Softwaretechnik-Projekt
im Rahmen der **MATSE-Ausbildung** mit parallelem BA-Studium **Angewandte Mathematik und Informatik**

Performance-Monitoring-Tool

S. Burk, P. Seyler, J. Böcker, J. Gummersbach, J. Kasemann, T. Ferdinandt

IT Center RWTH AACHEN UNIVERSITY

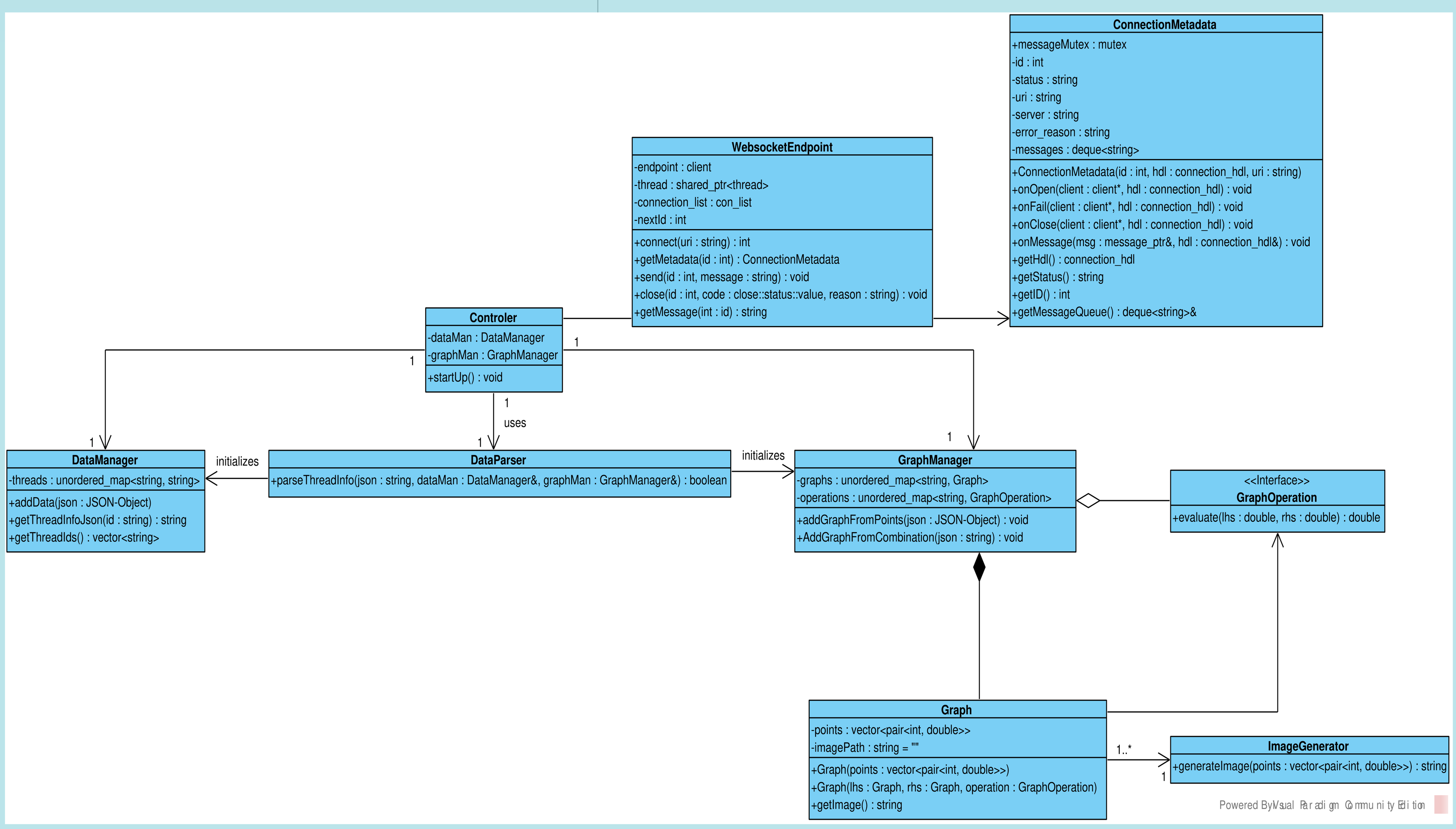
CAE

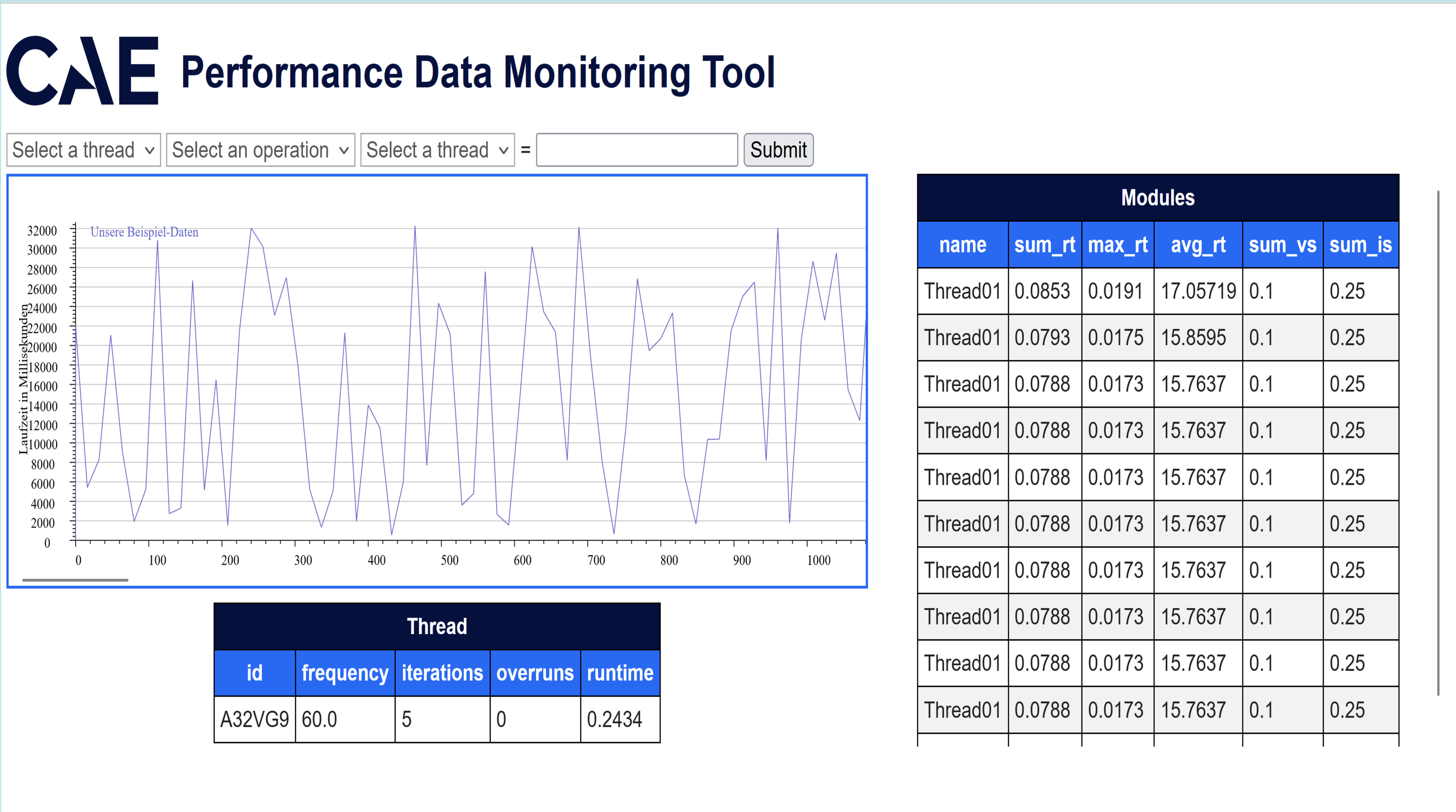
FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Motivation:

Bei CAE werden Simulationsanwendungen entwickelt, die in "Echtzeit" (60 Hz) aktualisierte Daten liefern müssen.
Um dies zu gewährleisten, werden Performance Parameter gemessen, die unter anderem die Zeiten für eine Aktualisierung als aktueller Wert,
Minimum, Maximum und Durchschnittswert über die Gesamtmessung angeben. Desweiteren gibt es zusätzliche Informationen, die die Performance beeinflussen können, wie z.B. "Memory Paging" oder "Context Switches".

