Monitoring and Analytics for the whole Lifecycle, on Models, Hardware, and Software

D6.1 Visual Analytics Requirements

Deliverable Due Date	2026-01-16
Deliverable Type	
Deliverable Number	D6.1
Authors	Artho Cyrille (KTH)
Dissemination Level	Public

Contributors

Name	Short Affiliation
Cyrille Artho	KTH

Reviewers

Name	Short Affiliation	Date
Batur Alp Akdoğan	LTG	2025-12-19

Document Revision Log

Version	Revision	Date	Description	Author
01	01	2026-12-15	First Draft	Artho Cyrille (KTH)
01	02	2026-01-16	Submitted	Artho Cyrille (KTH)
			Updated	

Table of Contents

Definitions and Acronyms

AI - Artificial Intelligence
API - Application Programming Interface
CI/CD - Continuous Integration and Continuous Delivery/Deployment
CPS - Cyber-Physical Systems

1. Executive Summary of Deliverable

This delivery provides an analysis of the visual analytics work packages. It also defines the architecture for the implementation. The objective of this executive section is to inform of the project's results, highlight significant findings, and recommend actionable steps moving forward. Overall, the analysis indicates that modern tools are designed to be accessible from a web browser, which suggests the use of a modular tool that permits this. Further, the summary defines the context in which WP6 interacts with its inputs from WP3–5.

The Visual Analytics work package (WP6) provides the user-facing front-end to present the results of the two analysis-related work packages: WP4 (model reengineering) and WP5 (semantic analysis).

The visual analytics tool therefore has to load the trace data provided by WP3 (infrastructure) and present the results of WP4 and WP5 together with that trace data.

The envisioned architecture of WP6 has two main components: a back-end (**trace server**) that loads the trace data and analytics results, and a front-end (**trace viewer**) that presents the results to the user (see Fig. 1).

The trace viewer provides a web-based interface to the user and therefore in turn requires a server component (to provide the static content of the web page) that acts as a client to the trace server. The user accesses the application through a web browser.

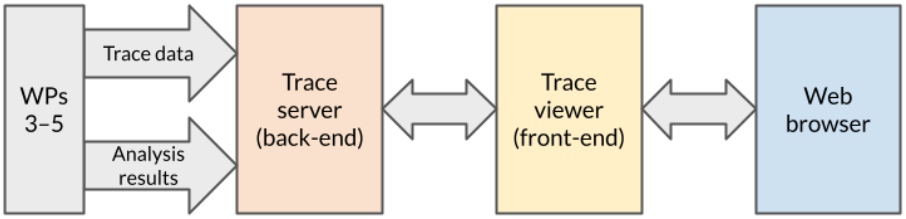

Fig. 1: WP6 architecture

The primary data format that we support is the JSON-based Google Trace Event format. Other formats like the Common Trace Format can be supported if the need arises. For each supported format, the incoming data shall be converted so the back-end receives all the required information (such as time stamps and events marking the beginning and end of an action).

It is desirable that we can include other components in the trace server and trace viewer; this can be arranged by supporting multiple data back-ends. A likely candidate for the technology of the front end is the Theia-based "Theia Trace Viewer", which operates on top of Eclipse Trace Compass. In that case, care needs to be taken so that the functionality can also be embedded in VS Code (which contains closed-source components and with which Theia shares much of the open-source part of the code base). This requires care to avoid Theia-specific API functions as to remain compatible with both platforms (VS Code and Theia).

In Fig. 1, the information flow to WP6 is in only one direction. Later in the project, we would like the user to also be able to manipulate analysis data. This will require an interface back to WP4 and WP5 to provide an interaction with analysis tools as a human refines their view while studying the trace.

2. Background

2.1 Trace data formats

In this text, we use the term "trace data" to represent both textual data from logs and data that represents recordings of signals, such as sensor values. In both cases, we consider the trace to be a sequence of data items. Each data item has a time stamp and a value. In case data does not contain time stamps, we will generate synthetic time stamps in ascending order.
Data items are stored by upstream work packages (WP3, WP4, WP5) in several ways:

1. Data can be stored in a binary format, which allows certain data items to be stored directly as it is generated.
2. Data can be stored in a textual format, which expresses, e.g., numbers, in a way that is typically human-readable but needs to be interpreted first before it can be used.

Furthermore, the exact nature of the data can be assumed to be fixed (and encoded in the software that reads the data) or be relatively generic and limited to common data such as numbers and text. The alternative is to have more flexible ways of defining data by also having a record of meta-data information. This meta-data provides a machine-readable description of what the exact data types are. Meta-data can cover cases where the data types are not obvious and go beyond numbers and strings.
The different formats have their pros and cons:

- Binary data is precise but requires additional tools to store, load, and manipulate.
- Textual data is human-readable, but numbers are subject to loss of precision due to conversion from binary to decimal representation and rounding (when storing information), and conversion back to binary (when loading data). A major benefit is that in principle, any text editor can display and modify data.
  JSON-based data in particular is both textual and well-supported by editors and software libraries.
- If meta-data is used, one piece of software can support many data formats, and adaptation to new data types is easier. Setting the correct meta-data and handling it correctly in all cases may be more challenging than handling a few fixed types of data, though.

