Tracing and Monitoring Tools for Distributed Multi-Core Systems Project
State of the Art Meeting

Trace Abstraction and Correlation Track

Abdelwahab Hamou-Lhadj and Waseem Fadel
Concordia University
{abdelw, w_fadel}@ece.concordia.ca

Ecole Polytechnique, Montreal, QC
December 10th, 2009

Agenda

- Introduction
- What is and Why Trace Abstraction?
- Review of Existing Trace Abstraction Techniques
- Proposed Trace Abstraction Approach
- Conclusion and Future Direction

Agenda

- **Introduction**
- **What is and Why Trace Abstraction?**
- Review of Existing Trace Abstraction Techniques
- Proposed Trace Abstraction Approach
- Conclusion and Future Direction

Introduction

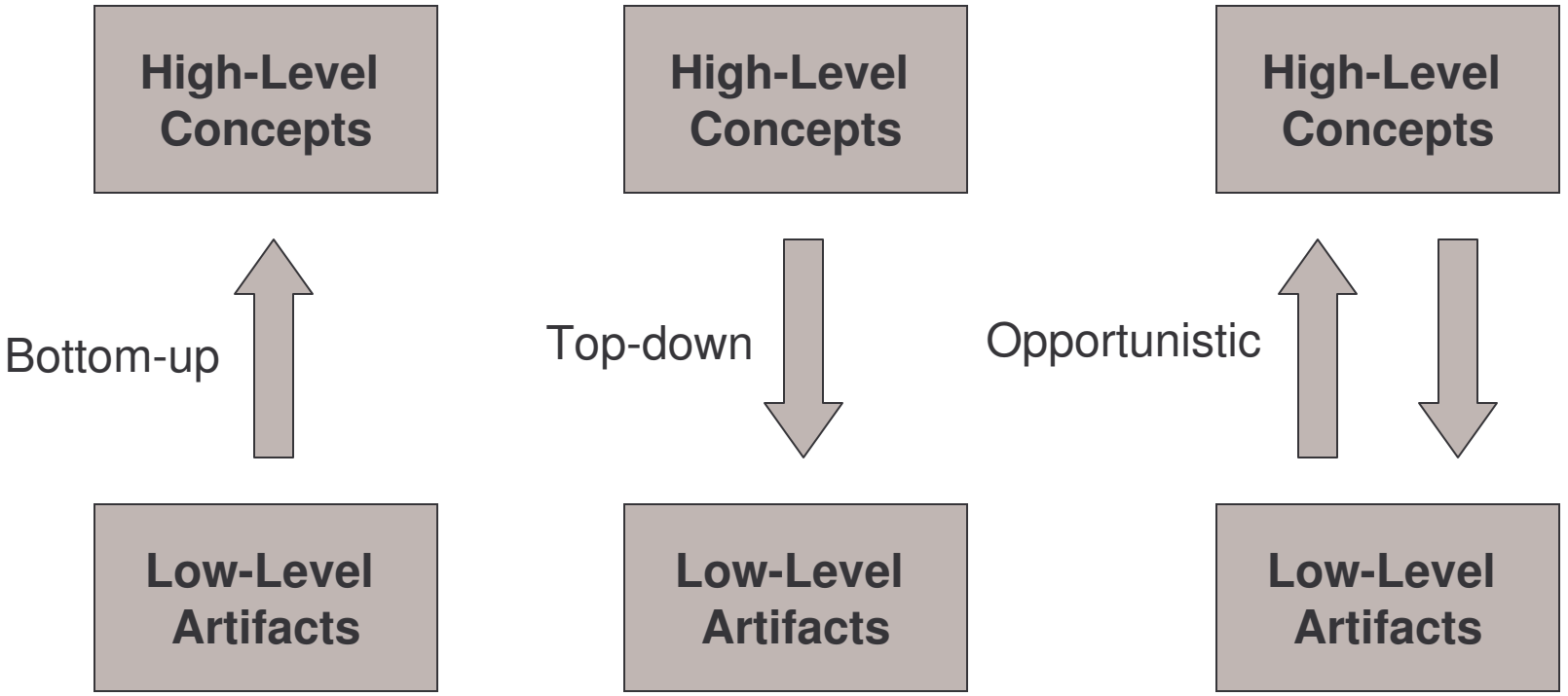
- Exploration of execution traces can help in a variety of applications such as:
  - Understanding why an unexpected behavior occurred (design faults, attacks,…?)
  - Understanding how a particular feature is implemented
  - Detecting causes of performance bottlenecks
  - Comparing traces from multiple versions of the system
  - Etc.
- Traces, however, tend to be excessively large and hard to understand.
  - Especially low-level, event-based, system call traces!

What is Trace Abstraction?

- A way to reduce the size of traces by abstracting out their main content
  - Two traces may look different but tell the same story
    - We are interested in the story and not the details
- The process of extracting high-level concepts from low-level trace events to facilitate the understanding and analysis of trace content

How People Understand System Artifacts?

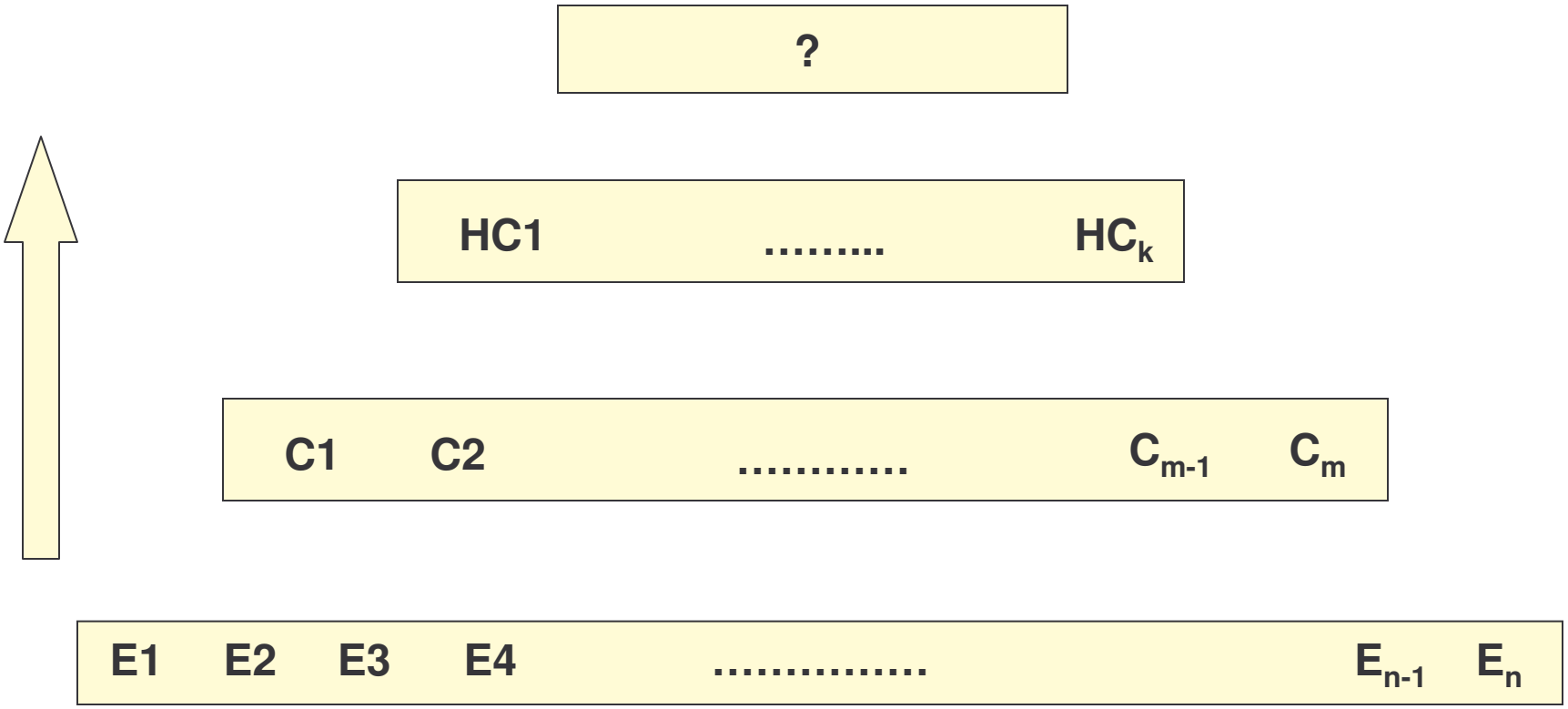
- Three strategies (based on empirical studies):



Trace Analysis: Trace Correlation

- Automatically comparing traces can help many applications:
  - Understanding how a system evolves by comparing traces of subsequent versions
  - Comparing traces for system health check
    - Important for detecting “zero-day” attacks
    - Security and self-healing systems
  - Very hard to do with low-level, event-based, system call traces!

