

École polytechnique de Louvain

The ΔQ Oscilloscope: Real-Time Observation of Large Erlang Applications using ΔQSD

Author: **Francesco NIERI**

Supervisor: **Peter VAN ROY**

Readers: **Neil DAVIES, Tom BARBETTE, Peer STRITZINGER**

Academic year 2024–2025

Master [120] in Computer Science

Abstract

It is difficult to study the detailed behaviour of large distributed systems while they are running. What happens when there is an overload? How can we feel something is wrong with the system early enough?

This thesis aims to provide further proof about how the Δ QSD paradigm can be used to study the behaviour of running systems and to explore tradeoffs in system design, thanks to the implementation of the **Δ Q oscilloscope**, a real time graphical dashboard that gives insights into a running Erlang system. Furthermore, the development of an Erlang application (Δ Q adapter), named `dqsd_otel`, allows the running system to communicate with the oscilloscope to receive real time insights about the execution of the former.

The oscilloscope performs statistical computations on the time series data it receives and displays the results in real time, thanks to the Δ QSD paradigm. We provide a set of triggers which are set to capture rare events, like an oscilloscope would, and give a snapshot of the system under observation as if it was frozen in time. An implementation of a textual syntax allows the creation of outcome diagrams which give an "observational view" of the system. Furthermore, the implementation of efficient algorithms allows for the computations to be done rapidly on precise representations of components.

We first provide an extensive summary of Δ QSD concepts, which have been extended to allow the instrumentation of Erlang systems. Subsequently, we explain the user level concepts which are essential to understand how the oscilloscope works and understand what is displayed on the screen, delving later on into the mathematical foundations of the concepts. Lastly, we provide synthetic applications which prove the soundness of Δ QSD and show how the oscilloscope is able to detect problems in a running system, diagnose it and explore design tradeoffs.

Acknowledgments

This thesis is the culmination of my studies, I would like to thank the people who made this possible, those who supported me through the years and those who helped me with the thesis.

My family, especially **my mom, my dad, my brother** and **my sister**, for their help which was a crucial shoulder I could lay on while writing this thesis and most importantly throughout these five years.

My **friends**, to those who have come from Italy and have taken time out of their lives to listen the thesis presentation and to those who through the years have been there for me.

A-M., for the moments we shared these four years together in uni.

My dad and **Maurizio**, who nurtured the passion for coding in me.

Peer Stritzinger, Stritzinger GmbH and the **EEF Observability Working Group** (Bryan Naegele and Tristan Slougher) for their help in the EEF Slack, which helped me in understanding OpenTelemetry and gave me the `send_after` intuition.

Neil Davies for taking the time to proofread my thesis.

The **PNSol Ltd.** team for their extensive groundwork on ΔQSD and its dissemination, which made this thesis possible.

Lastly, **Peter Van Roy**, for his year-long relentless interest, support and weekly and constant supervision which made sure the project would come to fruition.

AI disclaimer

AI was employed to help with the graphical dashboard in C++ and the triggers, in positioning the elements, refactoring the code so the widgets would properly interact together, helping understand the FFT algorithm and refactoring the server when communication errors occurred. For the dashboard, 25% of ELOC are **refactored** by AI, they are the constructors of the widgets which nicely place the widgets on screen. The ANTLR CMake was provided by ChatGPT. In total, of around 6000 ELOC, around 10 to 15% has been done or refactored by AI, this is mostly composed of the server and dashboard/trigger code. Comments were generated by ChatGPT and reviewed so they would reflect actual code.

In Erlang, it was used to provide documentation and help with TCP communication exceptions. To give an estimate, around 20% of 350 ELOC are done or refactored by AI and they mostly relate to TCP communication and errors handling. Comments were generated and restructured to present the tools nicely.

The **written master thesis** was written **entirely** without the aid of AI.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 1.1 | Context | 1 |
| 1.2 | Objective | 1 |
| 1.3 | Previous work | 2 |
| 1.4 | Contributions | 2 |
| 1.5 | Roadmap | 3 |
| 2 | Background | 4 |
| 2.1 | An overview of Δ QSD | 4 |
| 2.1.1 | Outcome | 4 |
| 2.1.2 | Quality attenuation (Δ Q) | 5 |
| 2.1.3 | Failure semantics | 6 |
| 2.1.4 | Partial ordering | 6 |
| 2.1.5 | Timeliness | 6 |
| 2.1.6 | QTA, required Δ Q | 6 |
| 2.1.7 | Outcome diagram | 7 |
| 2.1.8 | Outcome diagrams refinement | 9 |
| 2.1.9 | Independence hypothesis | 10 |
| 2.2 | Observability | 11 |
| 2.2.1 | erlang:trace | 12 |
| 2.2.2 | OpenTelemetry | 12 |
| 2.3 | Current observability problems | 15 |
| 2.3.1 | Handling of long spans | 15 |
| 3 | Design | 17 |
| 3.1 | Measurement concepts | 17 |
| 3.1.1 | Probes | 17 |
| 3.1.2 | Extending failure | 18 |
| 3.1.3 | Time series of outcome instances | 18 |
| 3.2 | Application side | 19 |
| 3.2.1 | System under test | 20 |
| 3.2.2 | Δ Q Adapter | 20 |
| 3.2.3 | Inserting probes in Erlang - From spans to outcome instances | 20 |
| 3.3 | Oscilloscope side | 21 |
| 3.3.1 | Server | 21 |

| | | |
|----------|--|-----------|
| 3.3.2 | ΔQ Oscilloscope | 21 |
| 3.3.3 | Inserting probes in the oscilloscope | 21 |
| 3.4 | Triggers | 22 |
| 3.4.1 | Snapshot | 23 |
| 3.5 | Sliding execution windows | 23 |
| 3.5.1 | Sampling window | 23 |
| 3.5.2 | Polling window | 23 |
| 4 | Oscilloscope: User level concepts | 25 |
| 4.1 | ΔQSD concepts | 25 |
| 4.1.1 | Representation of a ΔQ | 25 |
| 4.1.2 | $dMax = \Delta t \cdot N$ | 26 |
| 4.1.3 | QTA | 28 |
| 4.1.4 | Confidence bounds | 28 |
| 4.2 | ΔQ display | 29 |
| 4.3 | Outcome diagram | 30 |
| 4.3.1 | Causal link | 30 |
| 4.3.2 | Sub-outcome diagrams | 30 |
| 4.3.3 | Outcomes | 31 |
| 4.3.4 | Operators | 31 |
| 4.3.5 | Limitations | 32 |
| 4.4 | Dashboard | 32 |
| 4.4.1 | Sidebar | 32 |
| 4.4.2 | Plots window | 33 |
| 4.5 | Triggers | 33 |
| 4.5.1 | Load | 33 |
| 4.5.2 | QTA | 33 |
| 5 | Oscilloscope: implementation | 34 |
| 5.1 | ΔQSD implementation | 34 |
| 5.1.1 | $dMax$ | 35 |
| 5.1.2 | Operations | 35 |
| 5.1.3 | Confidence bounds | 36 |
| 5.1.4 | Rebinning | 36 |
| 5.2 | Adapter | 37 |
| 5.2.1 | API | 37 |
| 5.2.2 | Handling outcome instances | 39 |
| 5.2.3 | TCP connection | 40 |
| 5.3 | Parser | 41 |
| 5.3.1 | ANTLR | 41 |
| 5.3.2 | Grammar | 41 |
| 5.4 | Oscilloscope GUI | 42 |
| 6 | Application on synthetic programs | 43 |
| 6.1 | System with sequential composition | 43 |
| 6.1.1 | System composition | 44 |
| 6.1.2 | Determining parameters dynamically | 44 |

