Using Theia to take trace analysis and visualization to the next level

Bernd Hufmann, Ericsson AB
Agenda

- Background and Trace Compass introduction
- Trace Compass re-architecture
- Prototypes with Theia
- Demo
- Opportunities
- Q&A
What is tracing?

- Trace
  - Series of events over time
  - Event collected at tracepoints during program execution
  - Each event has a type and payload

- Use the events as input for analysis
- Create visualization graphs with these analysis

- Tracing use cases
  - Profile application
  - Find long executions
  - Investigate real-time deadlines
  - Find memory or load issues
  - Investigate concurrency problems
Eclipse Trace Compass is an open source application to solve performance and reliability issues by reading and analyzing traces and logs of a system.

Its goal is to provide views, graphs, metrics, and more to help extract useful information from traces, in a way that is more user-friendly and informative than huge text dumps.

Key characteristics
— Handles trace larger than available memory
— Correlate traces from heterogenous system
Trace analysis and visualization flow

- **Trace(s)**
- **Stream of events**
- **Persistence**
- **Views**

- **parse**
- **analyze**
- **query**

- Events Table
- Intermediate Results
Many trace types supported
Built-in or installable

- LTTng-UST
- LTTng-Kernel
- OpenTracing
- OpenSSH-logs
- HAProxy-logs
- FTrace
- GDB
- Atrace
- Trace Compass
- BTF
- Custom-Text
- Custom-XML
- Catalina-Logs
- UFTrace
- Hadoop-Logs
- PCAP
- Perf
- Bare-Metal
Build-in analysis

Resources

Critical Path

Control Flow

Kernel Memory Allocation

Futex

System calls

Events Density

Multi-node synchronization

Disk I/O

Malloc Tracking

IRQ Handlers

CPU Usage

Performance Counter

Calling Context Tree

Descriptive statistics

Density Distribution

Flame Graph

Scatter Plot
Monolithic application

- Standalone RCP
- Plug-ins for Eclipse IDEs

- Challenges
  - Limited to Eclipse
  - Scalability
  - Higher coupling
  - Re-use
  - Maintenance
Towards client-server architecture

- One trace ≠ one client ≠ one view
  - Different types of clients
  - Different visualizations for same trace
  - Prevent loading entire trace into client

- Domain specific logic is already implemented in Trace Compass
  - Service based solution
  - Avoid re-implementing it in the client
  - Preserve investment made
Trace Compass re-architecture

Trace Compass UI

Trace Compass Core

Data Providers

Trace Compass Core

Trace
State System
Segment store
Trace Compass re-architecture
Client-server architecture

Visualization UI

Data Serializer

Data Providers

Trace Compass Core

Trace
State System
Segment store
Client-server architecture

- **Presentation Layer**
- **Business Layer**
- **Data Layer**

**Trace Server**

- **Trace Server Protocol (TSP)**
- **Visualization UI**
- **Data Serializer**
- **Data Providers**

**Trace Compass Core**

- **Trace**
- **State System**
- **Segment store**
Client-server architecture

- Presentation Layer
- Business Layer
- Data Layer

Visualization UI

Data Serializer

Data Providers

Trace Server

Maximal re-use of core components

Trace Server Protocol (TSP)
Trace Server Protocol (TSP)
Trace Server Protocol (TSP)

- Protocol to handle communication between backend and frontend
- Allowing traces to reside and be analysed on the backend.
- Exchange visualization data between a client and a server
- Trace management, available data providers, server-side filtering and searching
- [https://github.com/theia-ide/trace-server-protocol](https://github.com/theia-ide/trace-server-protocol)
- Integration with Theia using tsp-typescript-client
  - TSP ready client to perform your requests
  - Abstract the technology used (REST, HTTP)
  - NPM package available
    - [https://github.com/theia-ide/tsp-typescript-client](https://github.com/theia-ide/tsp-typescript-client)
- Contribution and feedbacks are welcome
Trace Server Protocol (TSP)

Disclaimer

- Current implementation ≠ protocol
- It’s a REST API
- HTTP protocol
Eclipse Theia

- Extensible platform
- Uses modern web-technologies
- Allows integration with IDE workflow
  - LSP – for language support
  - DAP – for debug support
- Deployment
  - Browser or desktop
  - Workspace managers like Eclipse Che
- Open Source and community
- Vendor neutral
Theia front-end

- Theia based prototype using the TSP
- Prototype available on GitHub (https://github.com/theia-ide/theia-trace-extension)
- Opportunity for a new UI/UX
- React
- Chart.js
- agGrid
- New time graph library
Demo
New opportunities

— Integration with other web-based solutions
  — Cloud IDE
  — Dashboards integration (CI, bug reports tools)

— Deployment
  — Single click instead of desktop installation
  — Accessible from various devices

— Leverage modern, state-of-the-art UI technologies

— Scalability
  — Handle traces larger than local disk

— Increased security
Integration with workspace management applications

**Eclipse Che / GitPod**

- Prepare workspaces for trouble-shooting sessions
  - Setup cloud IDE
  - Get source code
    - LSP
  - Setup debuggers
    - DAP
  - Setup trace viewer
    - TSP
- Share trouble-shooting sessions (workspaces)
Dashboard integrations
For Example CI or TR tools

Submit code → Source repository

Trigger → Run build

Run unit test with tracing

Show result on dashboard

Analyze traces on trace server

Post traces to trace server

— One-click check-out traces in IDE
— Jump directly to interesting areas in traces
— Fetch source code if needed
— Setup debugger
Higher scalability

- Enables micro-services
- Distributed architecture
- Parallel, distributed analysis
  - Different traces
  - Same traces, different analysis
- Analyze traces that exceed local disk space
Trace & Debug

- Crash dump analysis
  - Show traces before crash
  - Open core file with debugger

- Navigate source code using traces
  - Using source locations

- Collect traces when hitting breakpoint
Other opportunities

- Leveraging modern UI technologies
- Thin UI clients
- Light weight / single function trace servers
- Server or client in different programming languages (e.g. Python, Go, R)
- Improved security
  - Traces in the cloud
- Trace and analyze Theia with Theia
Take-aways

— Tracing is a proven and efficient trouble-shooting technique

— Trace Compass is a tool to analyze all kinds of traces

— Shift to client-server architecture for Trace Compass

— Maximal re-use of domain specific logic of Trace Compass in server

— Trace Server Protocol allows for new front-end using Theia

— New opportunities open up with new architecture
References

— Trace Compass
  — http://tracecompass.org
  — https://projects.eclipse.org/projects/tools.tracecompass
  — https://projects.eclipse.org/projects/tools.tracecompass.incubator

— Trace Server Protocol
  — https://github.com/theia-ideTRACE-SERVER-PATTERN
  — https://github.com/theia-ide/tsp-typescript-client

— Theia frontend prototype
  — https://github.com/theia-ide/theia-trace-extension
Contacts

— Presenter
  — Bernd Hufmann: bernd.Hufmann@ericsson.com

— Mailing list
  — tracecompass-dev@eclipse.org

— IRC
  — oftc.net #tracecompass