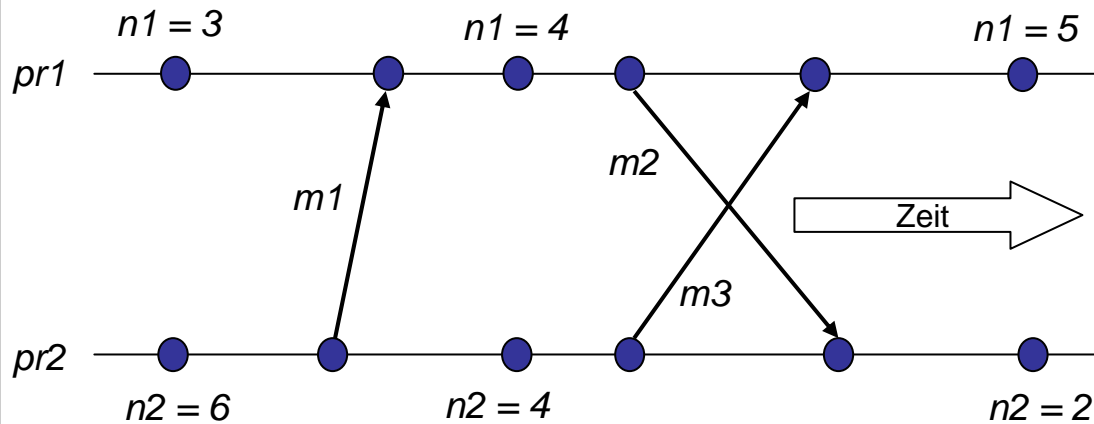


Model-based Approach to Passive Testing



Software &
Engineering
Development
Techniques

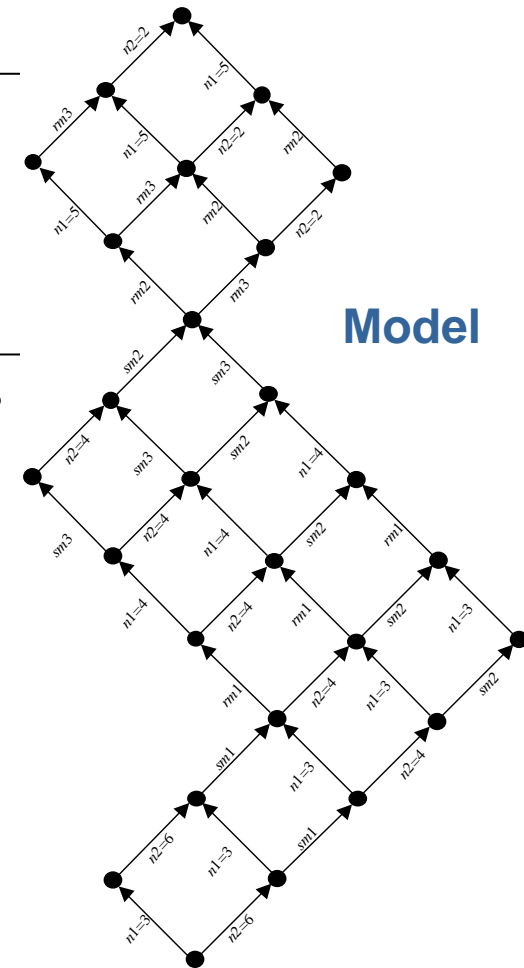
Log, Trace, and Model



Trace

```

pr1: n1=3
pr2: n2=6
pr2: snd m1 to pr1
pr2: n2=4
pr1: rcv m1 from pr2
pr2: snd m3 to pr1
pr1: n1=4
pr1: snd m2 to pr2
pr1: rcv m3 from pr2
pr2: rcv m2 from pr1
pr1: n1=5
pr2: n2=2
  