Different Levels of Abstraction



Agenda

- Introduction and Motivations
- What is Trace Abstraction?
- **Review of Existing Trace Abstraction Techniques**
- Proposed Trace Abstraction Approach
- Conclusion and Future Direction

Review of Existing Trace Abstraction Techniques

- We surveyed several trace analysis techniques and tools in various areas:
  - Performance analysis, development and debugging, software maintenance and program comprehension
- Examples of tools that have been surveyed:
  - Low-level trace analysis tools:
    - LTTV, Intel VTune, SystemTap, WindRiver Workbench, Zealcore System Debugger, etc.
  - High-Level trace analysis tools
    - ISVis, Jinsight, Ovation, SEAT, AVID, Scene, Shimba, Program Explorer, Collaboration Browser, AVID, OSE, TPTP, VET

Key Trace Abstraction Techniques

- Pattern detection
- Filtering of noise
- Sampling
- Visualization Techniques

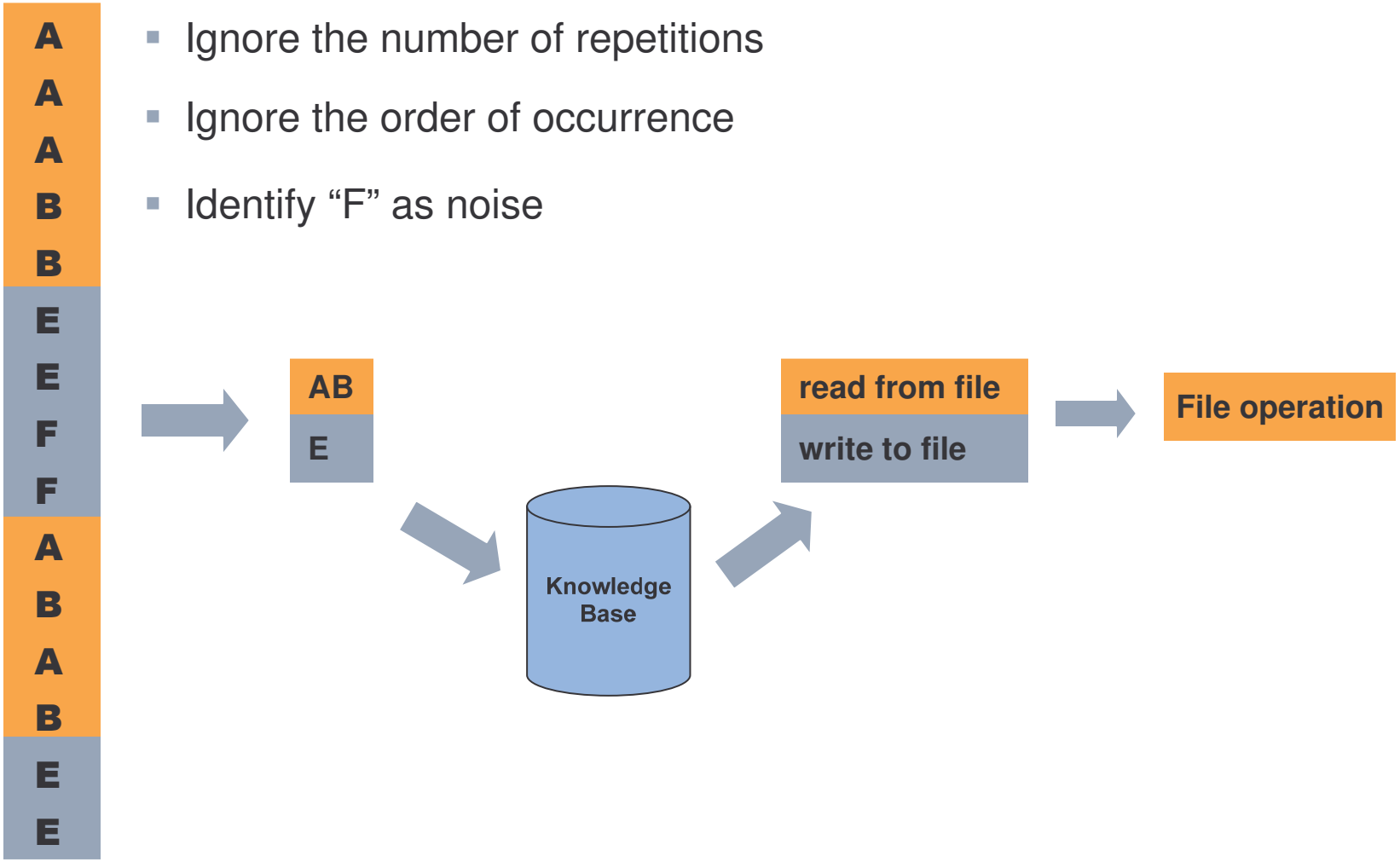
Pattern Detection Techniques

- A trace pattern is defined as a sequence of events that occurs repetitively but non-contiguously in several places in the trace.
- The more patterns in a trace, the less time is required to understand its content
  - We do not need to understand the same sequence twice!

More about Patterns

- Instances of the same pattern do not need to be identical
  - In fact, exact matching never leads to good abstraction!
  - Matching criteria need to be defined to enable generalization of a trace content
- Ideally, an extracted trace pattern should correspond to an abstract concept
  - E.g. a user identifiable computation of some feature

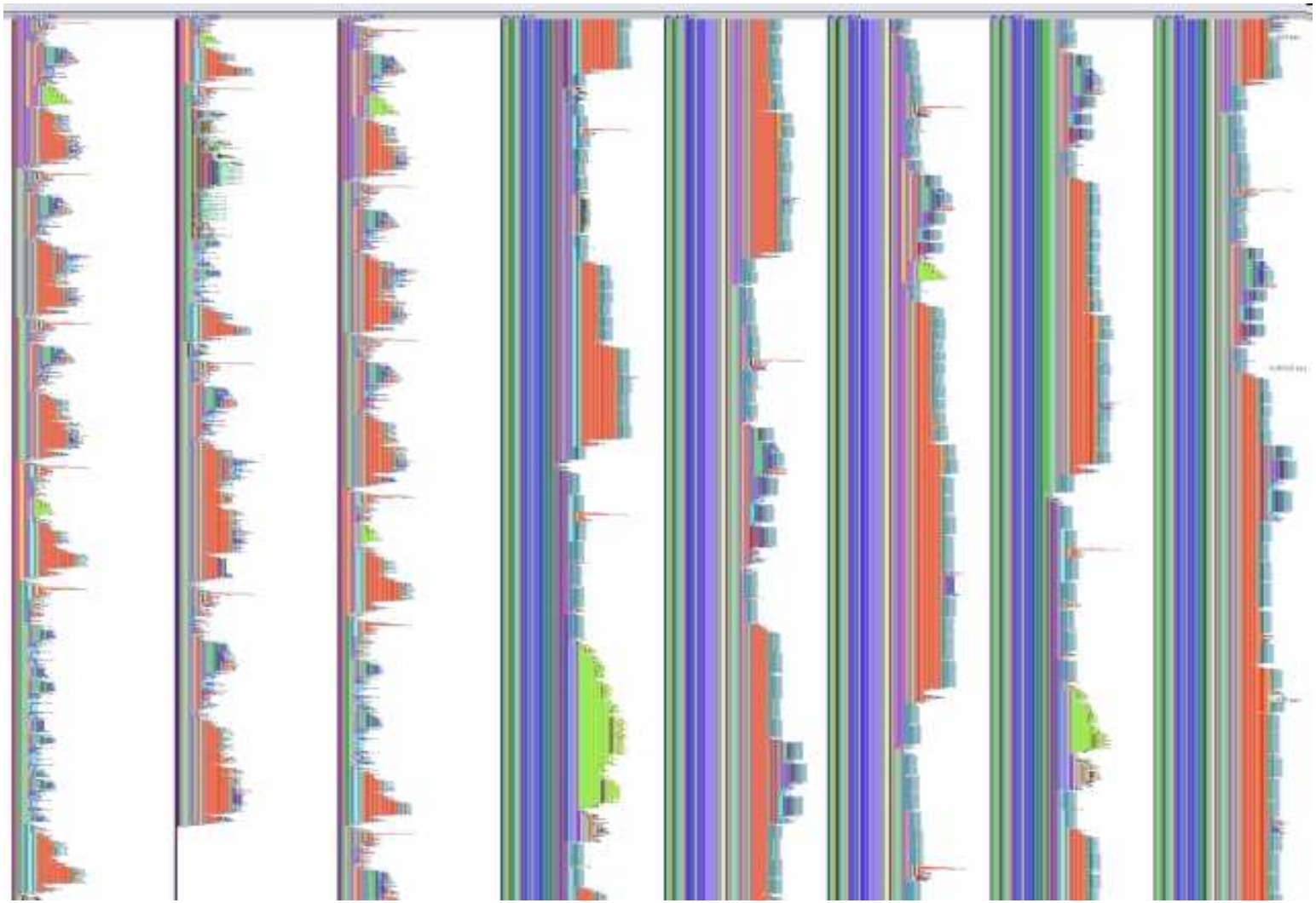
Example of Using Patterns



- Ignore the number of repetitions
- Ignore the order of occurrence
- Identify “F” as noise

Jinsight Pattern View (DePauw et al. 2004)

Part of IBM Rational® App. Developer for WebSphere® Software.



Pattern Matching Criteria

- Many pattern matching criteria have been proposed
  - Most of them can be applied to system call traces
  - De Pauw et al., Jerding et al., Richner et al., Systä et al, Hamou-Lhadj and Lethbridge, Bennett et al., etc.
- Examples:
  - Ignoring number of repetitions
  - Ignoring the order of occurrences
  - Treating a sequence of events as a set
  - Ignoring event information
  - Measuring the distance between two sequences
  - Etc.

Pattern Detection in Practice

- Various matching criteria have been used successfully in various studies:
  - Locate places in the system where enhancements are needed (Jerding et al.)
  - Helping debugging and fiingx system defects (Systä et al.)
  - Extracting component collaboration from traces (Richner et al.)
  - Recovery of high-level diagrams from traces (Hamou-Lhadj and Lethbridge)
  - Locating causes of performance bottlenecks (De Pauw et al.)