| | | |
|----------|---|-----------|
| 6.2 | Detecting slower workers in workers | 49 |
| 6.2.1 | First to finish application | 49 |
| 6.2.2 | All to finish application | 51 |
| 7 | Performance study | 53 |
| 7.1 | Convolution performance | 53 |
| 7.2 | ΔQ adapter performance | 54 |
| 7.3 | GUI plotting performance | 55 |
| 8 | Conclusions and future work | 57 |
| 8.1 | Future improvements | 58 |
| 8.1.1 | Oscilloscope improvements | 58 |
| 8.1.2 | Adapter improvements | 59 |
| 8.1.3 | Real applications | 59 |
| 8.1.4 | Licensing limitations | 59 |
| A | Tests specifications | 63 |
| B | User manual | 64 |
| B.1 | Sidebar: Outcome diagram and plots | 64 |
| B.1.1 | Creating the system | 64 |
| B.1.2 | Saving the system definition | 64 |
| B.1.3 | Loading the system definition | 64 |
| B.1.4 | Managing the plots | 65 |
| B.1.5 | Removing a plot | 66 |
| B.2 | Sidebar: Probes settings | 66 |
| B.2.1 | Setting a QTA | 66 |
| B.2.2 | Setting the parameters of a probe | 66 |
| B.3 | Triggers | 68 |
| B.4 | Instrumenting the Erlang application | 68 |
| B.4.1 | Including the adapter | 68 |
| B.4.2 | Starting spans | 68 |
| B.4.3 | Ending spans | 69 |
| B.4.4 | Failing outcome instances | 69 |
| B.5 | Establishing connection to the oscilloscope | 69 |
| I | Source Code Appendix | 70 |
| C | C++ Source Files | 71 |
| C.1 | Root | 71 |
| C.1.1 | Application.cpp | 71 |
| C.2 | dashboard | 75 |
| C.2.1 | ColorRegistry.cpp | 75 |
| C.2.2 | CustomLegendEntry.cpp | 76 |
| C.2.3 | CustomLegendPanel.cpp | 76 |
| C.2.4 | DQPlotController.cpp | 78 |

| | | |
|----------|-------------------------------------|------------|
| C.2.5 | DQPlotList.cpp | 87 |
| C.2.6 | DelaySettingsWidget.cpp | 89 |
| C.2.7 | DeltaQPlot.cpp | 92 |
| C.2.8 | MainWindow.cpp | 94 |
| C.2.9 | NewPlotList.cpp | 100 |
| C.2.10 | ObservableSettings.cpp | 101 |
| C.2.11 | PollingRateWidget.cpp | 102 |
| C.2.12 | QTAInputWidget.cpp | 103 |
| C.2.13 | Sidebar.cpp | 105 |
| C.2.14 | SnapshotViewerWindow.cpp | 108 |
| C.2.15 | StubControlWidget.cpp | 112 |
| C.2.16 | SystemCreationWidget.cpp | 114 |
| C.2.17 | TriggersTab.cpp | 117 |
| C.3 | diagram | 122 |
| C.3.1 | Observable.cpp | 122 |
| C.3.2 | Operator.cpp | 125 |
| C.3.3 | Outcome.cpp | 127 |
| C.3.4 | Probe.cpp | 127 |
| C.3.5 | System.cpp | 129 |
| C.4 | maths | 132 |
| C.4.1 | ConfidenceInterval.cpp | 132 |
| C.4.2 | DeltaQ.cpp | 135 |
| C.4.3 | DeltaQOperations.cpp | 140 |
| C.4.4 | Snapshot.cpp | 145 |
| C.4.5 | TriggerManager.cpp | 148 |
| C.4.6 | Triggers.cpp | 149 |
| C.5 | parser | 150 |
| C.5.1 | SystemBuilder.cpp | 150 |
| C.5.2 | SystemParserInterface.cpp | 156 |
| C.6 | server | 157 |
| C.6.1 | Server.cpp | 157 |
| D | C++ Header Files | 166 |
| D.1 | Root | 166 |
| D.1.1 | Application.h | 166 |
| D.2 | dashboard | 167 |
| D.2.1 | ColorRegistry.h | 167 |
| D.2.2 | CustomLegendEntry.h | 167 |
| D.2.3 | CustomLegendPanel.h | 168 |
| D.2.4 | DQPlotController.h | 169 |
| D.2.5 | DQPlotList.h | 171 |
| D.2.6 | DelaySettingsWidget.h | 173 |
| D.2.7 | DeltaQPlot.h | 174 |
| D.2.8 | MainWindow.h | 177 |
| D.2.9 | NewPlotList.h | 179 |
| D.2.10 | ObservableSettings.h | 180 |

| | | |
|----------|-----------------------------------|------------|
| D.2.11 | PollingRateWidget.h | 180 |
| D.2.12 | QTAInputWidget.h | 181 |
| D.2.13 | Sidebar.h | 183 |
| D.2.14 | SnapshotViewerWindow.h | 185 |
| D.2.15 | StubControlWidget.h | 186 |
| D.2.16 | SystemCreationWidget.h | 187 |
| D.2.17 | TriggersTab.h | 189 |
| D.3 | diagram | 191 |
| D.3.1 | Observable.h | 191 |
| D.3.2 | Operator.h | 193 |
| D.3.3 | OperatorType.h | 195 |
| D.3.4 | Outcome.h | 195 |
| D.3.5 | Probe.h | 195 |
| D.3.6 | Sample.h | 197 |
| D.3.7 | System.h | 197 |
| D.4 | maths | 199 |
| D.4.1 | ConfidenceInterval.h | 199 |
| D.4.2 | DeltaQ.h | 200 |
| D.4.3 | DeltaQOperations.h | 202 |
| D.4.4 | DeltaQRepr.h | 203 |
| D.4.5 | QTA.h | 204 |
| D.4.6 | Snapshot.h | 205 |
| D.4.7 | TriggerManager.h | 207 |
| D.4.8 | TriggerTypes.h | 209 |
| D.4.9 | Triggers.h | 209 |
| D.5 | parser | 210 |
| D.5.1 | SystemBuilder.h | 210 |
| D.5.2 | SystemErrorListener.h | 212 |
| D.5.3 | SystemParserInterface.h | 212 |
| D.6 | server | 213 |
| D.6.1 | Server.h | 213 |
| E | Build Configuration Files | 216 |
| E.1 | Root | 216 |
| E.1.1 | CMakeLists.txt | 216 |
| E.2 | dashboard | 217 |
| E.2.1 | CMakeLists.txt | 217 |
| E.3 | diagram | 218 |
| E.3.1 | CMakeLists.txt | 218 |
| E.4 | maths | 218 |
| E.4.1 | CMakeLists.txt | 218 |
| E.5 | parser | 219 |
| E.5.1 | CMakeLists.txt | 219 |
| E.6 | server | 221 |
| E.6.1 | CMakeLists.txt | 221 |
| F | Erlang Source Files | 222 |

| | | |
|----------|------------------------------------|------------|
| F.1 | Root | 222 |
| F.1.1 | dqsd_otel.erl | 222 |
| F.1.2 | dqsd_otel_app.erl | 226 |
| F.1.3 | dqsd_otel_sup.erl | 227 |
| F.1.4 | dqsd_otel_tcp_client.erl | 227 |
| F.1.5 | dqsd_otel_tcp_server.erl | 229 |
| G | Erlang Application Files | 233 |
| G.1 | Root | 233 |
| G.1.1 | dqsd_otel.app.src | 233 |

