Introducing:
Trace Server Protocol

Bernd Hufmann, Ericsson AB
Agenda

— Motivation and goal
— Background
— Trace Server Protocol (TSP)
— Opportunities
— Ongoing work / Demo
— Open discussions
Motivation

- Increasing popularity of web-based technologies
- Integration with next generation IDEs (e.g. Theia)
  - Success of Language Server Protocol
- Automated trace analysis
  - CI environment
  - Trouble reports
- Scale trace analysis for traces larger than local disk space
Goals of presentation

— Present idea of Trace Server Protocol
— Create awareness and interest in the community
— Collect early feedback
— Collaboration
Next generation IDE

Language Server Protocol (LSP)  Debug Adapter Protocol (DAP)

Language Server  Debug Server
Next generation IDE

- Language Server Protocol (LSP)
- Debug Adapter Protocol (DAP)
- Trace Server Protocol (TSP)
What is a trace?

— A series of events over time
— Events collected at tracepoints during program execution
— Each event has a type and content fields (payload)
What can we do with events?

Show the raw events in an events table
What can we do with events?

— Use as input for further analysis, e.g.
  — Write state machines
  — Create patterns
  — Analyze timing (measuring time between events)
  — Create execution graphs like critical path
  — Follow resource usage
  — ....
What can we do with the analysis results?

— Create all sorts of visualization graphs
What can we do with the analysis results?
Tracing use cases

— Help developers, trouble-shooters, system admins etc. to solve complex problem
  — Profile applications using trace data
  — Find long executions
  — Get critical path executions
  — Correlate traces from different nodes
  — Investigate real-time deadlines
  — Find memory usage issues
  — Find high processor load
  — Investigate concurrency problems
  — ...

Next generation IDE

- Language Server Protocol (LSP)
- Debug Adapter Protocol (DAP)
- Trace Server Protocol (TSP)
Proposed architecture

- Layered architecture
- Separation of UI and core
- Trace Server Protocol
  - REST API or RPC
Scope of Trace Server Protocol (TSP)

- Handle the communication between core and UI of trace viewer
- Exchange visualization data between a client and a server
- Trace file and trace set management
- Manage available visualizations
- Provide data for various visualization types
- Define filter and queries
- Manage trace annotations (e.g. bookmarks)
Not in TSP scope

— Interface to the compiler
— Interface to the debugger
— Interface to the tracer
Opportunities
Integrated feedback in design workflow

- Automatically detect traces
- Show trace files to user
Leveraging LSP and DAP

- Use DAP to get file and line number
- Then use LSP to lookup source code

- Use LSP to lookup source code

- Source code lookup from different graphs
- Automatic source code lookup when stepping
Continuous integration

Submit code

Trigger

Source repository

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
Integration with TR tools

- One-click check-out traces in IDE
- Jump directly to interesting areas in traces
- Fetch source code if needed
Integration with workspace management applications

Eclipse Che

- Easily **prepare** workspaces for trouble-shooting sessions
  - Use cloud IDE
  - Get source code
    - Languages servers
  - Setup debuggers
    - Debug server
  - **Share** trouble-shooting sessions (workspaces)
Enables micro-services architecture

Language Server Protocol (LSP)

Debug Adapter Protocol (DAP)

Trace Server Protocol (TSP)

Microservice

Trace Server
Higher scalability

- Distributed architecture
- Parallel analysis
- Distributed analysis
- Multiple trace servers analyzing
  - different traces
  - same traces but different analysis
- Analyze traces that exceed local disk space
Other opportunities

— Server or client in other programming languages (e.g. Python, Go, R)
— Light-weight / single function trace servers
— Leveraging modern UI technologies
— Thin UI clients
Proof-of-concept
Trace server protocol

- Specification (preliminary)
  - https://github.com/theia-ide/trace-server-protocol
- TSP TypeScript client library
  - https://github.com/theia-ide/tsp-typescript-client
Trace server PoC

- Trace server based on Trace Compass
  - Available in Trace Compass incubator project
  - Eclipse RCP using Trace Compass core components
  - REST over HTTP (JSON)
  - Jersey Rest library
— Framework for IDE-like applications
— Runs in the cloud or on the desktop
— Uses modern web technologies
— Supports multiple languages via LSP
— Supports debugging via DAP
— Extensible (via extension or plug-ins)
— https://www.theia-ide.org/
Trace visualization front-end PoC

- Implemented as extension to Theia
- **TSP TypeScript client library**
- **Timeline-chart** library for Gantt-chart type views
  - newly created
- **Chart.js** for XY-charts
- **agGrid** for Events table
Research Results
Data Exchange Protocols

- Results for a resolution of 1000 data points
- Time overhead is of the serialization/deserialization/network (localhost) vs the time of just getting the data

<table>
<thead>
<tr>
<th></th>
<th>JSON</th>
<th>JSON (Gzip)</th>
<th>Protobuf</th>
</tr>
</thead>
<tbody>
<tr>
<td>XY Charts (1 series)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response Size (KB)</td>
<td>80</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Time Overhead (%)</td>
<td>5.2</td>
<td>7.8</td>
<td>4.8</td>
</tr>
<tr>
<td>XY Charts (25 series)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response Size (KB)</td>
<td>800</td>
<td>50</td>
<td>300</td>
</tr>
<tr>
<td>Time Overhead (%)</td>
<td>25.3</td>
<td>34.3</td>
<td>9.2</td>
</tr>
<tr>
<td>Time graph row data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(25 rows)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response Size (KB)</td>
<td>1150</td>
<td>100</td>
<td>120</td>
</tr>
<tr>
<td>Time Overhead (%)</td>
<td>9.6</td>
<td>13.3</td>
<td>3.5</td>
</tr>
</tbody>
</table>
Demo
Potential Collaborations

- Trace Server Protocol
- Front-end libraries
  - E.g. Time graph (has started)
- Front-end (e.g. Theia)
- Trace server backend
Challenges

- Define a generic TSP and thin front-end
- Efficient data exchange (large amount of data, frequency of updates)
- How to customize for special / domain specific use cases, e.g.
  - Display of custom values
  - Special sorting
  - Server-side filtering
  - Command executions (e.g. follow thread)
- Security aspects
References (prototype repositories)

- Trace server protocol
  - https://github.com/theia-ide/trace-server-protocol
- TypeScript library implementing Protocol
  - https://github.com/theia-ide/tsp-typescript-client
- Timeline-chart library
  - https://github.com/theia-ide/timeline-chart
- Theia integration prototype
  - https://github.com/delislesim/theia-trace-extension
Other References

— Project pages
  — [http://tracecompass.org](http://tracecompass.org)

— Documentation
  — [Trace Compass User Guide](http://tracecompass.org)
  — [Trace Compass Developer Guide](http://projects.eclipse.org/projects/tools.tracecompass.incubator)

— Eclipse Theia
  — [https://www.theia-ide.org/](https://www.theia-ide.org/)
Contacts

- Presenter
  - Bernd Hufmann: Bernd.Hufmann@ericsson.com

- Mailing list
  - tracecompass-dev@eclipse.org

- IRC
  - oftc.net #tracecompass

- Mattermost
  - https://mattermost-test.eclipse.org/eclipse/channels/trace-compass