```

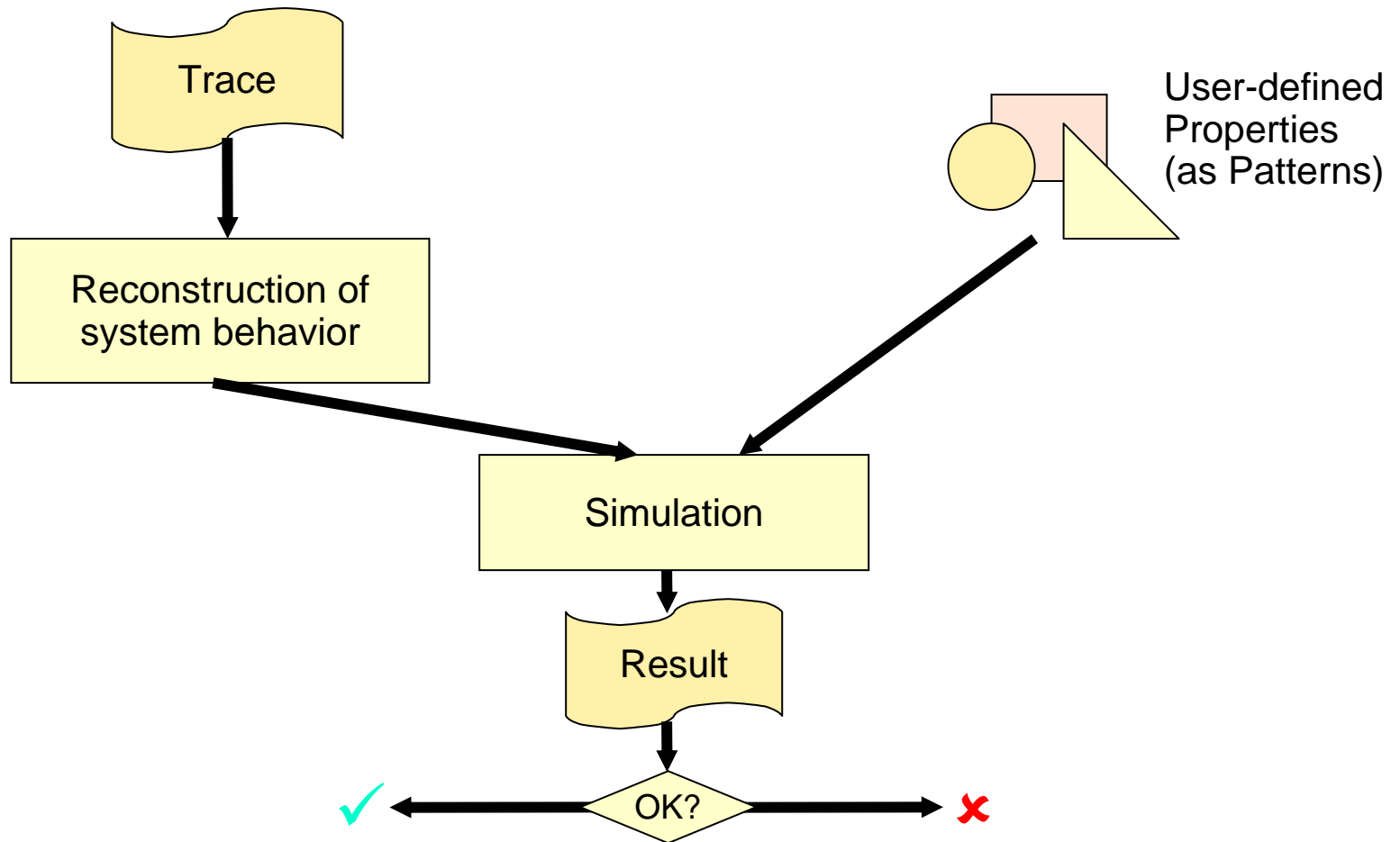


Model



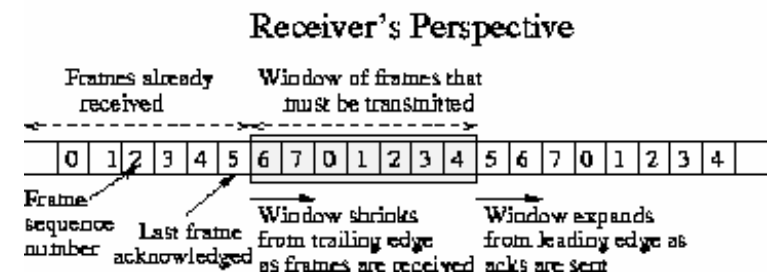
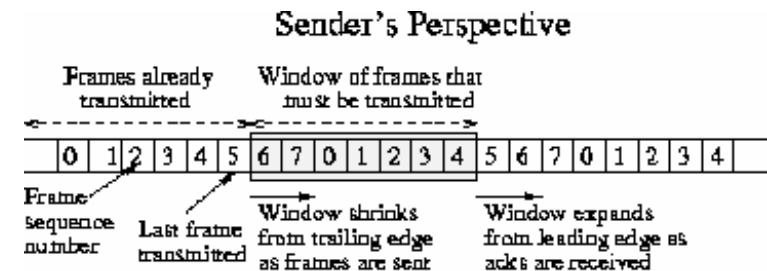
Software &
Engineering
Development