Detection and Filtering of Noise

- Traces often contain elements (noise) that are not needed at higher levels of abstraction
- What can be considered as noise depends on the objective of the analysis and the type of traces
  - LTTng traces contain many low-level memory management events that may not be needed at a high level
  - An example of noise in routine call traces are utilities and implementation details

Techniques for Detecting Noise

- Detection of noise can be user-guided or based on any available documentation
- A heuristic approach can also be used:
  - Frequency of the events
  - Order of occurrence
  - Dependency relationship

Sampling

- Sampling is also used to reduce the size of traces during its generation
- Several sampling criteria that can be used to generate small traces (Walker et al., Kuhn et al.)
  - Time-based sampling
  - Event-based sampling
  - Heuristic-based sampling
- The challenge is to find adequate sampling parameters

Trace Abstraction Based on Visualization Techniques

- Many tools have been developed to help analysts study execution traces
  - A set of features that visualize the traces and enable user interactions
  - Features categorized into:
    - Presentation Features: How trace is displayed?
    - Interaction Features: What can the user do with the system?

Presentation Features

- **Layout:** Defining the standard through which a trace is laid out (lines, points, graphs, etc.)
- **Multiple Linked Views:** Providing a number of views that are linked together
- **Visual Attributes:** Using colors and shapes
- **Labels:** Labeling events
- **Animation:** Supporting animation

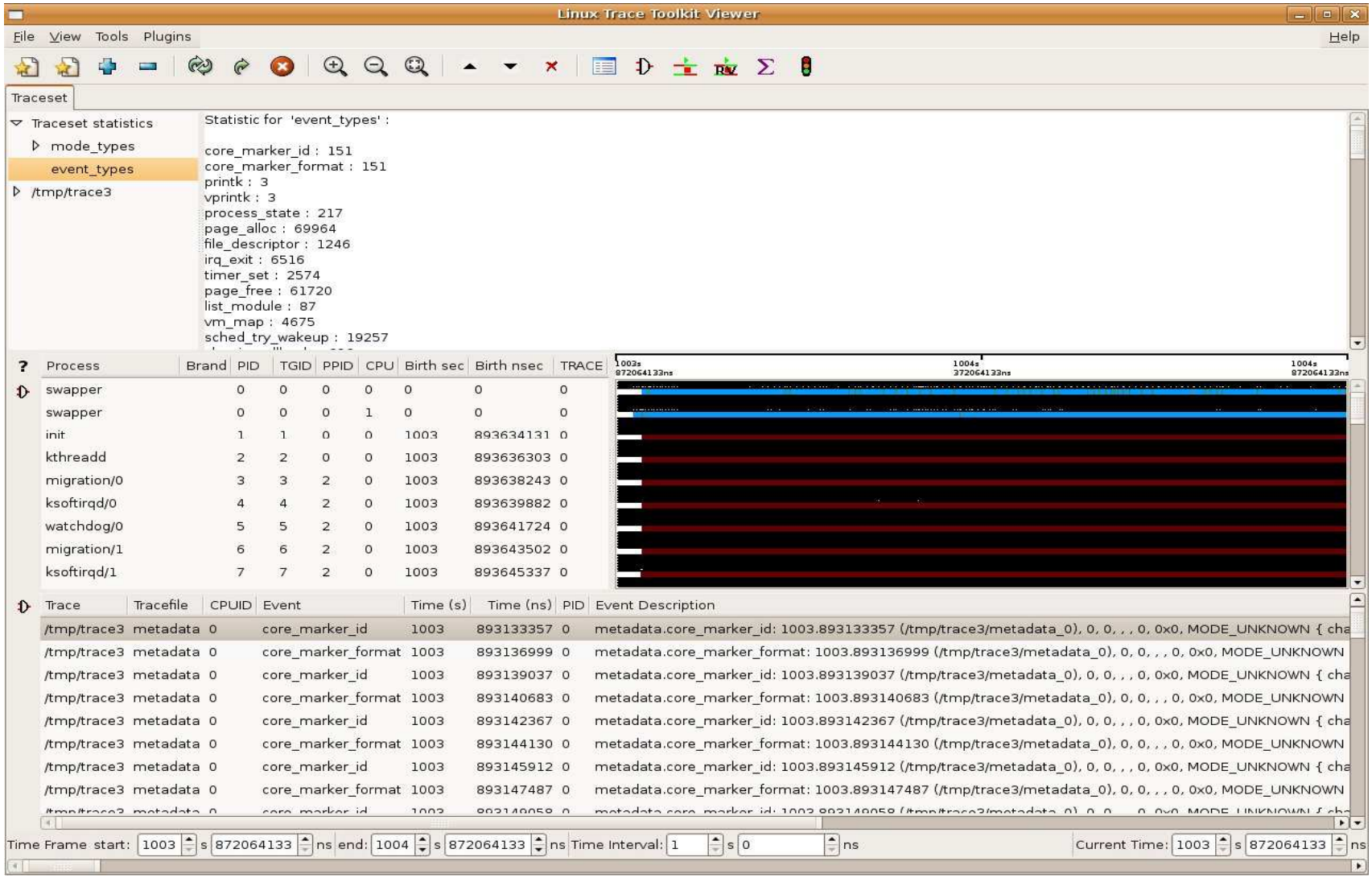
Interaction Features

- **Selection:** Selecting elements to manipulate, filter, or slice
- **Component Navigation:** Navigating between components and instances
- **Focusing:** Providing techniques such as: collapsing, partitioning, etc.
- **Zooming and Scrolling:** Enlarging or reducing the size of the event stream, moving up, down, left or right
- **Querying and Slicing:** filtering information, and selecting parts related to the selected component
- **Grouping:** Grouping objects, messages, repeated patterns
- **Annotating:** Describing grouped components, to store user notes while exploring the diagram
- **Highlighting:** Highlighting parts of the event sequence
- **Hiding:** Providing the ability to hide information

LTTV: Linux Trace Toolkit Viewer

- LTTV is provided to help studying the trace by visualizing it and providing a number of views:
  - The Statistic View displays statistics about the trace, the events' types, the processes, and the CPU
  - The Control Flow View provides an overall view of the trace, which helps developers to detect patterns that are recognized as lines with similar lengths and colors
  - The Detailed Event List View displays the list of events related to each process, like entry or exit events

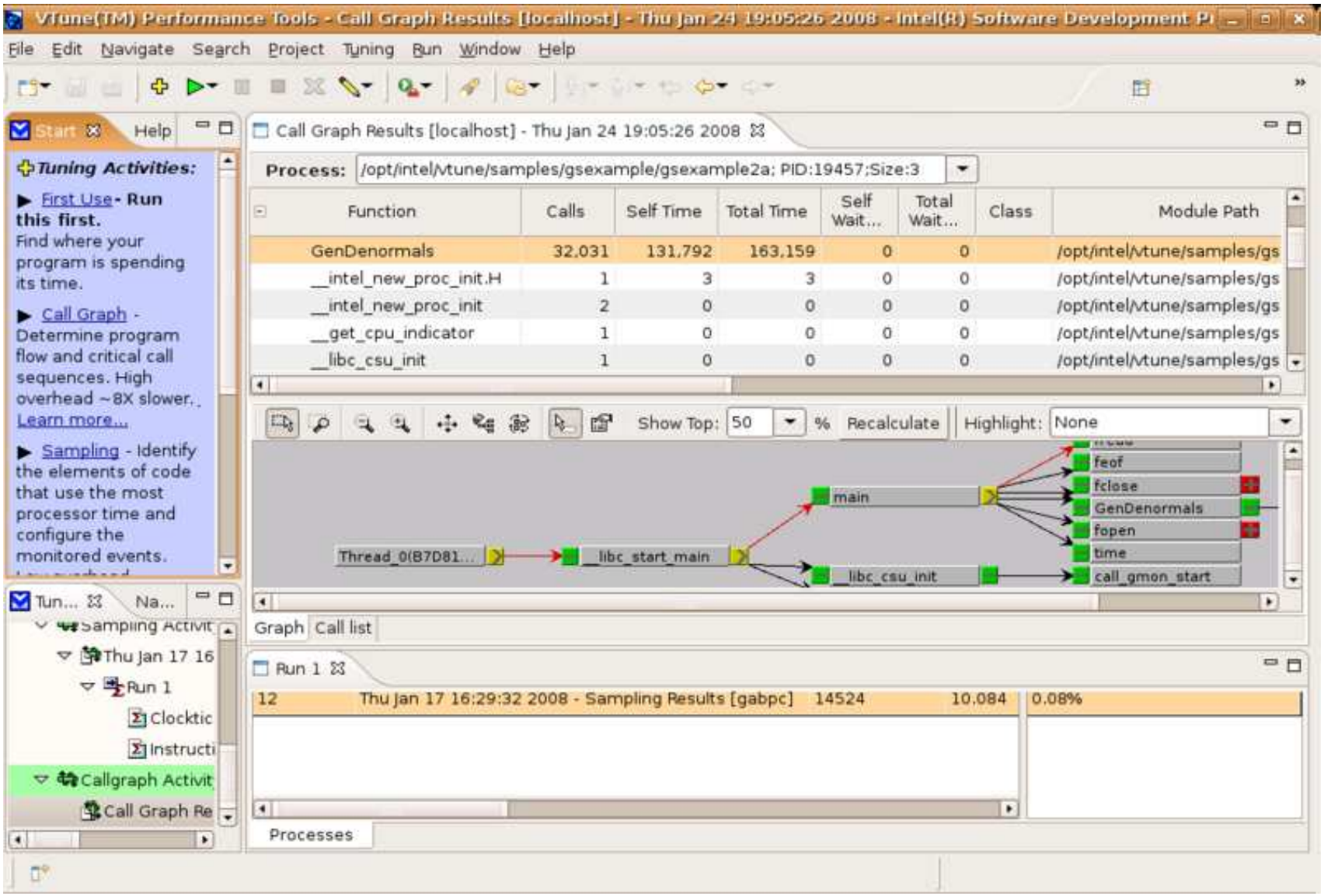
LTTV Screenshot



Intel VTune Performance Analyzer

- A profiler tool for applications running on intel-based systems
- Collects information through sampling and profiling
  - Sampling is accomplished by interrupting the processor at regular intervals and collecting samples of instruction addresses
- Provides a number of views like process view, thread view, module view, function view, view by call site and the critical path view

Intel VTune Performance Analyzer (cont.)



