Tracing and Monitoring Distributed Multi-Core Systems

The State of the Art of Trace Abstraction Techniques – Progress Report

Waseem Fadel and Abdelwahab Hamou-Lhadj
{w_fadel, abdelw}@ece.concordia.ca
Department of Electrical and Computer Engineering
Concordia University

Ecole Polytechnique, Montreal, Quebec - Sept17, 2009

Agenda

- Introduction and Motivation
- Trace Abstraction Techniques
  - Pattern Detection
  - Utility Removal
  - Data Collection
  - Visualization Techniques
- Conclusion
- Future Work

Introduction and Motivation

- Software engineers need to explore execution traces for a variety of reasons such as:
  - Understand why an unexpected behaviour occurs
  - Understanding how a feature is implemented
  - Detecting performance bottlenecks
  - Studying the impact of making changes to a system
  - Run-time monitoring
  - Etc.

- Traces, however, tend to be **excessively large and hard to understand**
  - Trace abstraction and analysis tools are needed to help software engineers understand and analyze large traces

Trace Abstraction Techniques

- Objective:
  - To facilitate the understanding, exploration, and analysis of large traces by abstracting out their main content

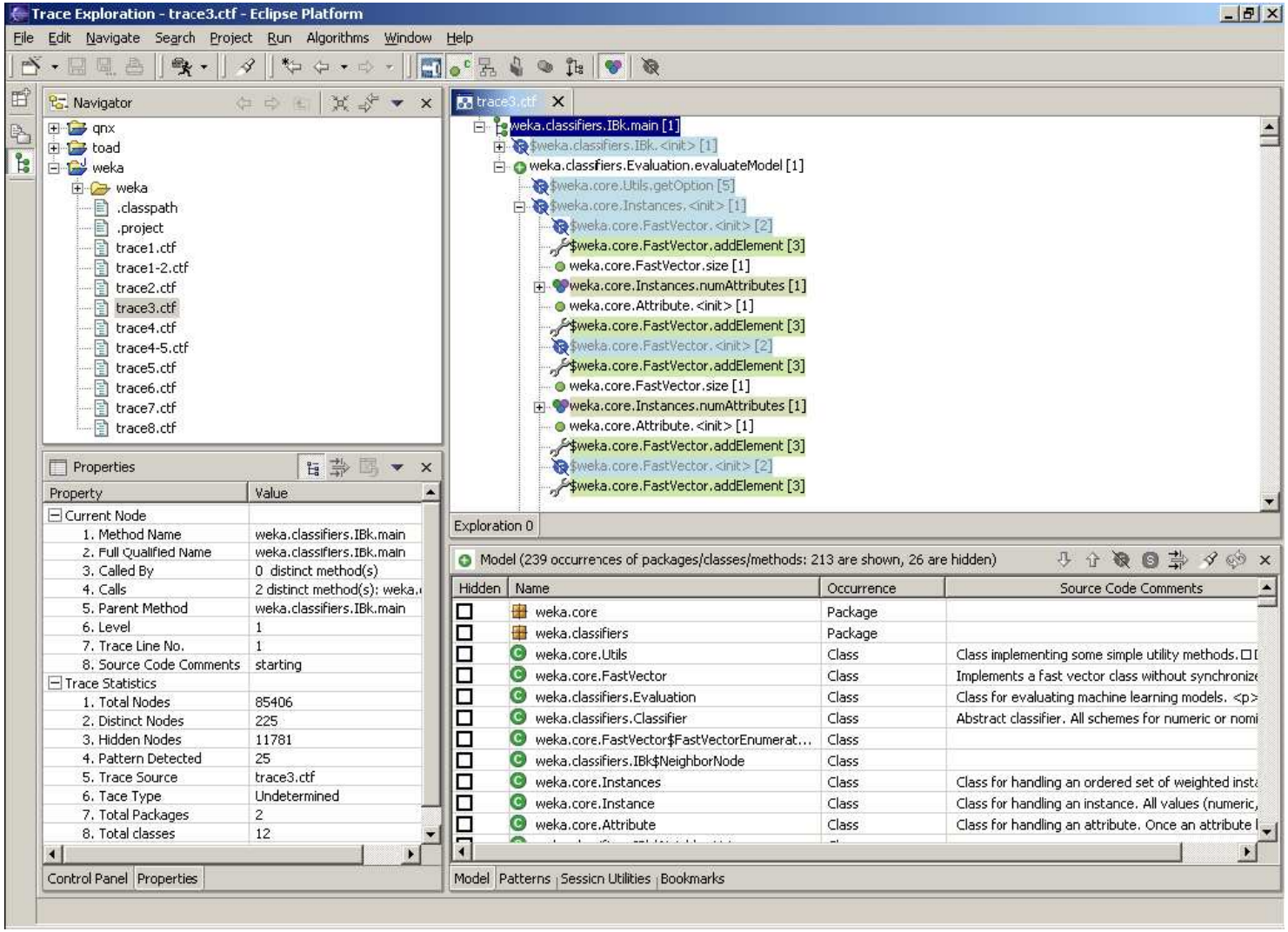
- Categories of trace abstraction techniques:
  - Pattern Detection
  - Utility Removal
  - Data Collection
  - Visualization Techniques

Pattern Detection Techniques

- We define a trace pattern 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
  - Software engineers do not need to understand the same sequence twice!