Chapter 1

Introduction

1.1 Context

Δ QSD is an industrial-strength approach for large-scale system design that can predict performance and feasibility early on in the design process. Developed over 30 years by a small group of people around Predictable Network Solutions Ltd, the paradigm has been applied in various industrial-scale problems with huge success and large savings in costs [1]. Moreover, it is the basis of Broadband forum's TR452 standard series, used in instrumenting data networks [2].

Modern software development practices successfully fail to adequately consider essential quality requirements or even to consider properly whether a system can actually meet its intended outcomes, particularly when deployed at scale, the Δ QSD paradigm addresses this problem! [3]

Δ QSD has important properties which make its application to distributed projects interesting, it supports:

- A compositional approach that considers performance and failure as first-class citizens.
- Stochastic approach to capture uncertainty throughout the design approach.
- Performance and feasibility can be predicted at high system load for partially defined systems [1].

While the paradigm has been successfully applied in **a posteriori** analysis, there is no way yet to analyse a distributed system which is running in real time with Δ QSD! This is where the Δ Q oscilloscope comes in.

1.2 Objective

This project will develop a practical tool, the **Δ Q oscilloscope**, for the Erlang developer community.

The Erlang language and Erlang/OTP platform are widely used to develop distributed applications that must perform reliably under high load [4]. The tool will provide useful information for these applications both for understanding their behaviour, for diagnosing performance issues, and for optimising performance over their lifetime. [5]

The ΔQ Oscilloscope will perform statistical computations to show real time graphs about the performance of system components. With the oscilloscope prototype we will present in this paper, we are aiming to show that the ΔQSD paradigm is not only a theoretical paradigm, but it can be employed in a tool to diagnose distributed systems. Its application can then be further extended to large systems once the oscilloscope is refined.

The oscilloscope targets large distributed applications handling many independent tasks where performance and reliability are important. [1]

1.3 Previous work

The ΔQSD paradigm has been formalised across different papers [3] [6] and was brought to the attention of engineers via tutorials [1] and to students at Université Catholique de Louvain. [7]

A Jupyter notebook workbench has been made available on GitHub [8], it shows real time ΔQ graphs for typical outcome diagrams but is not adequate to be scaled to real time systems, it is meant as an interactive tool to show how the ΔQSD paradigm can be applied to real life examples.

Observability tools such as Erlang tracing [9] and OpenTelemetry [10] lack the notions of failure as defined in ΔQSD , which allows detecting performance problems early on, we base our program on OpenTelemetry to incorporate already existing notions of causality and observability to augment their capabilities and make them suitable to work with the ΔQSD paradigm.

1.4 Contributions

There are a few contributions that make the master thesis and thus, the oscilloscope, possible:

- A graphical interface to display probes' ΔQ s.
- An Erlang OpenTelemetry adapter to give OpenTelemetry spans a notion of failure and to communicate with the oscilloscope.
- The implementation of ΔQSD concepts from theory to practice, including a textual syntax to create outcome diagrams, derived from the original algebraic syntax.
- An efficient convolution algorithm based on the FFTW3 library.
- A system of triggers to catch rare events when system behaviour fails to meet quality requirements, creating a snapshot of the system, giving the user insights

about their system's behaviour.

- Synthetic applications to test the effectiveness of Δ QSD on diagnosing running systems.

These contributions can show that the Δ QSD paradigm can be translated to real-time observation of running system. Furthermore, it proves that the paradigm has its practical applications and is not limited to a theoretical view of system design.

1.5 Roadmap

The following thesis will give the reader everything that is needed to use the oscilloscope and exploit it to its full potential.

We divided the thesis in multiple chapters, below is the roadmap of the content:

- The background chapter gives the reader an extensive background into the theoretical foundations of Δ QSD, which are the basis of the oscilloscope and are fundamental to understand how to correctly use and analyse the output given by the oscilloscope. Secondly, an introduction to OpenTelemetry, the library we base our Erlang adapter on. Lastly, we provide what we believe are the current limitations of the observability tool and how we plan to tackle them.
- The design chapter first provides the "measurement concepts", these concepts serve as an introduction to understand the following chapters and as a bridge from OpenTelemetry to the oscilloscope. We then delve on how the parts of the system interact together and how to correctly apply the concepts we just introduced in the application side and in the oscilloscope. Lastly, after having introduced the oscilloscope, we explain more abstract concepts implemented in it, like triggers and sliding windows.
- We then present the oscilloscope in two different chapters, first providing "user level concepts" of how Δ QSD is used and what the user should expect visually from the dashboard. It also provides a complete explanation on how to write outcome diagrams and what the different sections on the dashboard do. Secondly, a more low level explanation, which goes into more technical details of the parts that compose the oscilloscope and the mathematical explanations of Δ QSD concepts explained in the previous chapter.
- We then provide synthetic applications which have been tested with the oscilloscope that demonstrate the usefulness of the oscilloscope in a distributed setting. We also perform evaluations of the performance of the different parts we have developed to understand the overhead that are present.

We end by providing future possibilities which can be explored to improve the application. In the appendix, we provide a user manual to help users use the oscilloscope, along with C++ and Erlang source code of the oscilloscope and the adapter. The oscilloscope (<https://github.com/fnieri/DeltaQOscilloscope>) and adapter (https://github.com/fnieri/dqsd_otel) can be found on GitHub as open source projects.

Chapter 2

Background

This chapter aims to provide firstly a complete background of the concepts key to understanding the Δ QSD paradigm.

Secondly, we provide a comprehensive background into the observability solutions that have been explored for the oscilloscope, delving deeper into OpenTelemetry and its macros.

We finish by explaining what we believe are the current limitations of OpenTelemetry and explaining where the paradigm and the oscilloscope comes in.

2.1 An overview of Δ QSD

Δ QSD is a metrics-based, quality-centric paradigm that uses formalised outcome diagrams to explore the performance consequences of design decisions. [6]

Key concepts of Δ QSD are **quality attenuation** (Δ Q) and **outcome diagram** [1].

Outcome diagrams capture dependency and causality properties of the system. The Δ QSD paradigm derives bounds on performance expressed as probability distribution, encompassing all possible executions of the system. [6]

The following sections are a summary of multiple articles and presentations formalising the paradigm.