Intel VTune (cont.)

- To deal with size explosion problem:
  - Sampling, both time-based sampling and event-based sampling
  - It provides a number of visualization techniques like scrolling, zooming, highlighting, and filtering (by process id, CPU number ..etc)

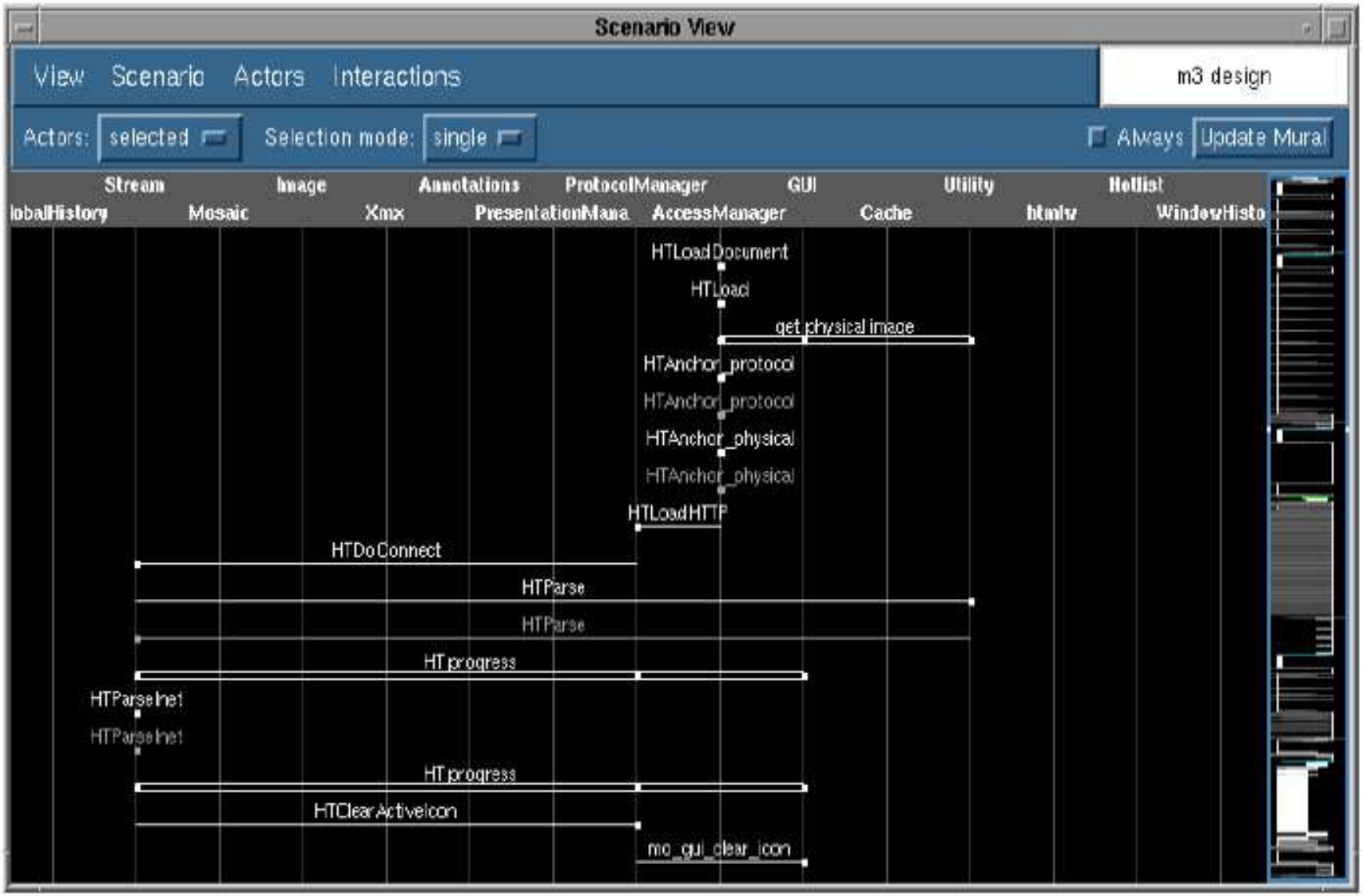
Wind River Workbench

- Wind River Linux uses the LTTng framework as a data provider
- Provides a set of tools and views for software development and debugging
- System Viewer displays the trace graphically in three different ways:
  - The Event Graph: Displays the succession of events relative to each thread
  - The Event Table: Displays events as rows of information ordered by their time stamp
  - The Memory usage graph displays memory allocation and deallocation

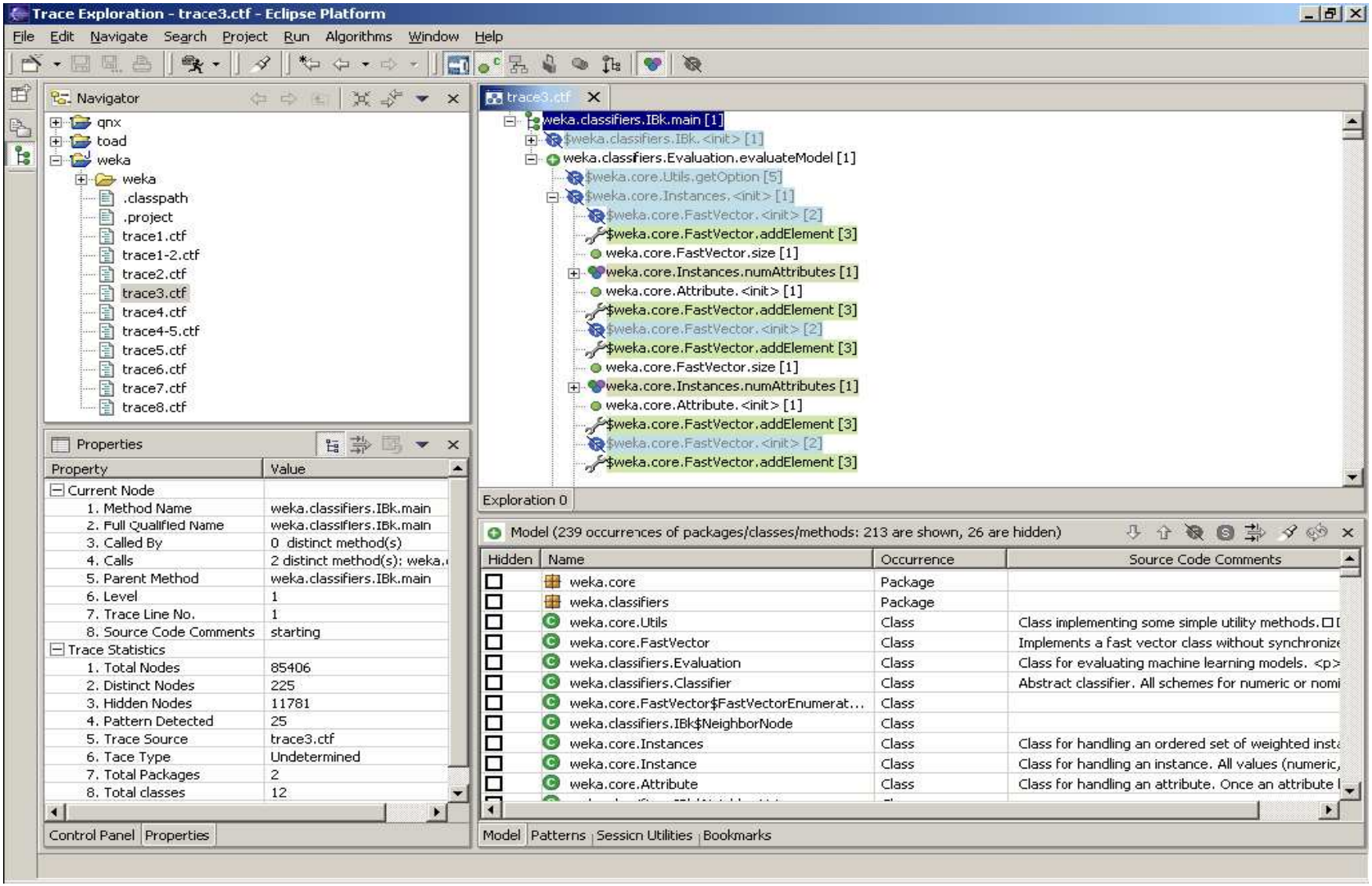
Wind River Workbench (cont.)