In our initial work, we will use the Google Trace Event format. It is a JSON-based representation of data with a fixed number of attributes. From what we expect, it supports most of our use cases while already having good support in Trace Compass and similar tools.

2.2 Trace analysis tools

General attributes

To perform the visual analysis of trace data (and fulfill the main objective of this work package), we want to have a tool that combines the following attributes:

- **Openness.** At least the base version of the tool should be available under a permissible open-source license, so that we can modify it and share it freely without having to enter a

licensing agreement with the tool provider. This ensures easy dissemination and provides longevity beyond the three years of the funded project duration.

- **Extensibility.** We will have to modify existing functionality and provide new functionality, so the tool should be designed with that in mind.
- **Web-based front-end.** Adoption of a new tool is difficult, especially if new versions have to be installed locally on each end user's computer. We therefore prefer tools that can be accessed through a web browser, so only a single installation per organization that uses a tool is needed. This greatly simplifies software updates and allows anyone to try out a demonstration version.

Selected tools

There are several tools that fulfill these criteria:

1. **Eclipse Trace Compass** (with the Theia Trace Viewer) [Ecl2025a,Ecl2025b]. This tool is very mature and has been designed to analyze traces from the Linux kernel and embedded systems. It is available as a monolithic application running on a desktop or as a client/server solution that has the Theia Trace Viewer front-end connect to the Eclipse Trace Compass back-end, providing a web-based interface.
2. **Perfetto** [Goo2025]. This tool by Google is designed with a web front-end and provides trace analytics for software from Google's software stack, such as Android applications or the Chrome browser.

The two tools in the list above are relatively general and thus suitable for MONA LISA. Eclipse Trace Compass has the advantage of supporting both the Google Trace Event format and the Common Trace Format and of being integrated with Theia. This is useful for an integration of our work in a general development environment (like VS Code). It therefore was, at the time of writing the project application, and still is, the best fit with our objectives.

Diagram capabilities

Based on prior experience with trace analytics, we initially want to support a set of general-purpose views. Specific types of views that are adapted to particular use cases will follow at a second stage. General visualizations include the following types of views:

Log view: A general tabular view can visualize the raw log data while at the same time providing the option to highlight certain parts of a log.

Waveform view of numerical data: Data where each event is associated with a numerical measurement is shown as a waveform-like view.

Summary statistics: Summary-level information such as the number of events of each type can be shown in tables and diagrams such as pie charts.

Flame graph: A flame graph is uniquely suited to show how different functions in software interact with each other. It shows each nested call in a new row below the current caller and uses the horizontal axis to visualize the time needed to complete a call.

3. Envisioned architecture of WP6

We again refer to Fig. 1 for an architecture overview. WP6 will ingest trace data (provided by WP3) and analysis results, which can be models (WP4) or less structured verdicts such as a textual description (WP5).
Note that in the figure, the data flow in the analysis is from upstream work packages to WP6, but not yet back. This ensures a clear way of integrating that data without requiring a bidirectional workflow yet. Once we have successfully integrated data from upstream work packages, we will experiment with model or data updates that are driven by visual analytics and may in turn cause a refinement of the data analysis.
As mentioned above, we will focus on Eclipse Trace Compass and Theia Trace Viewer as the software platform and the JSON-based Google Trace Event format for data exchange between work packages. It also supports all the visualization capabilities listed in the previous section.

T6.1: Use case support

The work in T6.1 has the following goals.

1. Support of general-purpose views for the use case data. This requires correctly importing and converting said data.
2. Support for domain-specific views. This can be achieved in two ways:
    a. If a project partner already has software that produces the desired view, we can embed that view in our front-end. The challenge here is that the external view is produced by an entirely different software stack. Unless that software is written in (or can be transpiled into) JavaScript, it has to be run in an external process, and we have to set up a bidirectional communication channel between a user interface element in the front-end that displays the view and receives user input and the external process.
    b. If no software to support a specific view exists yet, we will either adapt an existing view or create a new view by extending the back-end and front-end.

T6.2: Planned enhancements

Beyond these given views, we plan the following cross-cutting enhancements in WP6:

1. Scale-aware Zoom capabilities.
   As of now, the trace viewer simply scales the currently selected data to the available size of the view. The only adaptation to having more data than what can be reasonably shown is that text that cannot fit is omitted.
   In MONA LISA, we want to go beyond that and implement ways to summarize parts of a view or diagram. This corresponds to how cities are shown in maps, from showing individual buildings at a small scale to a simple circle at a larger scale.
2. Overlays.
   Currently, different views are synchronized in that they show the same part of the trace, but they are otherwise independent. We want to extend this to allow data from one view to be overlaid (superimposed) on another one. This corresponds to how, e.g., traffic data can be overlaid on a city map.

References

[Ecl2025a] Eclipse Foundation. Eclipse Trace Compass. https://eclipse.dev/tracecompass/ . Last accessed: 2025-12-01.

[Ecl2025b] Eclipse Foundation. Theia Trace Viewer extension. https://github.com/eclipse-cdt-cloud/theia-trace-extension . Last accessed: 2025-12-01.

[Goo2025] Google. Perfetto: System profiling, app tracing and trace analysis. https://perfetto.dev/ . Last accessed: 2025-12-01.