2.1.1 Outcome

An outcome o is a specific system behaviour that can be observed to start at some point in time and **may** be observed to complete at some later time. [2] Formally, what the system obtains by performing one of its tasks. One task corresponds to one outcome and viceversa. When an outcome is performed, it means that the task of an outcome is performed.

The particularity of outcomes is that they can represent multiple levels of granularity, suppose an outcome may be beyond the current system's control (e.g. a database/cloud request), is non-atomic (can be broken down in multiple sub-outcomes). These outcomes

can be represented as black-boxes (you can observe their start and end, but do not know what is being executed), as the system gets refined, these outcomes can then be refined to model a single outcome or multiple outcomes, if needed.

Even though these outcomes are defined as "black boxes", they still have timeliness constraints like any other outcome. [6]

Observables Each outcome has two starting sets of events: the starting sets and the ending sets. Such sets are called the *observables*. Once an event from the starting set occurs, there is no guarantee that a corresponding event in the terminating set will occur within the duration limit (required time to complete). An observable is *done* when it occurs during the time limit. [3]

Outcome instance An outcome instance is the result of an execution of an outcome given a starting event e_{in} and an end event e_{out} . [3]

Graphical Representation Outcomes are represented as circles, with the starting and terminating set of events being represented by boxes.

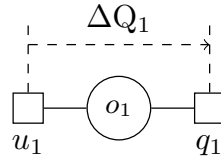


Figure 2.1: The outcome (circle) and the starting set (left) and terminating set (right) of events. [6]

2.1.2 Quality attenuation (ΔQ)

Assume a component C which receives a message m_{in} and outputs a message m_{out} after a delay d . Over multiple executions, we will have observed multiple delays which can be represented as a cumulative definition where p percent of delays have delay $\leq d$. [3]

ΔQ is a cumulative distribution function that defines both *latency* and *failure probability* between a start and end event. [1]

In an ideal system, an outcome would deliver a desired behaviour without error, failure, delay, but this is not the case. The quality of an outcome response "attenuated to the relative ideal" (the cumulative distribution function) is called "quality attenuation" (ΔQ) and can depend on many factors (geographical, physical ...). Its distribution may be modelled by a random variable.

As ΔQ captures deviation from ideal behavior and incorporates delay, which is a continuous random variable, and failures/timeouts, which are discrete variables, it can be described mathematically as an *Improper Random Variable*, where the probability of a finite or bounded delay < 1 .

$\Delta Q(x)$ is the probability that an outcome O occurs in time $t \leq x$. The *intangible mass* $1 - \lim_{x \rightarrow \infty} \Delta Q(x)$ of a ΔQ will encode the probability of failure/timeout/exception occurring. [6]

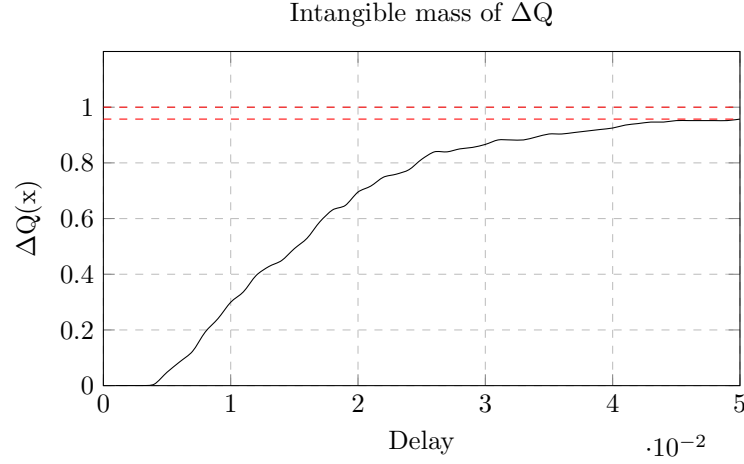


Figure 2.2: Intangible mass (red) of a ΔQ with failure rate of about 5%

2.1.3 Failure semantics

In the CDF representation of a ΔQ , there is an f percent probability that the delay is infinite, this is what failure models. Concretely, it means that an input message m_{in} **has no output message** m_{out} . [3]

Combining delay and failure in a single quantity is what makes ΔQSD a great choice to explore feasibility in system design. [1]

2.1.4 Partial ordering

A CDF of a ΔQ is *less than* the other if its CDF is everywhere to the left and above the other. Mathematically, it is a partial order.

If two ΔQ s intersect, they are not ordered. [1]

2.1.5 Timeliness

Timeliness is defined as a relation between an observed ΔQ_{obs} and a required ΔQ_{req} . Timeliness is delivering results within required time bounds (sufficiently often).

A system *satisfies timeliness* if $\Delta Q_{obs} \leq \Delta Q_{req}$. [3]

2.1.6 QTA, required ΔQ

The *Quantitative Timeliness Agreement* (QTA) maps objective measurements to the subjective perception of application performance. It specifies what the base system does and its limits. [2]

Slack There is performance *slack* when a ΔQ is strictly less than the requirement,

Hazard There is performance *hazard* when a ΔQ is strictly greater than the requirement. [6]

QTA example : Imagine a system where 25% of the executions should take < 15 ms, 50% < 25 ms and 75% < 35 ms, all queries have a maximum delay of 50ms and 5% of executions can timeout, the QTA can be represented as a step function.

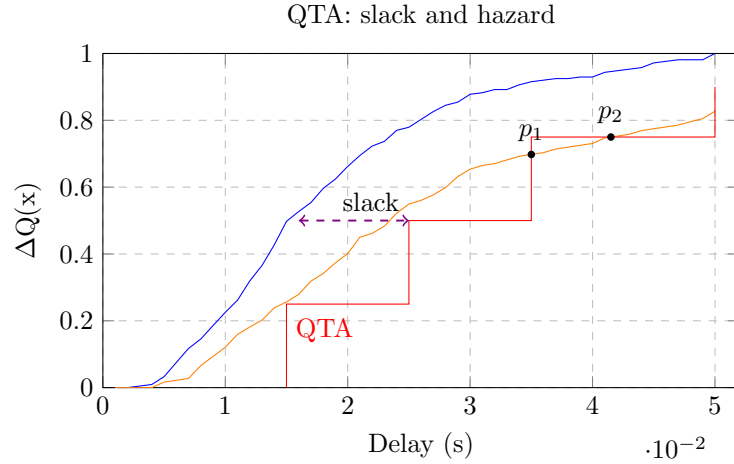


Figure 2.3: The system in blue is showing slack and satisfies the requirement. The system in orange is showing signs that it cannot handle the stress, it is not respecting the system requirements imposed by the QTA.

2.1.7 Outcome diagram

An outcome diagram is central to capture the causal relationships between the outcomes. It shows the causal connections between all the outcomes we are interested in, and it allows computing the ΔQ for the whole system [1]. It maps a system's behaviour as seen from outside to concrete outcomes [3]. There are four different operators that represent the relationships between outcomes. [1]

Sequential composition

If we assume two outcomes O_A , O_B where the end event of O_A is the start event of O_B , the two outcomes can be sequentially composed. The total delay ΔQ_{AB} is given by the convolution of the PDFs of O_A and O_B ($O_A \otimes O_B$).