Example of Using Patterns in a Trace Analysis Tool



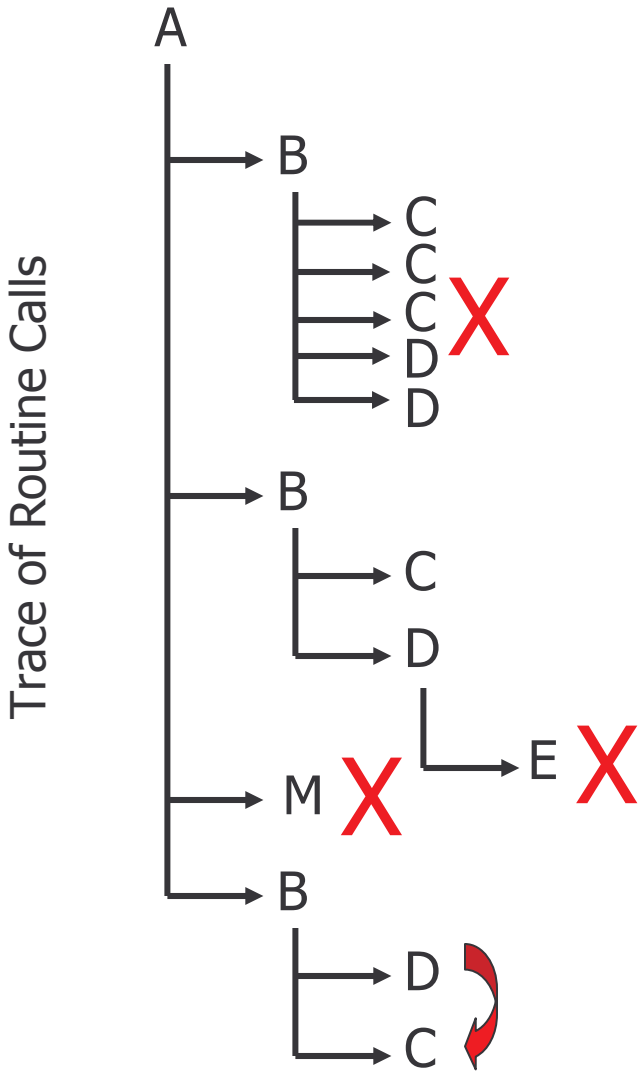
More about Patterns

- 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
  - But reality is far from the ideal!

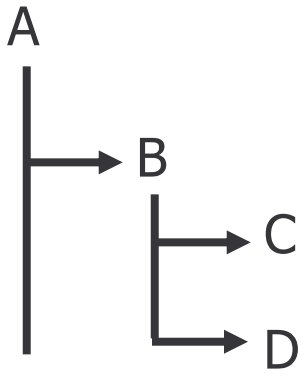
Pattern Matching Criteria

- Pattern matching criteria determine when two sequences of calls can be considered equivalent
- Users can select the criteria according to their knowledge of the system and the complexity of the trace
  - E.g. identical patterns might be useful to novices but less useful to experts.
- Many matching criteria have been proposed:
  - De Pauw et al., Jerding et al., Richner et al., Hamou-Lhadj and Lethbridge, etc.

Example of Pattern Matching Criteria



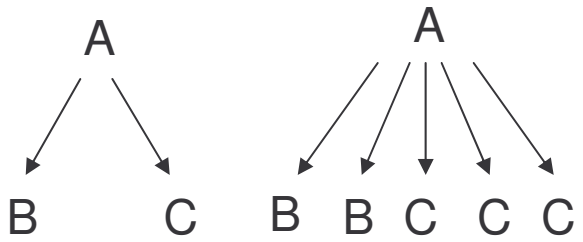
- Ignore number of contiguous repetitions
- Limit tree depth to 2
- Ignore order of calls
- Remove "m" considered as a utility



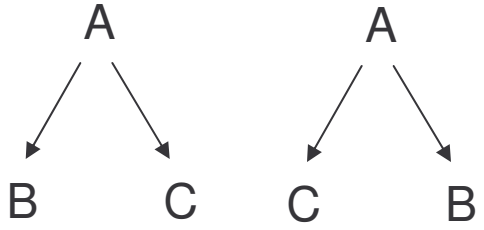
Review of Key Matching Criteria

- **Ignoring Node Labels:** C.obj1.m() and C.obj2.m() can be considered identical by ignoring the objects' names.

- **Ignoring Repetition:** repetition due to loops and recursions can be ignored when looking for patterns



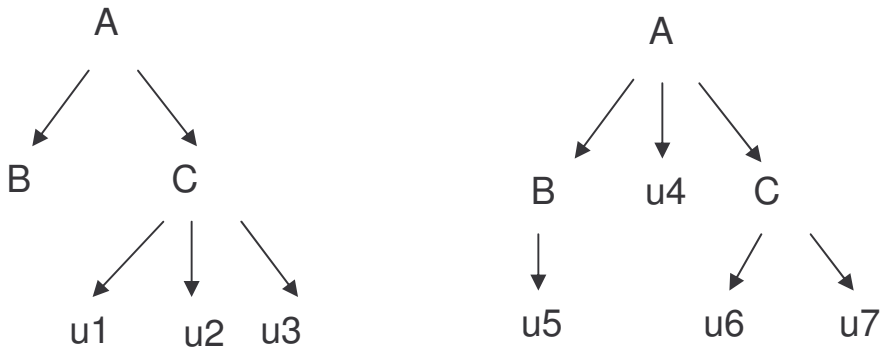
- **Ignoring Ordering:** Order of calls might not be important at some levels of the call tree.