Aufgabe war es ein Programm zu entwickeln, welches diese Performance Daten, zu verschiedenen Threads, einer Anwendung, aus einer Quelle abfragen, gruppieren und visualisieren kann.





Thread				
id	frequency	iterations	overruns	runtime
A32VG9	60.0	5	0	0.2434

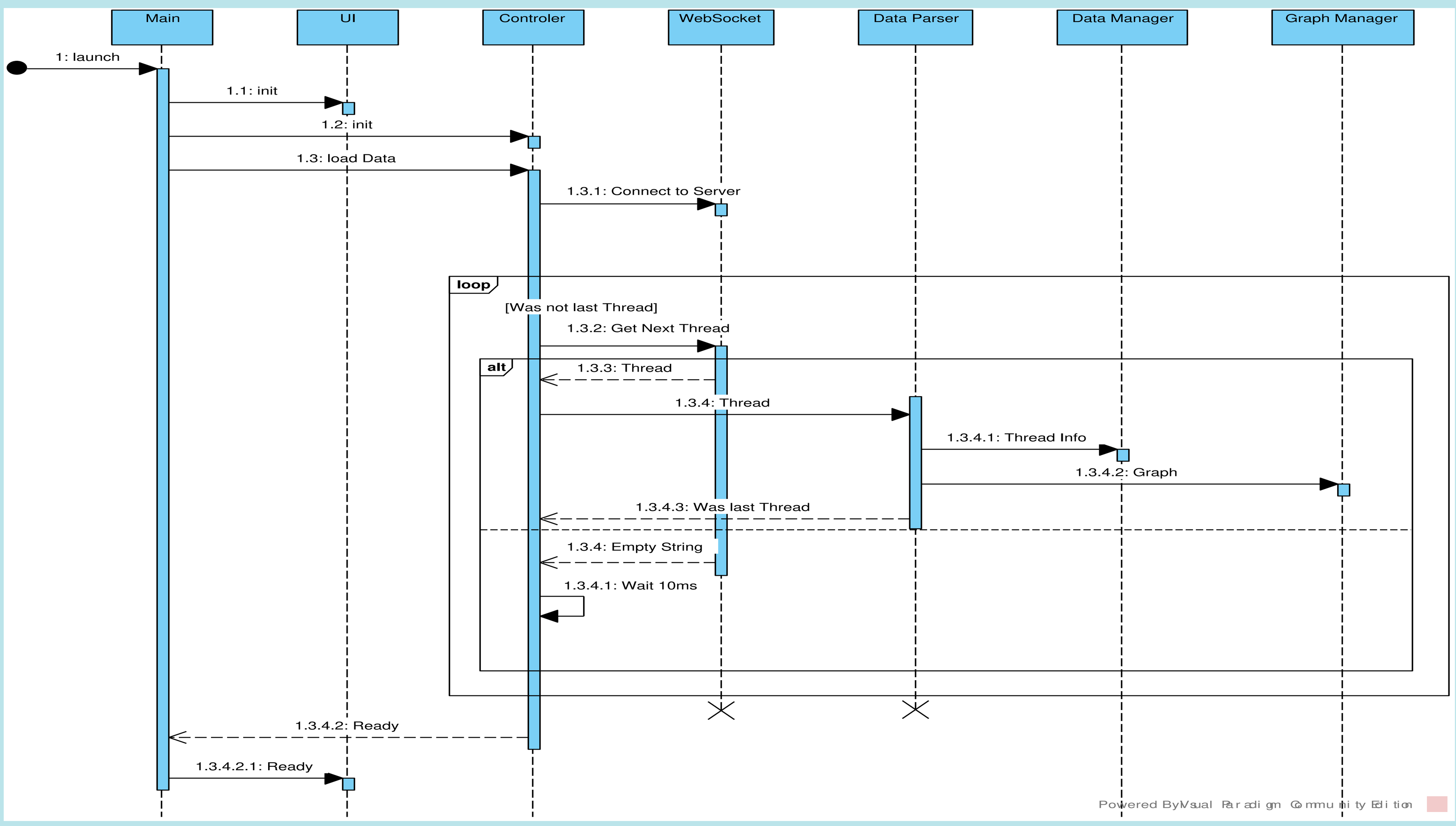
Modules					
name	sum_rt	max_rt	avg_rt	sum_vs	sum_is
Thread01	0.0853	0.0191	17.05719	0.1	0.25
Thread01	0.0793	0.0175	15.8595	0.1	0.25
Thread01	0.0788	0.0173	15.7637	0.1	0.25
Thread01	0.0788	0.0173	15.7637	0.1	0.25
Thread01	0.0788	0.0173	15.7637	0.1	0.25
Thread01	0.0788	0.0173	15.7637	0.1	0.25
Thread01	0.0788	0.0173	15.7637	0.1	0.25
Thread01	0.0788	0.0173	15.7637	0.1	0.25
Thread01	0.0788	0.0173	15.7637	0.1	0.25
Thread01	0.0788	0.0173	15.7637	0.1	0.25