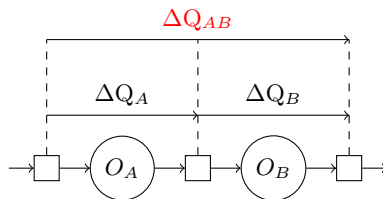


Figure 2.4: Sequential composition of O_A and O_B .

Where convolution (\circledast) between two PDF is:

$$PDF_{AB}(t) = \int_0^t PDF_A(\delta) \cdot PDF_B(t - \delta) d\delta \quad (2.1)$$

Thus, ΔQ_{AB} :

$$\Delta Q_{AB} = PDF_A \circledast PDF_B \quad (2.2)$$

Convolution is the only operation which is based on the PDFs, the following operations are based on the CDF of the ΔQ s (hence the use of the ΔQ notation).

First to finish (FTF)

If we assume two independent outcomes O_A, O_B with the same start event, first-to-finish occurs when at least one end event occurs, it can be calculated as:

$$\begin{aligned} \Delta Q_{FTF(A,B)} &= Pr[d_A > t \wedge d_B > t] \\ &= Pr[d_A > t] \cdot Pr[d_B > t] = (1 - \Delta Q_A) \cdot (1 - \Delta Q_B) \\ \Delta Q_{FTF(A,B)} &= \Delta Q_A + \Delta Q_B - \Delta Q_A \cdot \Delta Q_B \end{aligned} \quad (2.3)$$

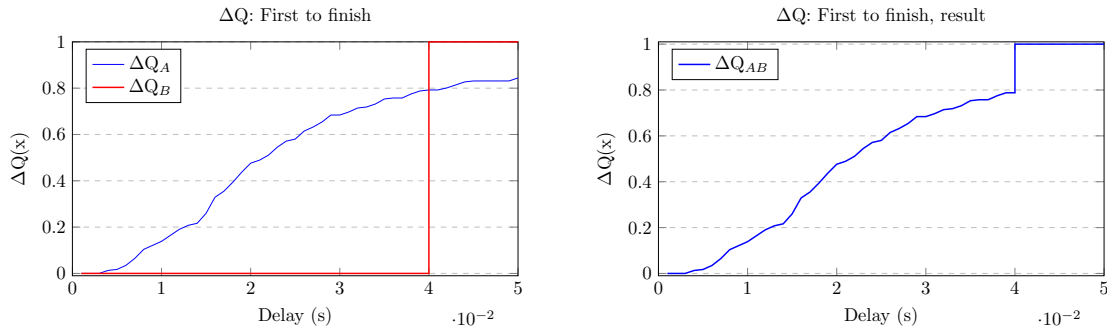
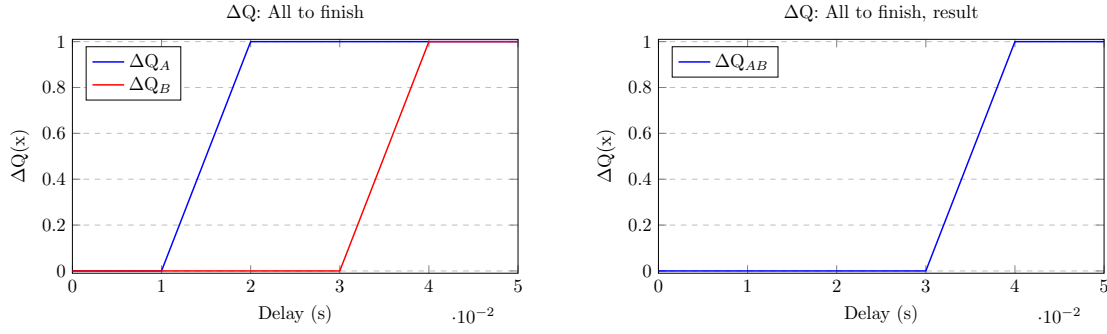


Figure 2.5: Left: $\Delta Q_{(A,B)}$. Right: $FTF_{(A,B)} = \Delta Q_{AB}$

All to finish (ATF)

If we assume two independent outcomes O_A, O_B with the same start event, all-to-finish occurs when both end events occur, it can be calculated as:

$$\begin{aligned} \Delta Q_{ATF(A,B)} &= Pr[d_A \leq t \wedge d_B \leq t] \\ &= Pr[d_A \leq t] \cdot Pr[d_B \leq t] = \Delta Q_A \cdot \Delta Q_B \\ \Delta Q_{ATF(A,B)} &= \Delta Q_A \cdot \Delta Q_B \end{aligned} \quad (2.4)$$


 Figure 2.6: Left: $\Delta Q_{(A,B)}$. Right: $ATF_{(A,B)} = \Delta Q_{AB}$

Probabilistic choice (PC)

If we assume two possible outcomes O_A and O_B and exactly one outcome is chosen during each occurrence of a start event and:

- O_A happens with probability $\frac{p}{p+q}$
- O_B happens with probability $\frac{q}{p+q}$

$$\Delta Q_{PC}(A, B) = \frac{p}{p+q} \Delta Q_A + \frac{q}{p+q} \Delta Q_B \quad (2.5)$$

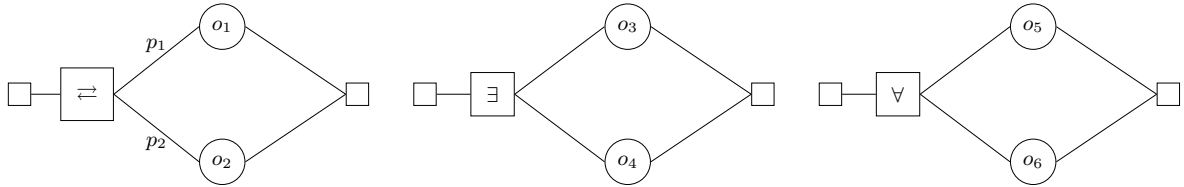


Figure 2.7: The possible operators in an outcome diagram: Probabilistic choice, first-to-finish, all-to-finish

First-to-finish, All-to-finish and probabilistic-choice are calculated on the CDF of the ΔQ of their components.

These operators can be assembled together to create an outcome diagram, later on, we will see how one can go from the graphical representation to outcome diagrams which can be used in the ΔQ oscilloscope.

2.1.8 Outcome diagrams refinement

An important feature of outcome diagrams is the ability to be able to design a system even with "*black-boxes*", before the complete details of it are known.

An outcome diagram can be "unboxed" by refining the outcomes that compose it. We can adapt a situation described in Mind your Outcomes to understand how refinement can allow the user to have a very precise representation of a system. [6]

We first start with a black-box, unnamed outcome with start event A and end event Z , somewhere in the system. The first refinement step would be giving the outcome a name.

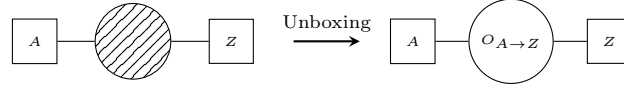


Figure 2.8: Refinement from black box to named outcome.