Review of Key Matching Criteria (cont.)

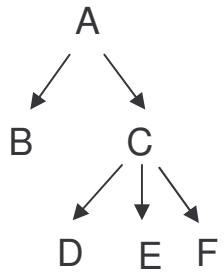
- **Depth-limiting**: allows comparing two subtrees up to a certain depth

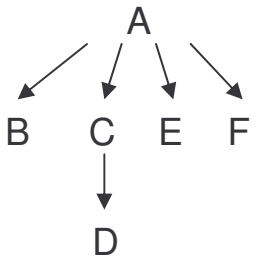
- **Filtering of Components**: ignoring some components (e.g., utilities) when comparing sequence of calls



Review of Key Matching Criteria (cont.)

- **Flattening:** Ignore the structure of the tree
  - Treating the sequence of calls as a set
  - Useful for experts who are not interested in the detailed call structure
- **Edit Distance:** The minimum number of operations required to transform one sequence into another





Matching Criteria in Practice

- Various matching criteria have been used successfully in various studies:
  - Architectural localization (Jerding et al.): Using traces to locate the right place in the source code where enhancements to the architecture of the system are needed
  - Design Recovery
    - Extracting component collaboration from traces (Richner et al.)
    - Recovery of high-level behavioural diagrams from traces (Hamou-Lhadj and Lethbridge)
  - Fixing defects (Systä et al.)
  - Detecting caused of performance bottlenecks (De Pauw et al.)

Limitations of Matching Criteria

- We need to validate the matching criteria and analyze at which level of the trace they can be applied usefully
- We also need to study how they can be combined
- It is also important to understand how the matching criteria are used by SW engineers
  - Will depend on the knowledge they have of the system and the complexity of the task at hand

Trace Abstraction Techniques

- Pattern Detection
- **Utility Removal**
- Data Collection
- Visualization Techniques

Utility Removal Techniques

- Hamou-Lhadj and Lethbridge introduced the concept of trace summarization, which is a process that
  - takes a trace as input and returns a summary of its main content as output
- Similar in principle to text summarization
- Key Applications:
  - Enable top-down analysis of execution traces
  - Recovery of high-level views of the system
  - Understanding how features are implemented

Selection of the Main Content

- Based on the removal of implementation details (including utilities) from traces
- A study was conducted at QNX to determine what SW engineers consider as a utility
- A utility:
  - Is something called from several places
  - Can be packaged in a non-utility module
  - Is used for implementation purposes
- Not all implementation details are utilities!

Utilityhood Metric

Hamou-Lhadj and Lethbridge, 2006

$$U(r) = \frac{Fanin(r)}{N} \times \frac{Log\left(\frac{N}{Fanout(r) + 1}\right)}{Log(N)}$$

- U(r) has 0 (not a utility) as its minimum and approaches 1 (most likely to be a utility) as its maximum
- Relies on a static call graph to measure fan-in and fan-out of each routine
- Based on extensive experimental studies
- An improvement to the metric was proposed by Rohatgi and Hamou-Lhadj and in which impact analysis is used instead of fan-in analysis

Trace Summarization Process

- Step 1: Set the parameters for the summarization process (exit condition, etc.)
- Step 2: Remove implementation details (constructors, accessing methods)
- Step 3: Detect and remove utility components
- Step 4: Output the result

Application of Trace Summarization

- The trace summarization was applied to a trace *that* initially contained **97413 routine calls** to successfully extract a summary from this trace that contained **453 calls**!

- The results were validated with the designers of the target system
  - The summary was confirmed to be an excellent representation of the main content of the trace

Trace Abstraction Techniques

- Pattern Detection
- **Utility Removal**
- **Data Collection**
- Visualization Techniques

Data Collection Based Abstraction Techniques

- The idea is to reduce trace size by considering only certain events
  - This results in smaller traces that can be further abstracted out
- Systä et al. proposed an approach that combines static and dynamic techniques to explore large traces
  - Static analysis is used to **select only parts** of the system that need to be analyzed using dynamic analysis
    - Based on information extracted from the user's request
  - Additional processing of the resulting trace using pattern detection techniques are applied to extract a high-level view of the trace

Sampling

- Sampling is another technique used often to reduce the size of traces during its generation
- Walker et al. proposed several sampling criteria that can be used to generate small traces. Examples include:
  - the events that appear after a certain timestamp only
  - a snapshot of the call stack every x^{th} event and so on
  - Etc.
- The challenge is to find the adequate sampling parameters for the understanding of a specific feature

Monotone Subsequence Summarization

- Kuhn and Greevy introduced the Monotone Subsequence Summarization (Grouping) technique
  - Starting from the first event, a group is built by adding events with the same or with higher nesting level
  - When facing a decrease greater than a threshold known as gap size, a new group is created
  - The result is groups of events identified by the first event. The events of each group are hidden in a repository for later exploration