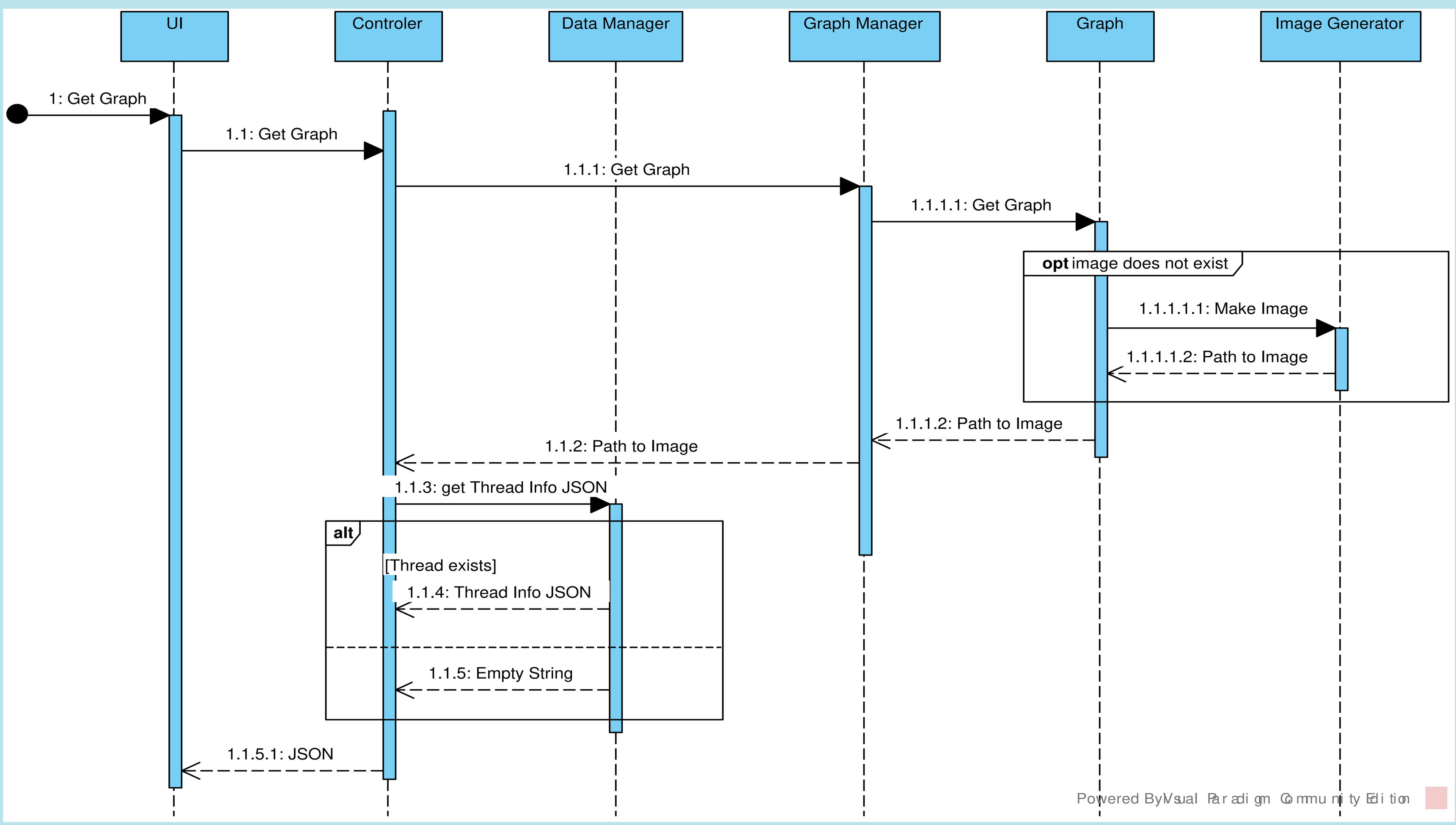
Umsetzung:

Unser Tool verbindet sich per Websocket mit einem Server, der die Daten bereit stellt.
Nach einer Anfrage durch das Tool werden eine Reihe von Datensätzen, im Json Format, gesendet.
Aus diesen werden die benötigten Informationen gewonnen und in einer lokal gehosteten Webanwendung angezeigt.
Graphen werden als Vektorgrafiken aus den Daten generiert. Auf den Graphen sind verschiedene mathematische Operationen anwendbar, beispielsweise die Addition mit einem anderen Graphen.
Die Websocket Verbindung wird nach Erhalt aller benötigten Daten automatisch beendet.

```cpp
/**
 * Defines what to do once a message is received.
 * Pushes the received string into a queue to await usage by the controller.
 * @param hdl a thread-safe pointer to this specific connection
 * @param msg the message-pointer
 */
void ConnectionMetadata::onMessage(const websocketpp::connection_hdl &hdl, const client::message_ptr &msg) {
    // lock the queue to avoid conflicts with the main thread
    std::lock_guard<std::mutex> lk(m_messageMutex);
    // add the message string to the queue
    m_messages.push_back(msg->get_payload());
}
```

```cpp
    // Configures the (client) endpoint and sets up a connection between it and the data Server
    CAEMonitoringTool::Websocket::WebsocketEndpoint endpoint;
    int dataConId = endpoint.connect("ws://localhost:5000");
    // Sends a message to the dataServer to let it know to start sending its data (JSON-Strings)
    endpoint.send(dataConId, "startData");
    bool isDone{false};
    while (!isDone) {
        // Checks the message content. If it is not empty, it parses the string into a JSON object.
        // The return value of parseThreadInfo is a boolean determined by a JSON field of the string.
        // If it equals true, that means that this was the last message to be received and the loop can be stopped.
        string mes = endpoint.getMessage(dataConId);
        if(!mes.empty()){
            isDone = m_dataParser.parseThreadInfo();
        } else {
            // If the message content is empty, that means the next message has not been received yet,
            // and we need to wait for that.
            std::this_thread::sleep_for(std::chrono::milliseconds(500));
        }
    }
```





Um den Graphen an das Frontend zu überreichen, benutzen wir svg (Scalable Vector Graphics)-Dateien.
Diese erlauben es uns unterschiedlich lange Zeitabschnitte anzuzeigen.
Jeder generierter Graph enthält eine svg-Datei, welche auf Abfrage an das Frontend weitergegeben wird.
Des Weiteren werden neue svg-Dateien für die Verknüpfungen angefertigt.
Es wird also nicht bei jedem Aufruf des Graphen eine neue svg-Datei erstellt, sondern nur wenn noch keine existiert.

Programmiersprachen und Technologien:
C++, Python, HTML, CSS, Nix, Websocket++, PPlot, Nlohmann/Json, Boost, JavaScript, NodeJS