The system can be further refined by adding outcomes that represent tasks. For example, the engineer might believe that it will take two tasks to get from A to Z . We can then add another outcome, sequentially composed, to represent this situation.

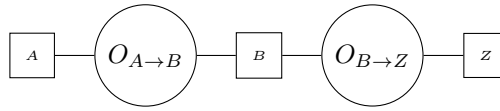


Figure 2.9: Further refinement from one task to two tasks.

We can also model the chance of executing two tasks as a probabilistic choice, where there is p_2 probability that the execution from A to Z will execute two tasks. The outcome diagram can be refined as a probabilistic choice.

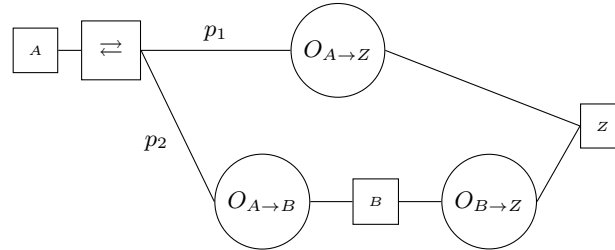


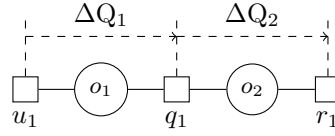
Figure 2.10: Refinement as probabilistic choice of executing either one or two tasks.

In essence, the refinement could model a very fine-grained representation of the system by further refining the system, to represent the possibility of executing n tasks. This demonstrates the power of outcome diagrams to represent system diagrams with high precision. They can help explore design decisions thanks to outcomes and operators.

2.1.9 Independence hypothesis

An important aspect of sequential composition is the assumption of outcomes having independent behaviour. Let us explain the following assumption clearly.

Assume two sequentially composed outcomes o_1 , o_2 running on the same processor. The overall delay of execution can be observed from the start event of o_1 (u_1) to the end event of o_2 (r_1).



At low load, the two components behavior will be independent, the system will behave **linearly**. According to the superposition principle, the overall delay will be the sum of the two delays, as will the processor utilisation. [11]

When load increases, the two components will start to show dependent behaviour due to the processor utilisation increasing. The ΔQ of the observed total delay will then deviate from the sum of the two delays ($\Delta Q_1 \circledast \Delta Q_2$).

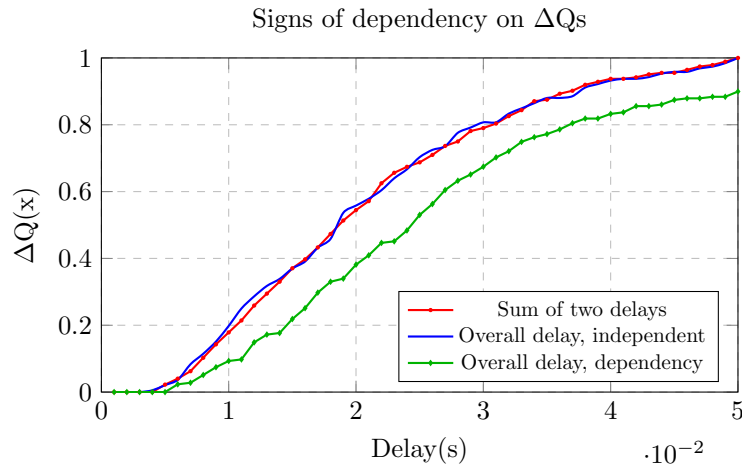


Figure 2.11: When the components are independent, the sum of the two delays (blue) and the overall delay (red) can be superposed.

When o_1 and o_2 show initial signs of dependency, the overall delay (green) can be seen deviating from the sum of the two delays.

When the system is far from being overloaded, the effect is noticeable thanks to ΔQSD . As the cliff edge of overload is approached, the nonlinearity will increase [5]. These theoretical results can be observed in practice in the oscilloscope. We will explore such cases in the synthetic applications section.

2.2 Observability

Observability refers to the ability to understand the internal state by examining its output, in the context of a distributed system, being able to understand the internal state of the system by examining its telemetry data. [12]

In the case of the Erlang programming language, we explain below two tools that can be used to observe an Erlang program.

2.2.1 erlang:trace

The Erlang programming language gives the users different ways to observe the behaviour of a system, one of those is the function `erlang:trace/3`. The erlang run-time system exposes several trace points that can be observed, observing the trace points allows users to be notified when they are triggered [9]. One can observe function calls, messages being sent and received, process being spawned, garbage collecting

```
-spec trace(PidPortSpec, How, FlagList) -> integer()
when
    PidPortSpec ::
        pid() |
        port() |
        all | processes | ports | existing | existing_processes |
        ↪ existing_ports | new |
        new_processes | new_ports,
    How :: boolean(),
    FlagList :: [trace_flag()].
```

Figure 2.12: erlang:trace/3 specification

Nevertheless, in Erlang Tracing there is no default way to follow a message and get its whole execution trace. This is a missing feature that is crucial for observing a program functioning and being able to connect an application to our oscilloscope. This is where the OpenTelemetry framework comes in.

2.2.2 OpenTelemetry

OpenTelemetry is an open-source, vendor-agnostic observability framework and toolkit designed to generate, export and collect telemetry data, in particular traces, metrics and logs. OpenTelemetry provides a standard protocol, a single set of API and conventions and lets the user own the generated data, allowing to switch between observability backends freely. [12]

OpenTelemetry is available for a plethora of languages [13], including Erlang, although, as of writing this, only traces are available in Erlang [14].

The Erlang Ecosystem Foundation has a working group focused on evolving the tools related to observability, including OpenTelemetry and the runtime observability monitoring tools [15].

Traces

Traces are why we are basing our program on top of OpenTelemetry, traces follow the whole "path" of a request in an application, traces are comprised of one or more spans. Traces can propagate to multiple services and record multiple paths in different microservices [16].

Span A span is a unit of work or operation. Spans can be correlated to each other and can be assembled into a trace. The spans can have a hierarchy, where *root spans* represent a request from start to finish [16]. We will see in later sections how this can relate to what the oscilloscope does.

The notion of spans and traces allows us to follow the execution of a "request" and carry a context. Spans can be linked to mark causal relationships between multiple spans [17], as is the case for root spans. This relation can be represented in the oscilloscope via probes and we will present an example in following sections.

```
{
  "name": "oscilloscope-span",
  "context": {
    "trace_id": "5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "5fb397be34d26b51"
  },
  "parent_id": "0515505510cb55c13",
  "start_time": "2022-04-29T18:52:58.114304Z",
  "end_time": "2022-04-29T22:52:58.114561Z",
  "attributes": {
    "http.route": "some_route"
  },
}
```

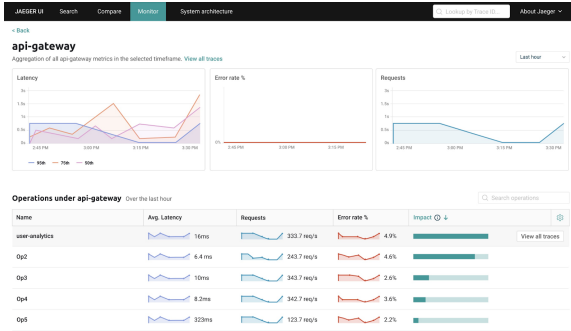
Figure 2.13: Example of span with a parent, indicating that child and parent are related and are both part of the same trace [17].