Techniques

Model-based Analysis – Overview



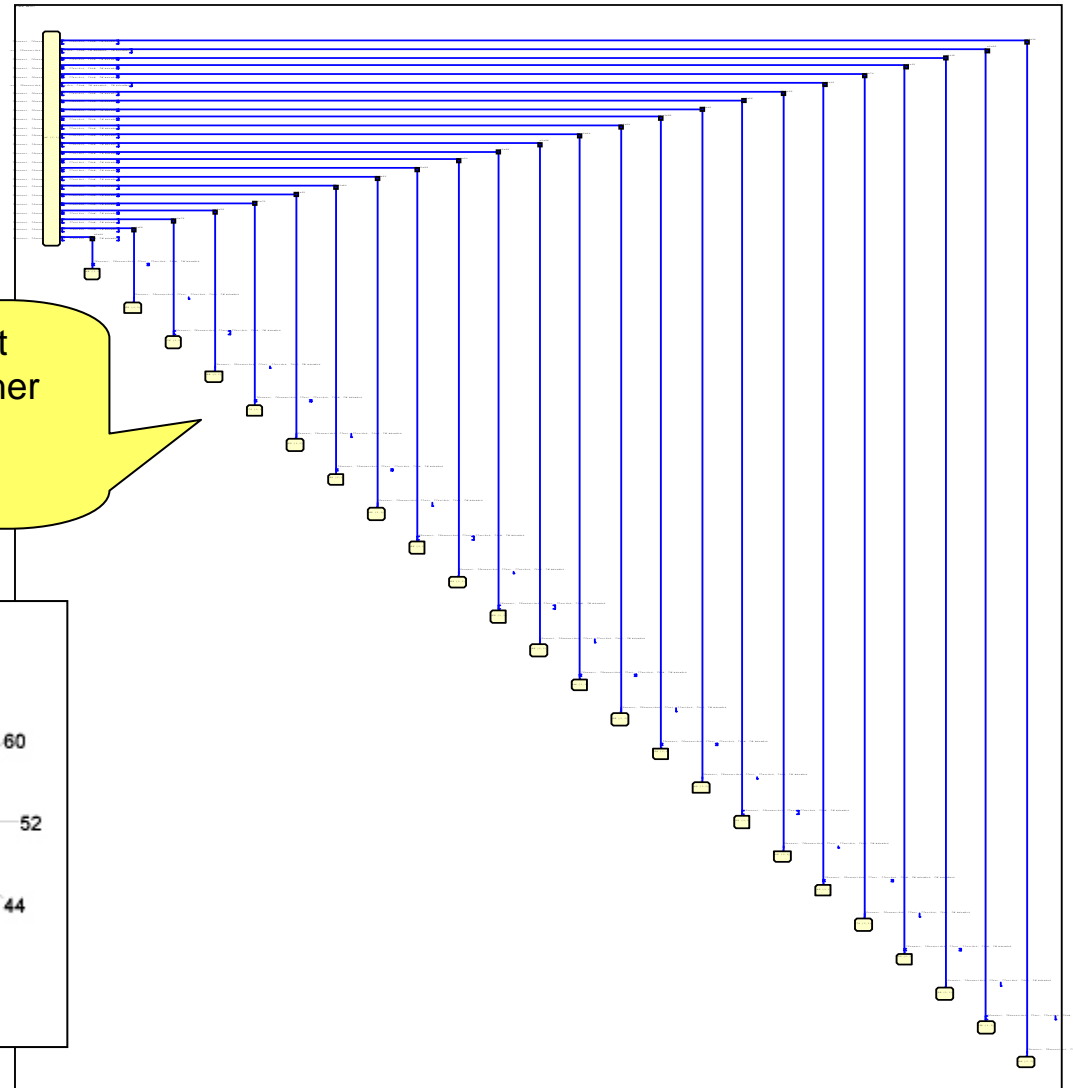
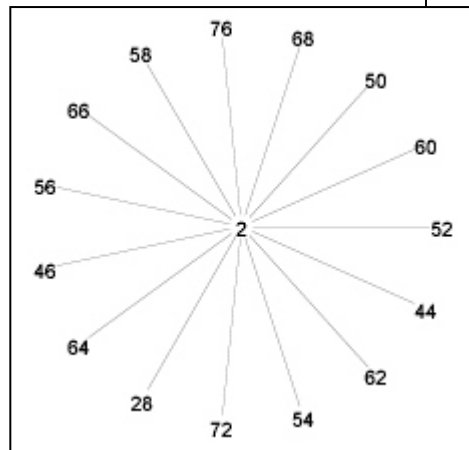
Example: Sliding Window Protocol