Comparing Abstraction Techniques

- Cornelissen et al. conducted a comparison of four abstraction techniques:
  - Monotone Subsequence Summarization (Grouping)
  - Stack depth limitation
  - Language-based filtering
  - Sampling

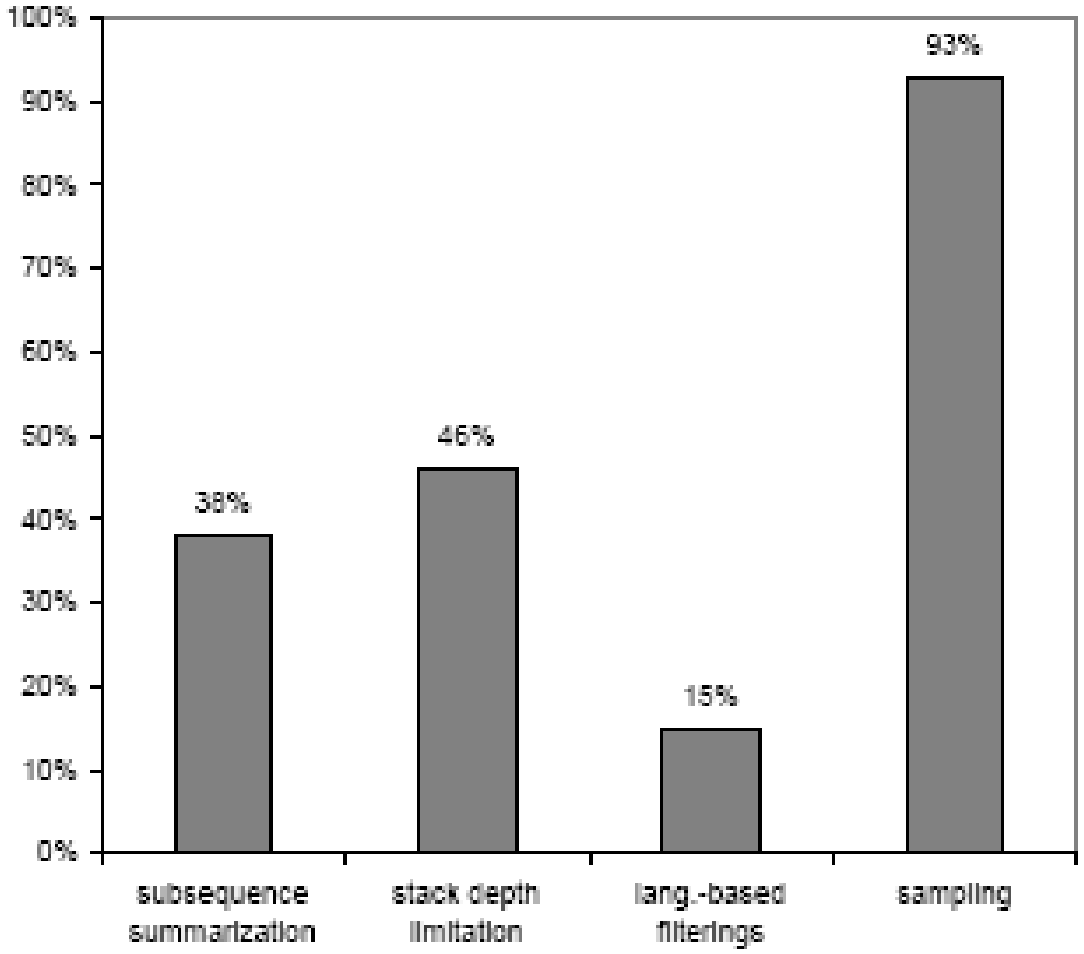
Evaluation Metrics

- Actual output size: The actual size of the output dataset after reduction
- Computation time: The amount of time spent on the reduction, in seconds
- Information Preservation: Preservation of events per type. Three types are defined:
  - High-level events (no fan-in)
  - Low-level events (no fan-out), and
  - Medium-level events (remaining events)