Monitoring OpenTelemetry spans

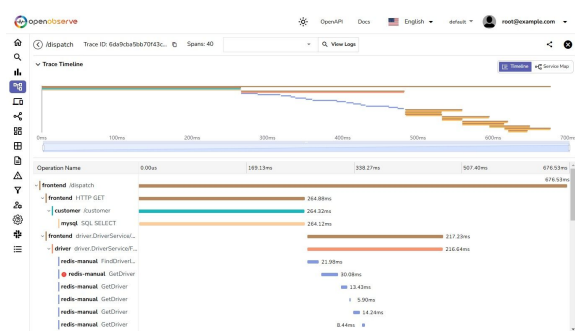
OpenTelemetry gives the possibility to export traces to backends and monitoring such as Jaeger, Zipkin, Datadog [18]. There, a user can monitor their workflows, analyse dependencies, troubleshoot their programs by observing the flow of the requests. These monitoring tools give extensive details about a running system, but may fail to capture essential requirements and performances issues early enough.

Our oscilloscope is a kind of monitoring tool, one that gives precise statistical insights about a running system. It is clear that the oscilloscope does not have the same capabilities as Datadog [19] might have, where you can observe cloud instances, instances cost, dependency graphs ... but the oscilloscope can nevertheless provide precise insights about dependency, overload thanks to the Δ QSD paradigm.

This is also the reason that we work alongside OpenTelemetry, the oscilloscope can be put next to a monitoring tool where one might export spans to. An engineer might consult the main monitoring tool to get the global picture of a running app, and the oscilloscope to give more precise insights to understand the system's behaviour.



(a) Jaeger interface [20].



(b) A span analysis on OpenObserve [21]

Span macros

OpenTelemetry provides macros to start, end and interact with spans in Erlang, the following code excerpts are taken from the instrumentation wiki. [14]

?with_span ?with_span creates active spans. An active span is the span that is currently set in the execution context and is considered the "current" span for the ongoing operation or thread. [22]

```
parent_function() ->
    ?with_span(parent, #{}, fun child_function/0).

child_function() ->
    %% this is the same process, so the span parent set as the active
    %% span in the with_span call above will be the active span in
    ↪ this function
    ?with_span(child, #{},
        fun() ->
            %% do work here. when this function returns, child
            ↪ will complete.
        end).
```

?start_span ?start_span creates a span which isn't connected to a particular process, it does not set the span as the current active span.

```
SpanCtx = ?start_span(child),
Ctx = otel_ctx:get_current(),
proc_lib:spawn_link(fun() ->
    otel_ctx:attach(Ctx),
    ?set_current_span(SpanCtx),
    %% do work here
    ?end_span(SpanCtx)
end),
```

?end_span ?end_span ends a span started with ?start_span

2.3 Current observability problems

A legitimate question to pose would be why one would need an additional tool to observe their system, monitoring tools are already plenty and provide useful insights into an application’s behaviour. While they may seem adequate to provide a global oversight of applications, they fail to diagnose real time problems like overload, dependent behaviour early enough and in a quick manner.

The problem we are trying to tackle can be described by the following situation: Imagine an Erlang application instrumented with OpenTelemetry, suddenly, the application starts slowing down, and the execution of a function takes 10 seconds instead of the usual 1 second. Between its start and its end, the user instrumenting the application sees nothing in their dashboard.

This is a big problem! One would like to know right away if something is wrong with their application, better! Even before problems are apparent. This is where the Δ QSD paradigm and the Δ Q oscilloscope come in handy.

By leveraging Δ QSD notion of failure and QTAs, problems can be detected right away in the oscilloscope.

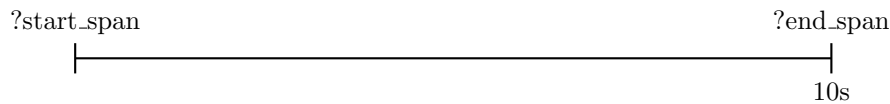


Figure 2.15: Execution of a long span in OpenTelemetry, the user will only be notified after 10 seconds that the function has ended (and taken too long).

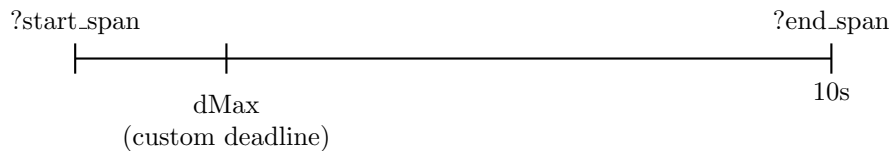


Figure 2.16: Execution of a long span in OpenTelemetry, the *dMax* deadline allows knowing that the span has taken too long.

2.3.1 Handling of long spans

OpenTelemetry presents a bigger problem, what happens when there are long-running spans? Worse, what happens when spans are not actually terminated?

OpenTelemetry limits the length of its spans, moreover, those who are not terminated are lost and not exported. Why? Failed executions are those that tell more about a program’s execution!

If the span is the parent/root span, its effect could trickle down to child spans. We can quickly see how this becomes problematic, all the information about an execution of your program ...lost. Moreover, a span could not be terminated for trivial reasons:

refreshing a tab, network failures, crashes ... [13]. There are a few hacks that can be implemented, having shorter spans, carrying data in child spans, saving spans in a log to track spans which were not ended to manually set an end time; why the need to circumvent limitations when observing a system?

We believe that the adapter we provide can be a great start to improve observability requirements surrounding OpenTelemetry. We will show in the evaluation on synthetic applications how Δ QSD's notion of failure can help to detect overload problems in running systems right away.

Chapter 3

Design

This chapter aims to first extend the concepts of Δ QSD, giving more insights into how the systems need to be instrumented to correctly work together, and how the different parts need to be integrated to interact together.

- We first provide concepts of probes, we extend the Δ QSD notion of failure and describe how time series will work in our oscilloscope, this part is crucial to understand how the measurements are done in real time.
- We then split the design of the oscilloscope in two. First explaining the Erlang side, where the system to be tested is. Secondly, we explain the C++ side. Both chapters explain how probes can be inserted and made to work together.
- Lastly, we provide high level concepts of triggers and execution windows, the key elements of the oscilloscope.

3.1 Measurement concepts

3.1.1 Probes

A system instrumented with OpenTelemetry has spans and traces to observe the execution of an operation [17]. The same level of observability must be assured in the oscilloscope, this is why we provide the concept of probes, which, like spans, follow an execution from start to end.

To observe a system, we must put probes in it. For each outcome of interest, a probe (observation point) is attached to measure the delay of the outcome, like one would in a true oscilloscope [5].