- A connection-oriented protocol (used e.g. in TCP)
- Allows data to be sent in one direction
 - between a pair of protocol entities,
 - subject to a maximum number of unacknowledged messages (window)
- If this window becomes full,
 - the protocol is blocked until an acknowledgement is received for the earliest outstanding message.
 - At this point the transmitter is clear to send more messages.



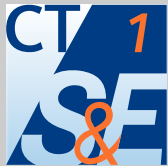
Architectural View of the System

1 centralized component communicating with all other components (star-form communication)

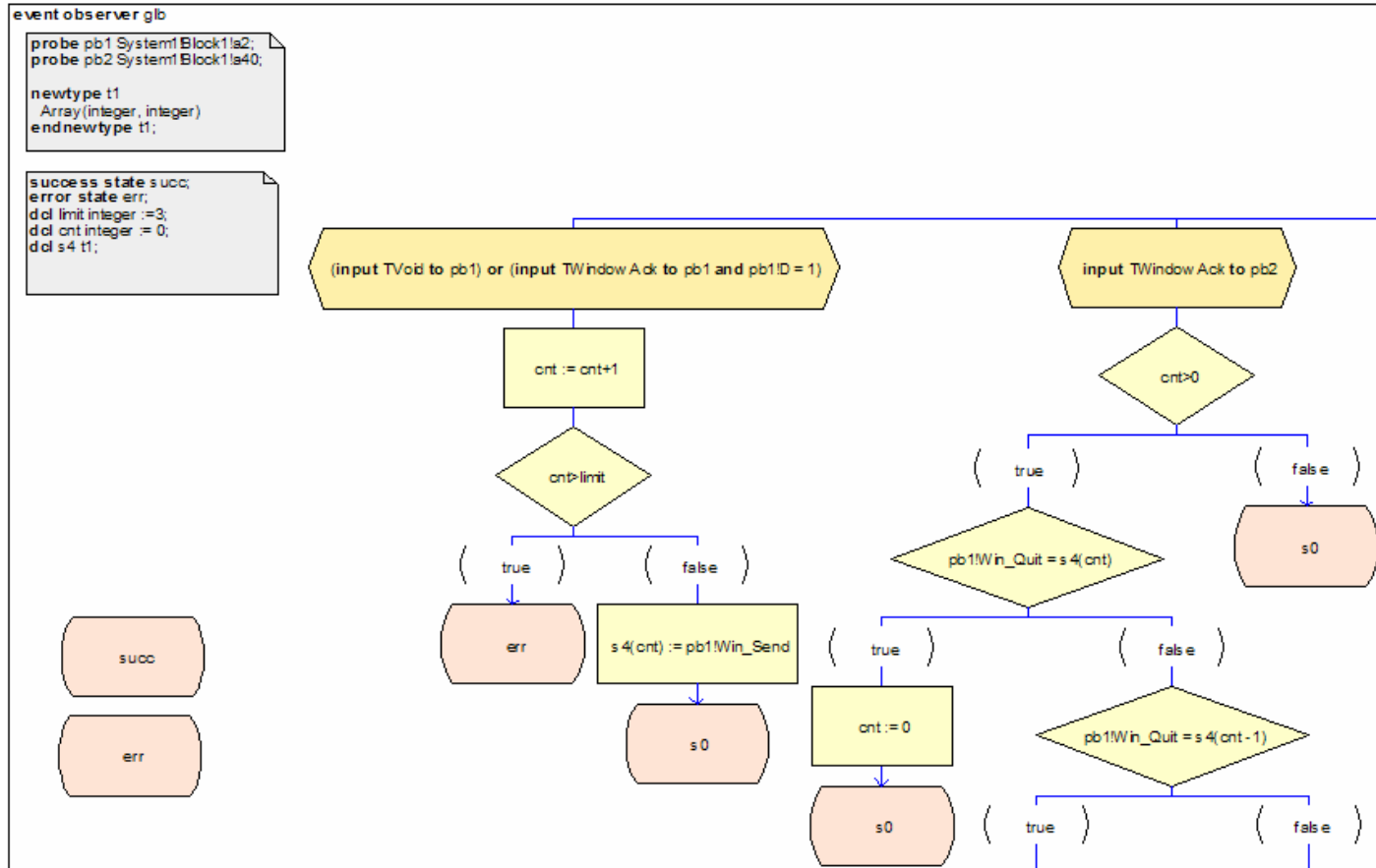


Passive Testing Workflow – Exemplified

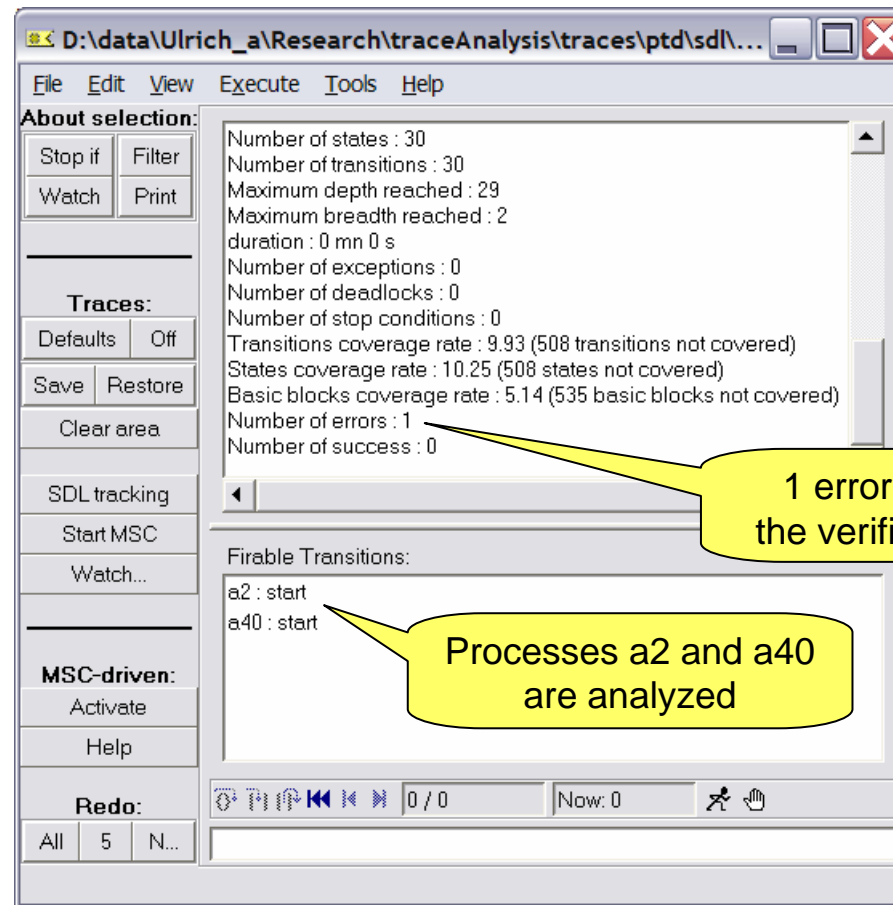
- **Start: Formalize system property**
 - **The sliding window property is not violated.**
- **Rephrase property as pattern over trace**
 - **Let SWS be the maximum sender window size.**
Let LAR be the sequence number of the last acknowledgment received.
Let LFS be the sequence number of the last frame sent.
 - **The sender maintains the invariant:**
 $LFS - LAR + 1 \leq SWS$.
- **Simulation of trace and pattern using a model checker**
- **Interpretation of result**