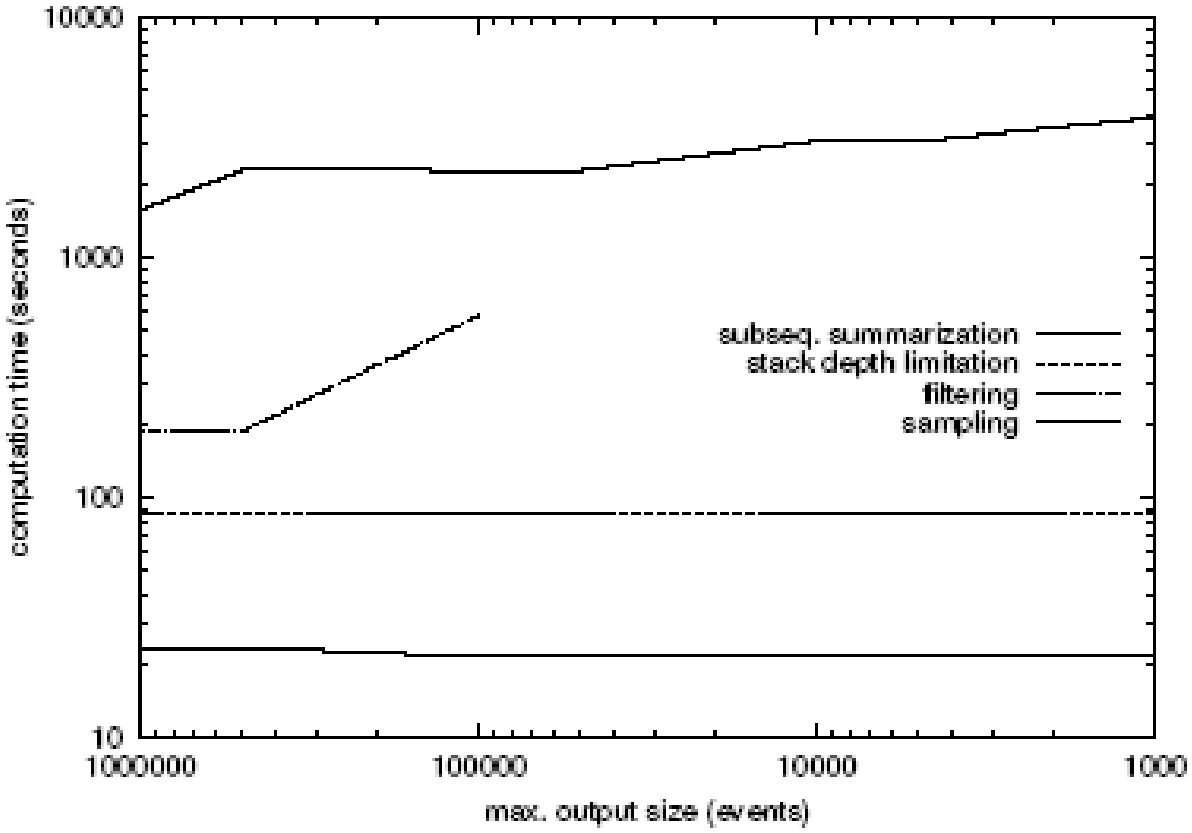
Application

- Each of the four techniques is applied on several traces
- The reductions conform to the maximum output size of the trace
- Seven thresholds with values between 1000 and 1000000 are applied
- 196 runs are performed

Average Reduction Success Rates



Performance



Performance for the ant-selfbuild trace

Information Preservation

- Subsequence summarization attains the best results
- Sampling technique is the least useful technique in preserving high-level and medium-level events in our context
- Stack depth limitation and language-based filtering techniques preserve low-level events at the cost of medium-level events.

Summary of the Results

	Subsequence summarization	Stack depth limitation	Language-based filterings	Sampling
reduction success rate	o	o	–	+
performance	–	o	o	+
information preservation	+	o	o	–

Where:

+ Good Results

o Acceptable Results

– Bad Results

Trace Abstraction Techniques

- Pattern Detection
- Utility Removal
- Data Collection
- **Visualization Techniques**

Trace Abstraction Techniques Based on Visualization

- Many tools have been developed to help analysts in the process of studying execution traces
- These tools provide a collection of trace abstraction techniques supported by visualization techniques in a graphical user interface mode
- They also provide a set of features that visualize the traces and enable user interactions

Trace Analysis Tools that Support Trace Abstraction Techniques

Shimba	Systä et al.
ISVis	Jerding et al.
Ovation	De Pauw et al.
Jinsight	De Pauw et al.
Scene	Koskimies et al.
Program Explorer	Lange et al.
Collaboration Browser	Richner et al.
AVID	Walker et al.
SEAT	Hamou-Lhadj et al.
OSE	Bennett et al
TPTP	Eclipse Plugin
LTTNg and LTTV	Desnoyers et al.
VET	McGavin et al