- To deal with size explosion problem:
  - Custom filtering is provided
  - Highlighting and selection
  - Multiple linked views
  - User-guided filtering

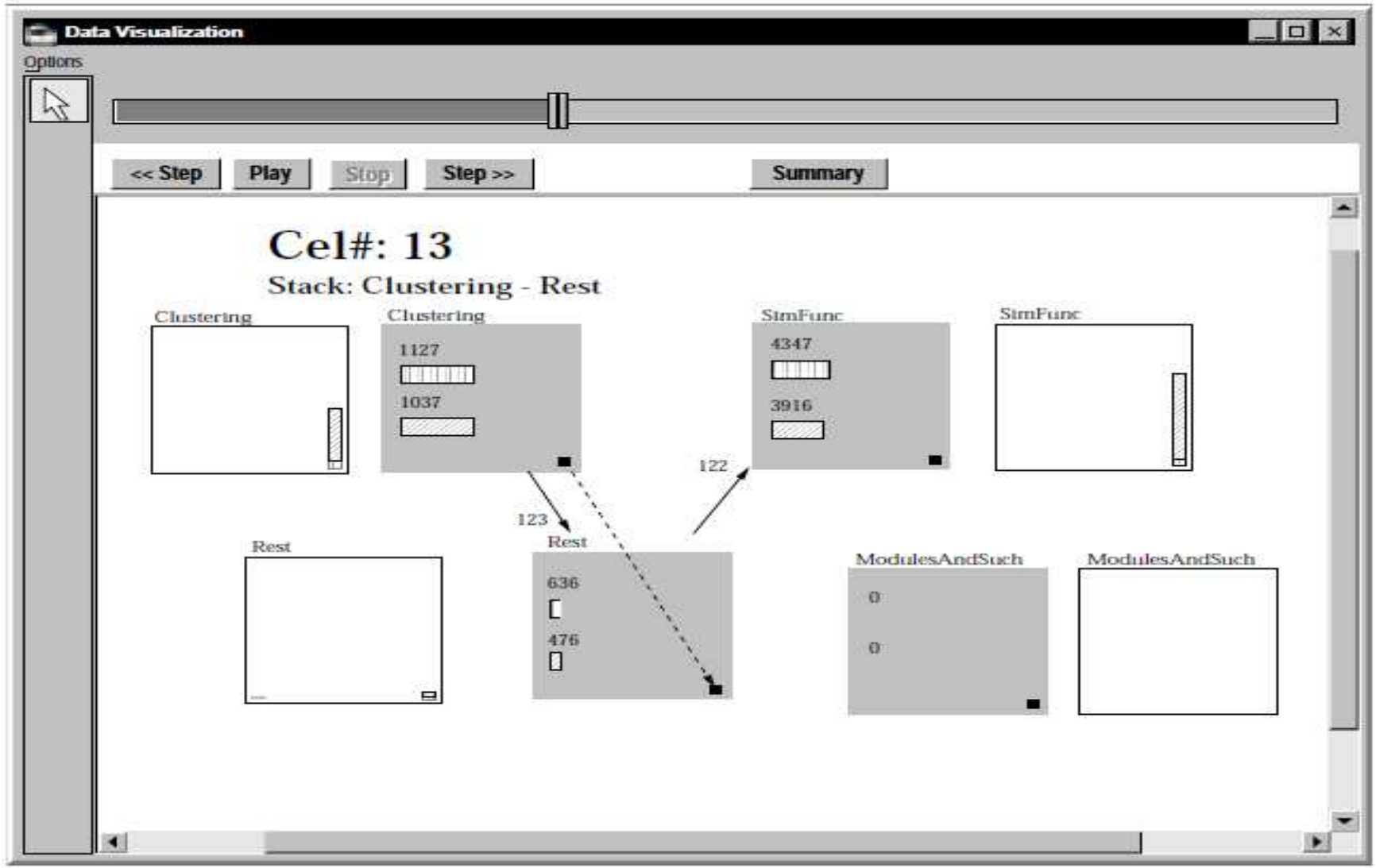
ISVis (Jerding et al. 97)



SEAT: Software Exploration and Analysis Tool (Hamou-Lhadj and Lethbridge)



AVID: Architecture Visualization of Dynamics in Systems (Walker et al. 2000)



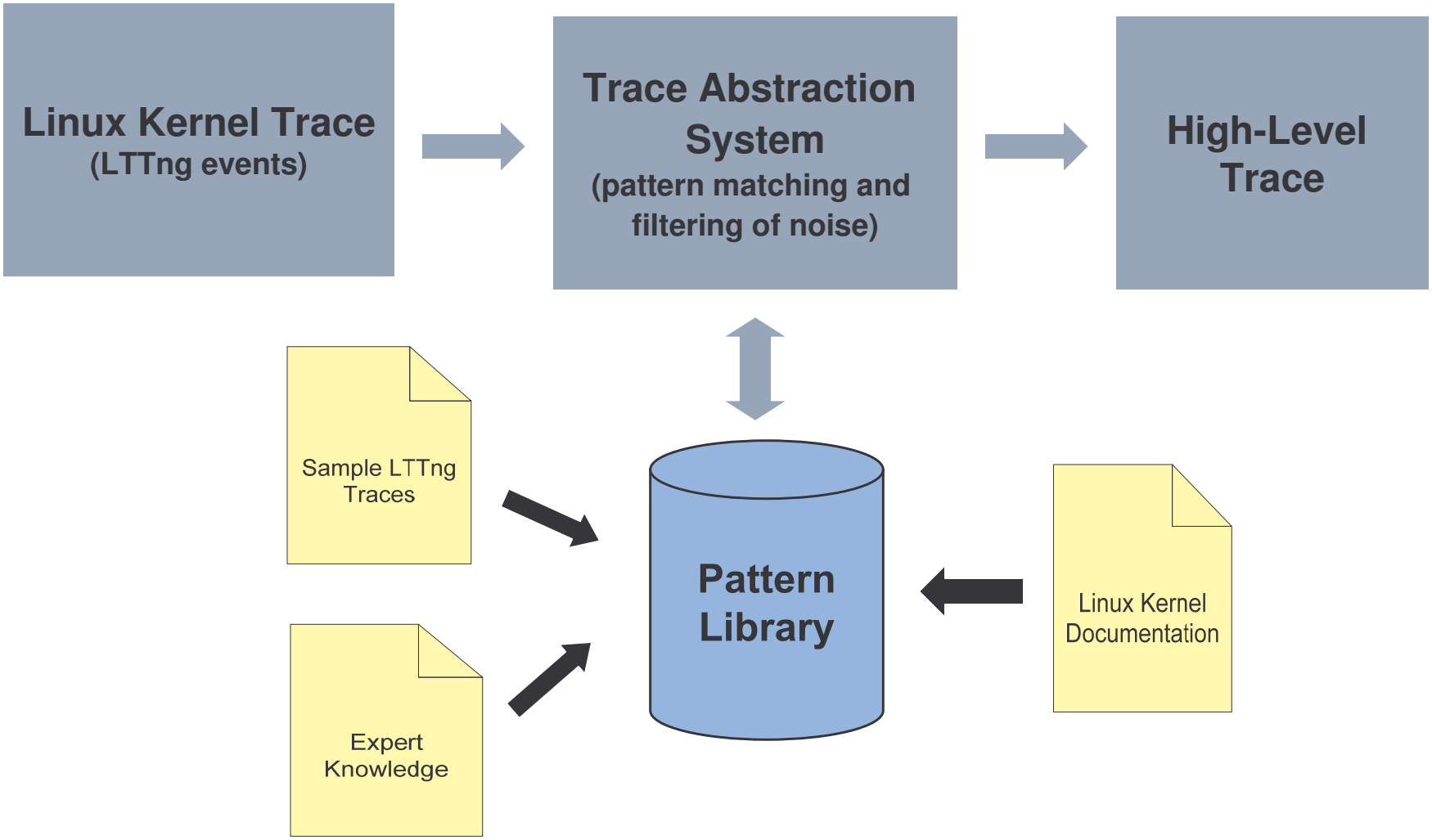
Limitations of Existing Work

- Require extensive user involvement
- Description of high-level concepts is provided by the users
- Many don't scale up
- Some of them are not applicable to low-level traces such as system call traces
- Require fine tuning of parameters and thresholds (yet to be determined)

Agenda

- Introduction
- What is and Why Trace Abstraction?
- Review of Existing Trace Abstraction Techniques
- **Proposed Trace Abstraction Approach**
- Conclusion and Future Direction