Main Part of the Analysis Pattern



Trace Analysis Execution



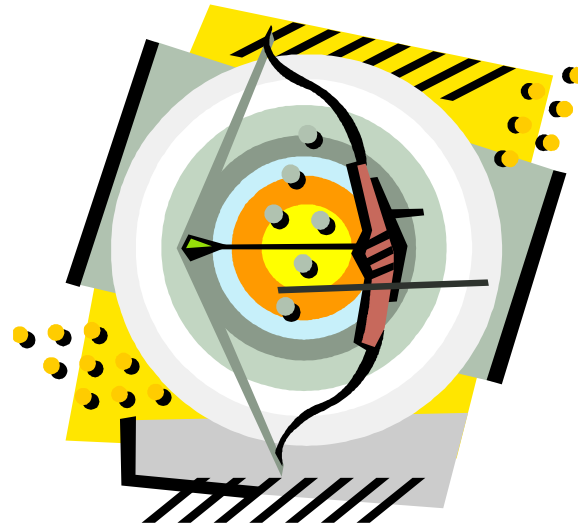
Analysis of Other System Properties

- **System architecture**
 - Analysis of initialization phase (thread or object view)
 - Analysis of creation and termination of threads or objects
 - Deadlock analysis
- **Race analysis**
 - Identification of objects used by more than one thread
 - Identification of messages intended for a single recipient (races)
 - Races are the main cause for nondeterministic system behavior
- **Performance analysis**
 - Analysis of communication, delays, usage times



Software &
Engineering
Development
Techniques

Conclusions



Software &
Engineering
Development
Techniques

Summary (1)

- **Input for Passive Testing**
 - A trace file with all the necessary information
 - Desired or undesired system properties
- **Fully automated analysis**
 - Tools exist!
- **Analysis results**
 - Indicators for fault analysis
 - Support during regression testing
 - Universal analysis of system properties
 - Non-determinism
 - Performance bottle-necks
 - Hints for redesign
- **Passive Testing usable for**
 - Distributed systems, especially for systems with many devices
 - Poorly documented systems
 - Systems containing Components of the shelve (COTS)



Software &
Engineering
Development
Techniques

Summary (2)

- **XML-based approach to trace analysis**
 - Offers quick and cheap solution to support fault diagnosis
 - Enables an interactive way to extract and visualize information gathered during testing
 - Uses standardized XML technologies
 - Benefits from the large variety of freely available XML tools
 - Scales with the trace size
- **Model-based approach to trace analysis**
 - Can be integrated into XML-based approach (filter concept)
 - Takes concurrency into account
 - Finds non-reproducible faults



Software &
Engineering
Development
Techniques