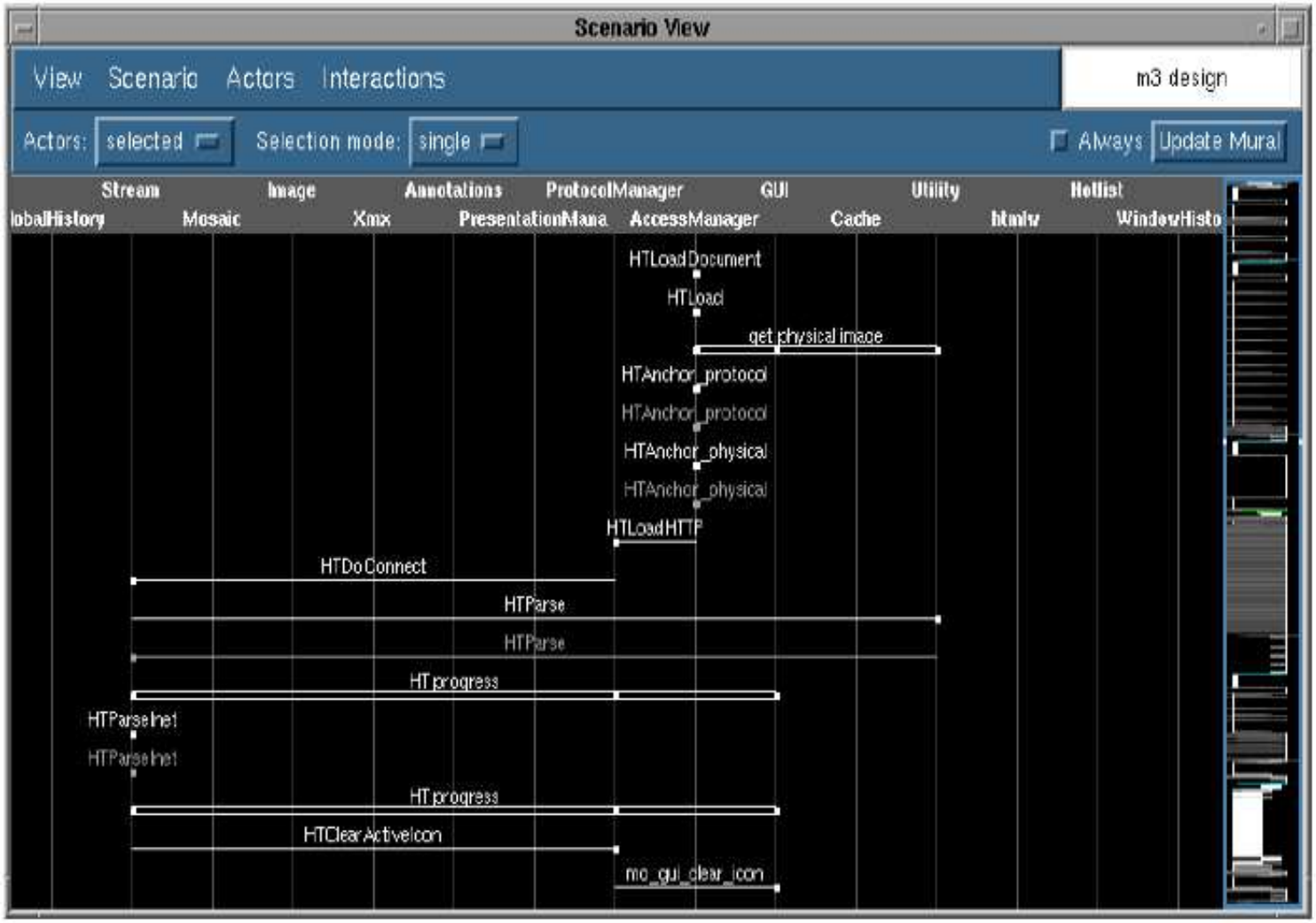
Presentation Features

- **Layout:** Defining the standard through which a sequence diagram is laid out.
- **Multiple Linked Views:** Providing a number of views that are linked together.
- **Highlighting:** Highlighting a part of the sequence corresponding to user selection.
- **Hiding:** Providing the ability to hide some information.
- **Visual Attributes:** Using colors and shapes.
- **Labels:** Labeling classifier, messages, and return values.
- **Animation:** Supporting animation.

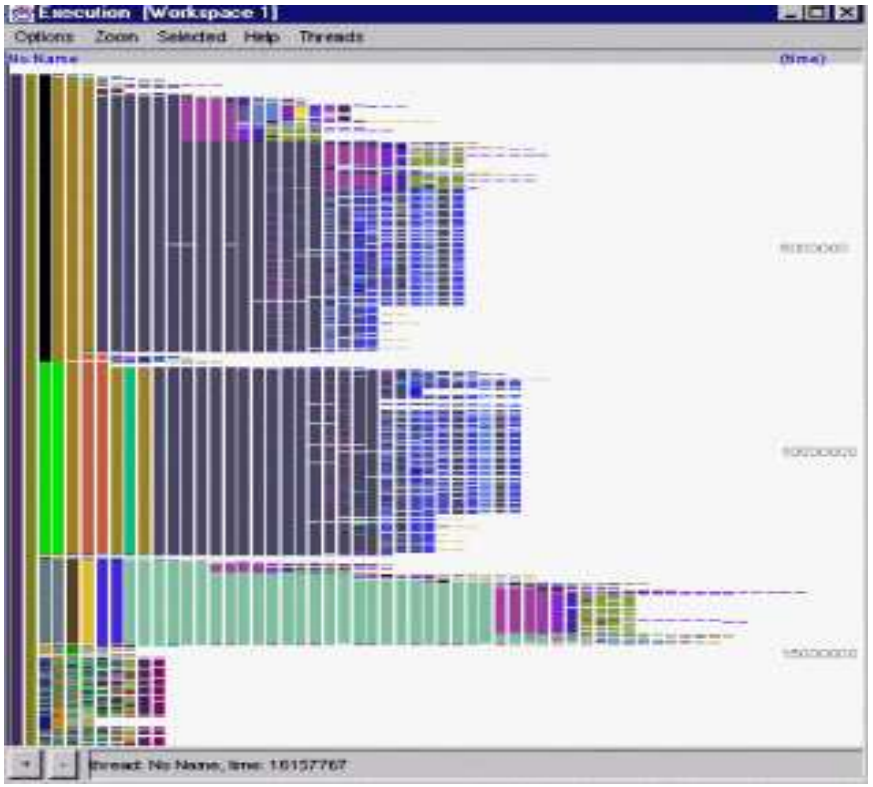
Interaction Features

- **Selection:** Selecting elements to manipulate, filter, or slice
- **Component Navigation:** Navigating between components and instances
- **Focusing:** Providing techniques such as: collapsing, partitioning, an showing related messages to selected objects only
- **Zooming and Scrolling:** Enlarging or reducing the size of the components, and 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