Approach for Abstracting LTTng Traces



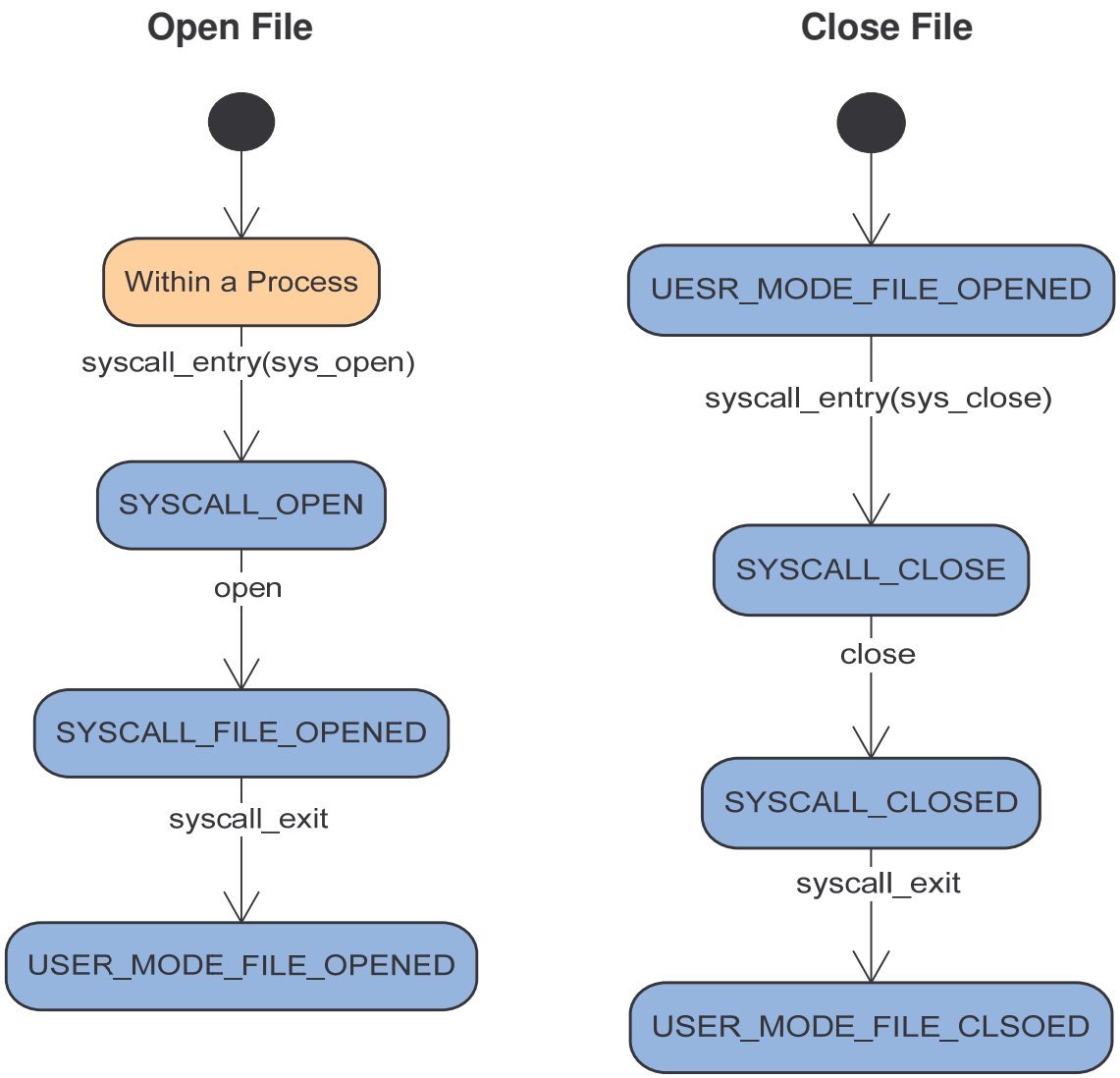
Pattern Library

- We have built a pattern library that contains patterns that represent key Linux kernel operations
  - File, socket and process management operations
- The patterns are modeled as UML state diagrams:
  - States represent system modes (user_mode, sys_call mode, etc.)
  - Events consist of LTTng events

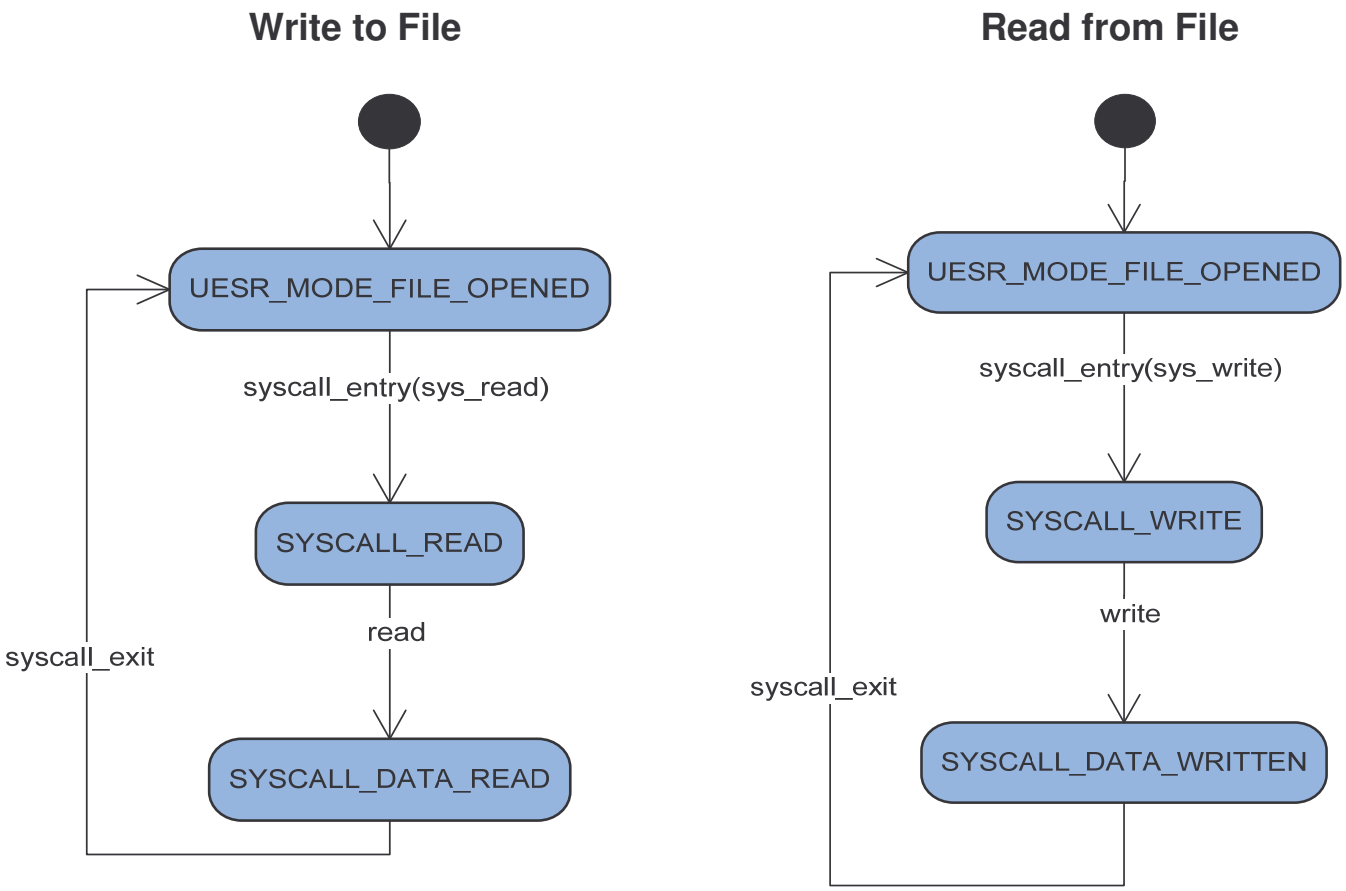
Patterns we have so far

- We created patterns for the following operations:
  - File Management (Open, Read, Write, Seek, Close)
  - Socket Management for both TCP and UDP (Create, Connect, Bind, Listen, Accept, Send, Receive, Close)
  - Process Management (Execution with exec and execve, Exit, Fork, Clone)