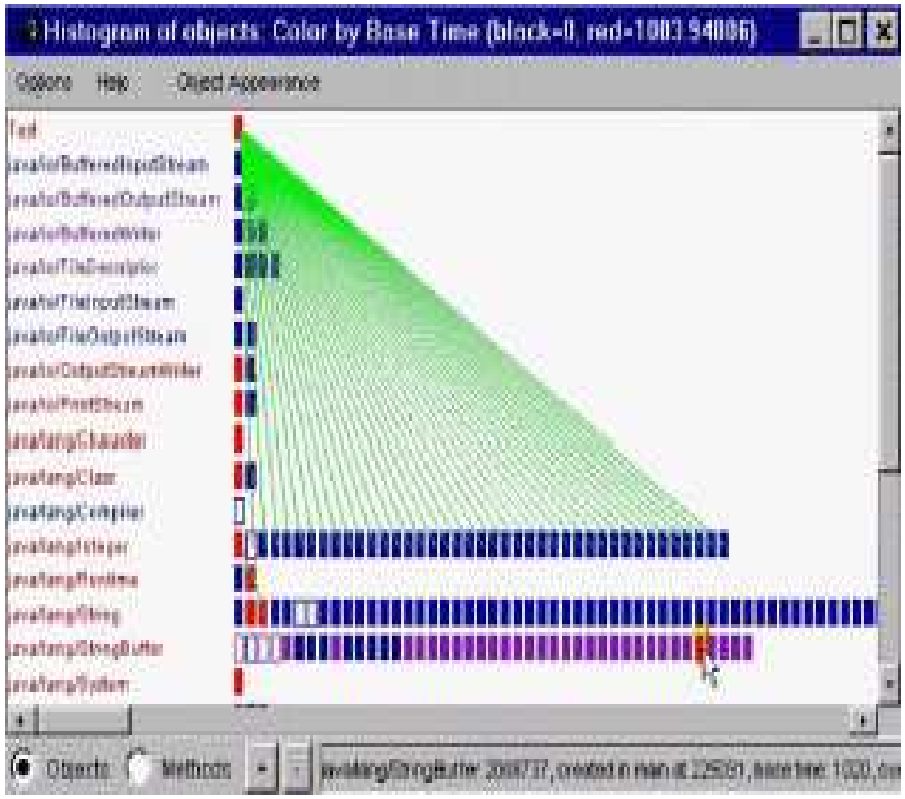
Example: ISVis
Jerding and Rugaber



Jinsight (DePauw et al.)



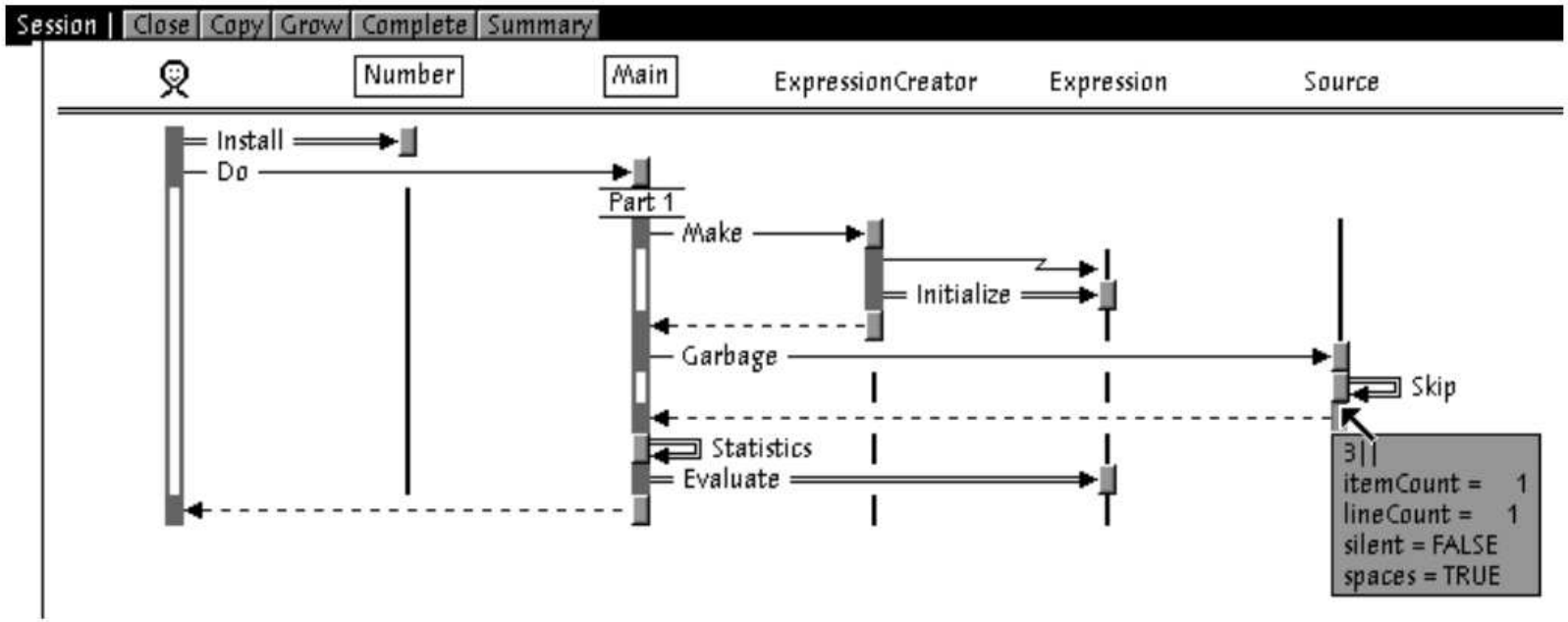
Execution View
thread interactions, detecting deadlocks, etc.



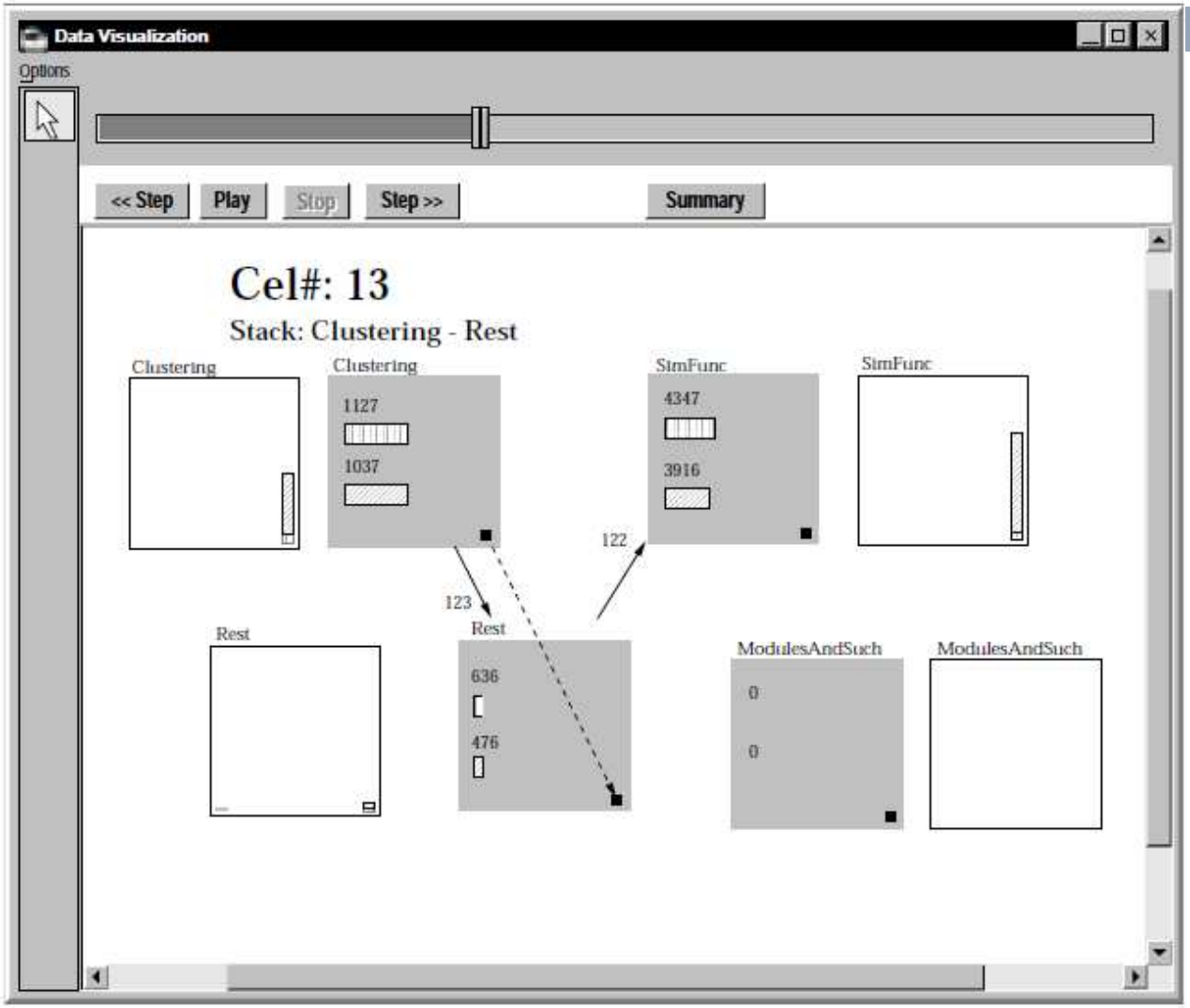
Histogram View
Object creation and deletion

Scene
Koskimies et al.

- Scen (SCENario Enviroment) is a visualization tool that helps analyzing object oriented systems.
- It generates scenario diagrams from execution traces



AVID (Walker et al.)



Conclusion

- We presented and discussed several trace abstraction techniques and tools that can be used to understand large traces
- Four category of techniques:
  - Pattern detection
  - Utility removal
  - Data collection
  - Visualization features
- These techniques and tools vary in their design, their usage, and their effectiveness

Future Work

- We will continue comparing and studying trace abstraction techniques
  - We will present a complete study on the next scheduled meeting
- We will start collecting traces generated from multi-core systems to build a trace bank on which we can test our techniques
- We will start developing a new abstraction algorithm that combines key features from various categories
  - The objective is to be able to compare and analyze traces generated from various runs of a system

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