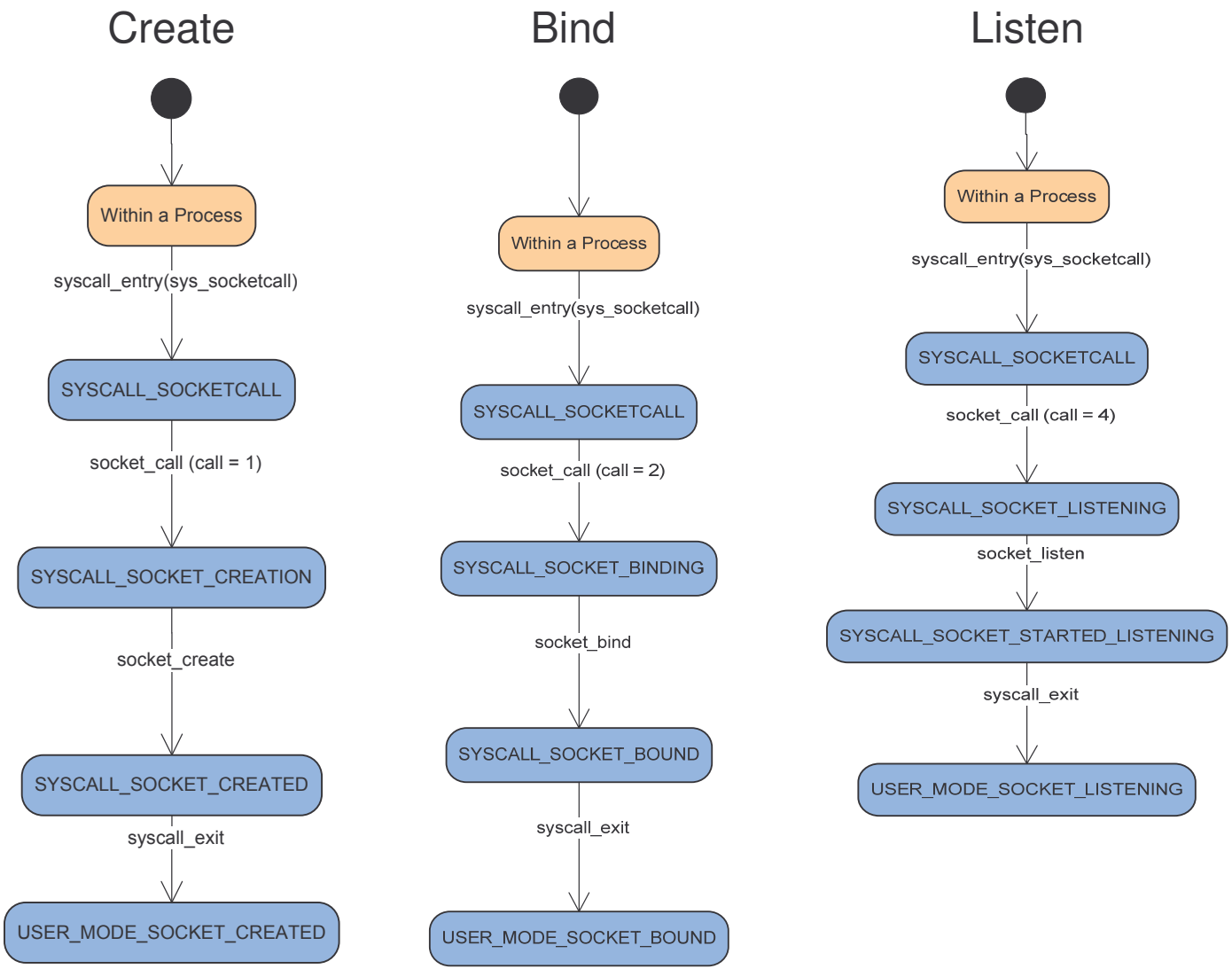
File Management: Open & Close



File Management: Read and Write

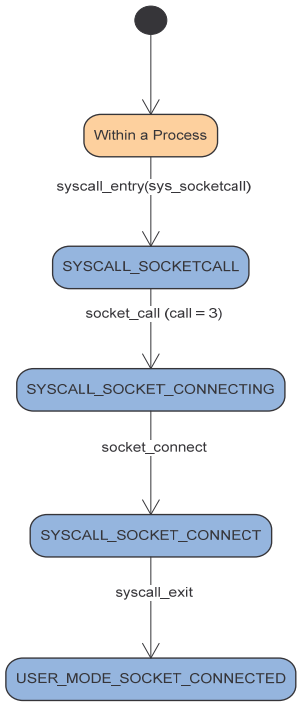


Socket Management (1)

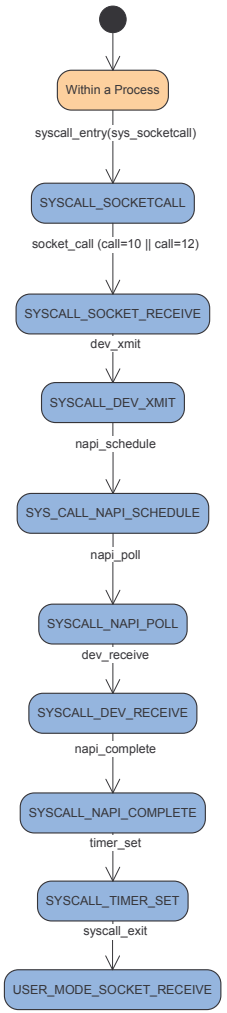


Socket Management (2)

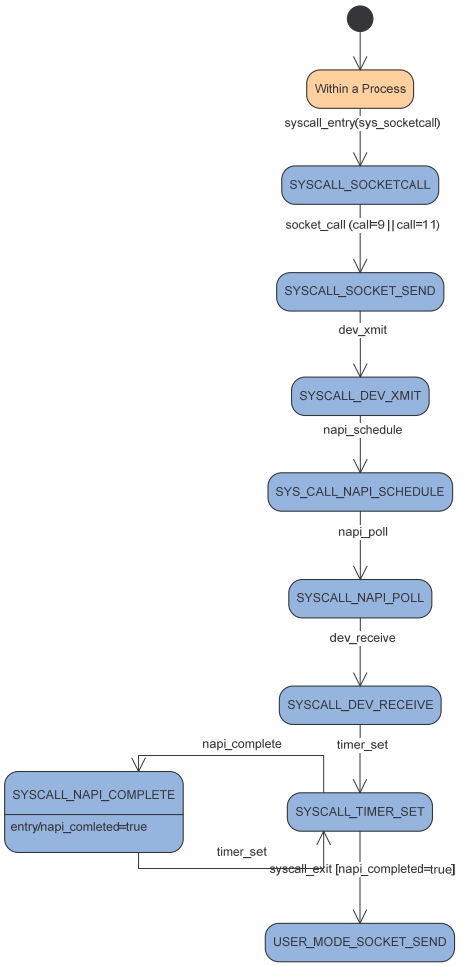
Connect



Receive

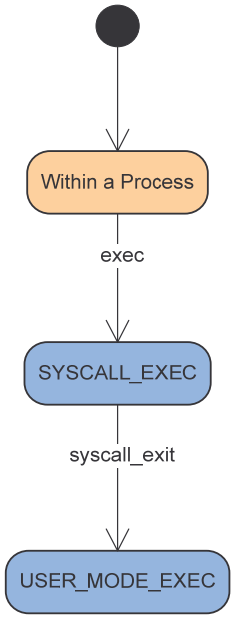


Send

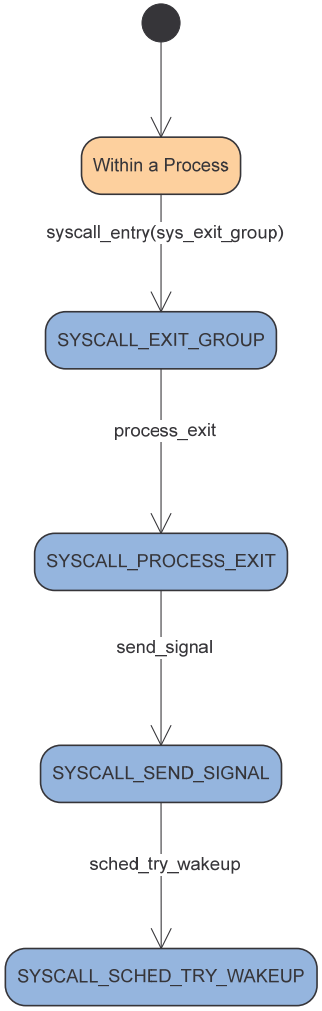


Process Management

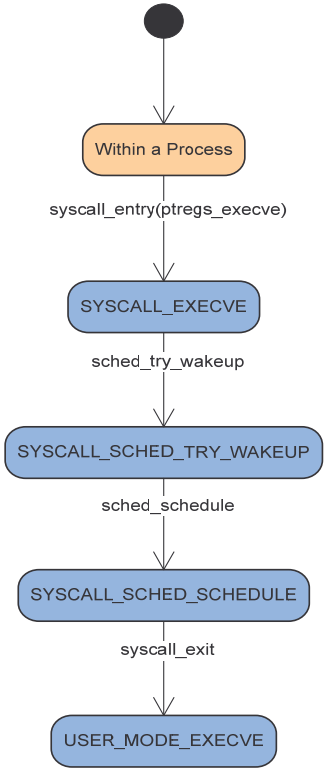
Execution with exec



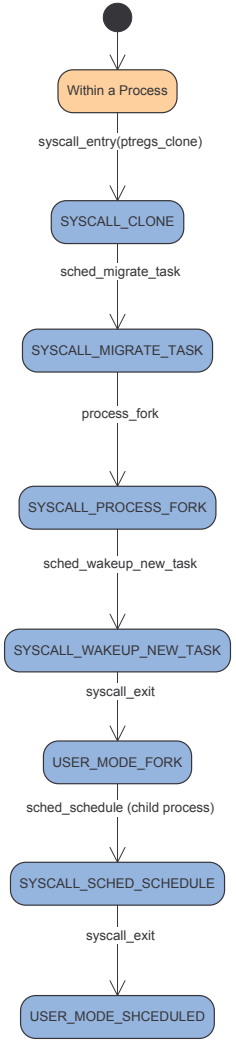
with execve



Exit



Cloning



Filtering of Noise

- We define noise in an LTTng trace as any event associated with memory management, page faults, and interrupts
  - can occur anywhere in the trace and in any order
  - are treated similarly to the way utilities have been treated in related work
- Associated events are treated as a set
  - i.e. order of occurrence of detailed events is ignored

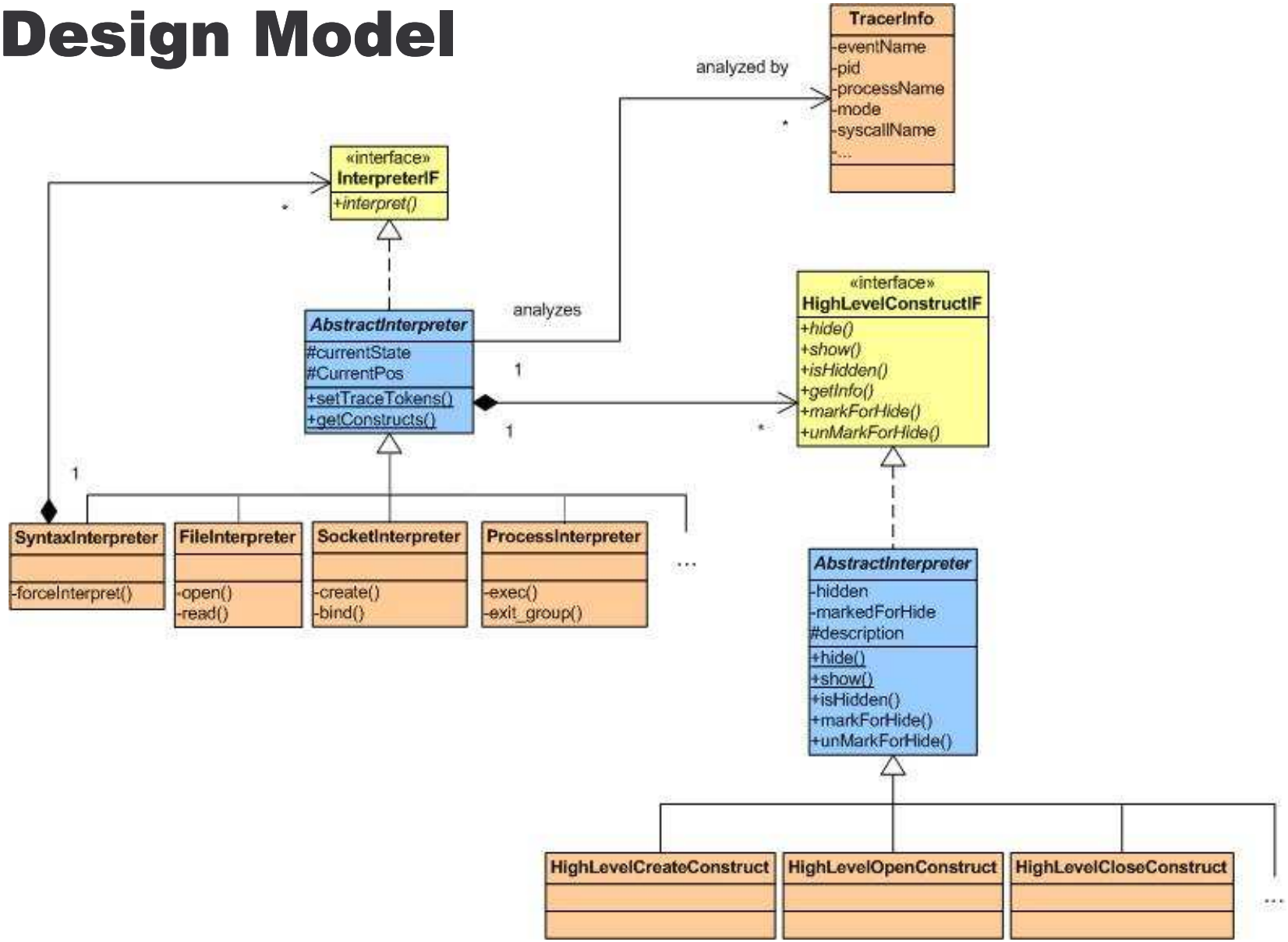
Validation of the Patterns

- We worked Pierre-Marc Fournier and Mathieu Desnoyers from École Polytechnique de Montréal to validate the patterns
  - Regular meetings with them have also helped in the process of understanding the markers used by LTTng
- Both users agreed with the way we defined noise found in traces
  - But further and more formal validation is needed

LTTng Trace Abstraction Tool

- We have built a prototype tool that takes an LTTng trace as inputs and return a more abstract trace by replacing
- Key characteristics of the tool:
  - Adding new patterns can easily be done
  - Multiple implementations representing different trace formats can be applied using the same interfaces
  - Noise interpreters are marked with the NoiseIF interface
  - High-level constructs can be hidden or shown easily by marking-unmarking them

Design Model



Some Preliminary Experiments

- We generated LTTng traces from small programs
  - Generated traces for the targeted processes contain around 1000 events
  - We were able to reduce the size of these traces to around 35 events
- We need to work on experimenting with larger traces (hundred of thousands of events)
  - From industrial systems with multiple processes

Agenda

- Introduction
- What is and Why Trace Abstraction?
- Review of Existing Trace Abstraction Techniques
- Proposed Trace Abstraction Approach
- **Conclusion and Future Direction**

Conclusion

- Trace abstraction is needed to make use of traces in an effective manner
- There are several techniques that have been proposed:
  - Pattern detection, filtering of noise, sampling, visualization
- We proposed a knowledge based approach to abstract out LTTng traces
  - A pattern library for main Linux Kernel operations has been created

Future Directions

- Continue developing and validating Linux sys calls patterns
- Start experimenting with large traces
- Completing
  - A paper that compares trace analysis tools
  - A paper on abstracting system call trace
    - The overall approach
    - The pattern library
    - Proof of concept

Thank You!
Questions and Discussion

References

- A. Hamou-Lhadj and Timothy Lethbridge. Reasoning about the Concept of Utilities. ECOOP PPPL, Oslo, Norway, June 14, 2004
- A. Hamou-Lhadj and Timothy Lethbridge. Compression Techniques to Simplify the Analysis of Large Execution Traces. In Proc. of the 10th International Workshop on Program Comprehension (IWPC), pages 159-168, Paris, France, 2002
- A. Hamou-Lhadj and Timothy Lethbridge. Measuring Various Properties of Execution Traces to Help Build Better Trace Analysis Tools. In Proc. of the 10th International Conference on Engineering of Complex Computer Systems, IEEE Computer Society, pages 559–568, 2005
- Adrian Kuhn and Orla Greevy. Exploiting the Analogy between Traces and Signal Processing. In Proc. of IEEE International Confernce on Software Maintainance (ICSM 2006). IEEE Computer Society Press: Los Alamitos CA, 2006
- Andrew Chan, Reid Holmes, Gail C. Murphy and Annie T.T. Ying. Scaling an Object-oriented System Execution Visualizer through Sampling. In Proc. of the 11th IEEE International Workshop on Program Comprehension (IWPC'03), 2003
- A. Hamou-Lhadj and Timothy Lethbridge. Summarizing the Content of Large Traces to Facilitate the Understanding of the Behaviour of a Software System. In Proc. 14th Int. Conf. on Program Comprehension (ICPC), pages 181–190. IEEE, 2006
- Bas Cornelissen, Leon Moonen, and Andy Zaidman. An Assessment Methodology for Trace Reduction Techniques

References (cont.)

- Robert J.Walker, Gail C. Murphy, Bjorn Freeman-Benson, DarinWright, Darin Swanson, and Jeremy Isaak. Visualizing Dynamic Software System Information through High-level Models. In Proc. of the Conference on Object-Oriented Programing, Systems, Languages, and Applications (Vancouver, British Columbia, Canada; 18–22 October 1998), ACM SIGPLAN, pp. 271–283, 1998. Published as ACM SIGPLAN Notices, 33(10), October 1998
- A. Hamou-Lhadj and Timothy Lethbridge. An Efficient Algorithm for Detecting Patterns in Traces of Procedure Calls. In Proc. of the 1st International Workshop on Dynamic Analysis (WODA), May 2003
- A. Hamou-Lhadj and Timothy Lethbridge. Techniques for Reducing the Complexity of Object-Oriented Execution Traces. In Proc. of VISSOFT, 2003, pp. 35-40
- A. Hamou-Lhadj and Timothy Lethbridge. A Survey of Trace Exploration Tools and Techniques. In Proc. of IBM Centers for Advanced Studies Conferences (CASON 2004). IBM Press: Indianapolis IN, 2004; 42–55
- A. Hamou-Lhadj. Techniques to Simplify the Analysis of Execution Traces for Program Comprehension. PhD Thesis, Ottawa-Carleton Institute for Computer Science, School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, Canada
- Dean Jerding and Spencer Rugaber. Using Visualization for Architectural Localization and Extraction. In Proc. of the 4th Working Conference on Reverse Engineering, October 1997, the Netherlands, IEEE Computer Society, pp. 56-65

References (cont.)

- A. Hamou-Lhadj. Techniques to Simplify the Analysis of Execution Traces for Program Comprehension. PhD Thesis, Ottawa-Carleton Institute for Computer Science, School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, Canada
- A. Hamou-Lhadj, Edna Braun, Daniel Amyot, and Timothy Lethbridge. Recovering Behavioral Design Models from Execution Traces. In Proc. of the 9th European Conference on Software Maintenance and Reengineering (CSMR'05) 2005
- Wim De Pauw, David Lorenz, John Vlissides, and MarkWegman. Execution Patterns in Object-Oriented Visualization. In Proc. of the 4th USENIX Conference on Object-Oriented Technologies and Systems, pp. 219–234, 1998
- Tarja Systä. Understanding the Behavior of Java Programs. In Proc. of the 7th Working Conference on Reverse Engineering, Australia, Brisbane, 2000, pp. 214-223
- Kai Koskimies and Hanspeter Mössenböck. Scene: Using Scenario Diagrams and Active Text for Illustrating Object-Oriented Programs. In Proc. of ICSE-18, pages 366 375. IEEE, Mar. 1996
- Tamar Richner and St´ephane Ducasse. Using Dynamic Information for the Iterative Recovery of Collaborations and Roles. In Proc. of the 18th International Conference on Software Maintenance (ICSM), pages 34-43, Montréal, QC, 2002
- Abdelwahab Hamou-Lhadj, Timothy C. Lethbrridge, Lianjiang Fu. SEAT: A Usable Trace Analysis Tool. In Proc. of the 13th International Workshop on Program Comprehension (IWPC'05) 2005

References (cont.)

- C. Bennett, D. Myers, M. A. Storey, D.M. German, D. Ouellet, M. Salois, and P. Charland. A Survey and Evaluation of Tool Features for Understanding Reverse Engineered Sequence Diagrams. Journal of Software Maintenance and Evolution: Research and Practice, March 2008
- Eclipse Documentation – Archived Release. Overview of the Java Profiling Tool. http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.tptp.platform.doc.user/tasks/tesqanac.htm
- Eclipse Documentation – Archived Release. Profiling Views. http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.tptp.platform.doc.user/tasks/tesqanac.htm
- Eclipse Documentation – Archived Release. Using the Execution Statistics View. http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.tptp.platform.doc.user/tasks/tesqanac.htm
- Eclipse Documentation – Archived Release. Method Invocation Tab. http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.tptp.platform.doc.user/tasks/tesqanac.htm
- Eclipse Documentation – Archived Release. UML2 Trace Interaction Views. http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.tptp.platform.doc.user/tasks/tesqanac.htm
- Mathieu Desnoyers and Michel R. Dagenais. Tracing for Hardware, Driver, and Binary Reverse Engineering in Linux. CodeBreakers Journal Vol. 1, No. 2, 2006