Consider the figure below, a probe is attached at every component to measure their Δ Qs (c_2, c_3), Another probe (p_1) is inserted at the beginning and end of the system to measure the global execution delay. Thanks to this probe, the user can observe the Δ Q "*observed at p_1* ", which is the Δ Q which was calculated from the data received by inserting probe p_1 . The Δ Q "*calculated at p_1* " is the resulting Δ Q from the convolution of the observed Δ Qs at c_2 and c_3 .

Probe p_1 is the equivalent of a "root/parent span" which observe the whole execution of c_1, c_2 , while p_2 and p_3 are child spans which represent single instances of execution.

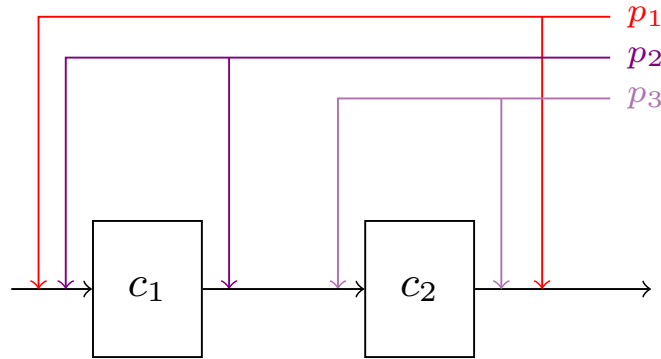


Figure 3.1: Probes inserted in a component diagram. In an applications instrumented with OpenTelemetry, p_1 could be considered the root span, c_1 and c_2 its children spans sharing a causal link.

3.1.2 Extending failure

Recall the definition of failure: *"an input message m_{in} that has no output message m_{out} "* [3]. In the previous section 2.16, we also introduced the notion of a maximum delay.

By extending the notion of failure to include $dMax$, we can know right away when execution is straying away from engineer defined behaviour, avoiding having to wait until the execution is done. In ΔQSD , an execution may as well take 10 or 15 seconds, but if the delay of execution is $> dMax$, we consider that **failed** right away, we do not need to know the total execution time, the execution has already taken too much! Moreover, the full span will be exported regardless to monitoring tools which were set up by the user.

The user can observe both real time information with ΔQSD notion of failure on the ΔQ oscilloscope, and observe those spans in their monitoring tools if they wish.

The notion of failure is extended to the following definition:

"An input message m_{in} that has no output message m_{out} after $dMax$ "

We can leverage this new definition to observe the system and the ΔQ s in real time.

3.1.3 Time series of outcome instances

Consider a probe p with two distinct sets of events, the starting set of events s and ending set of event e . The outcome instance of a message $m_s \rightarrow m_e$ contains:

- The probe's p name
- The start time t_s
- The end time t_e

- Its status
- Its elapsed time of execution

The instance has three possible statuses: **success**, **timeout**, **failure**, it can thus be broken down in the representations, based on its status:

- (t_s, t_e) : This representation indicates that the execution was successful ($t < dMax$).
- (t_s, \mathcal{T}) : This representation indicates that the execution has timed out ($t > dMax$). The end time and elapsed time is equal to $t_s + \text{timeout}$
- (t_s, \mathcal{F}) : This representation indicates the execution has failed given a user defined requirement (i.e. a dropped message given buffer overload in a queue system). It must not be confused with a program failure (crash), if a program crashes during the execution of event e , it will time out since the adapter will not receive an end message.

The **time series** of a probe is the sequence of n outcome instances and can then be easily modeled by ΔQ .

What can be considered a failed execution? Imagine a queue with a buffer: the buffer queue being full and dropping incoming messages can be modeled as a failure.

More generally, the choice of what is considered a failed execution is left up to the user who is handling the spans and is program-dependent. Exceptions or errors can be kinds of failure.

On another note, the way of handling errored spans in OpenTelemetry can differ from user to user, so the adapter will not handle ending and setting statuses for "failed" spans. [23]

In any case, by the new definition of failure, **timed out and failed will both be considered as a failure** in the calculation of a ΔQ .

3.2 Application side

Before delving deeper into the parts, we present the global system design diagram. We recognise two separate parts, the application side, where the Erlang system under test is, and the oscilloscope side, where the ΔQ oscilloscope receives information from the system under test to display graphs.

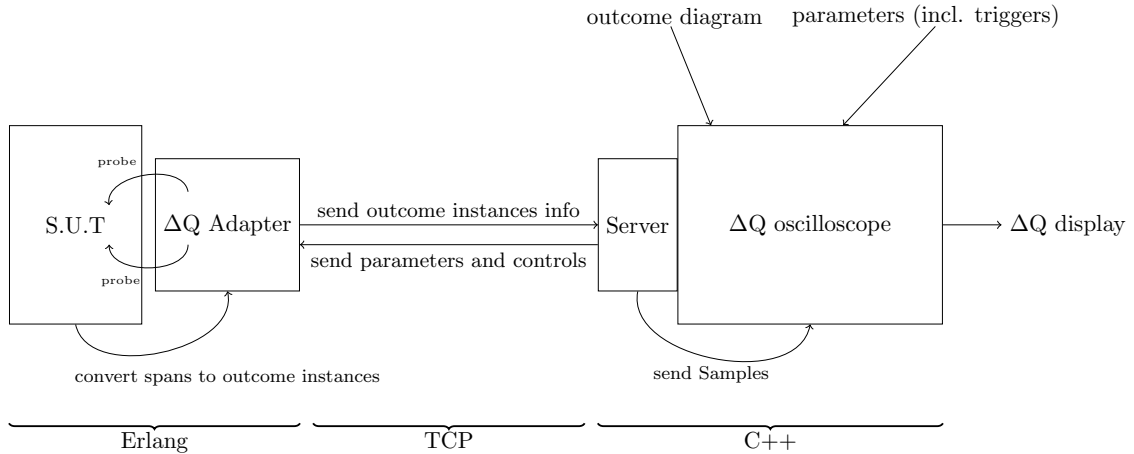


Figure 3.2: Global system design diagram.

3.2.1 System under test

The system under test (**S.U.T**) is the Erlang system the engineer wishes to observe, it ideally is a system which already is instrumented with OpenTelemetry. The ideal system where ΔQSD is more useful is a system that executes many independent instances of the same action [1].

3.2.2 ΔQ Adapter

The ΔQ adapter is the `dqsd_otel` Erlang application [24], it starts and ends OpenTelemetry spans and translates them to outcome instances which are useful for the oscilloscope. This can be done thanks to probes being attached to the system under test, like an oscilloscope would! The outcome instances end normally like OpenTelemetry spans or, additionally, can timeout, given a custom timeout (*dMax*), and fail, *according to user's definition of failure*.

Handling of OpenTelemetry spans which goes beyond starting and ending them is delegated to the user, who may wish to do further operations with their spans. The adapter is called from the system under test and communicates outcome instances data to the oscilloscope via TCP.

The adapter can receive messages from the oscilloscope, the messages are about updating probe's *dMax* or starting and stopping the sending of data to the oscilloscope.

3.2.3 Inserting probes in Erlang - From spans to outcome instances

OpenTelemetry spans are useful to carry context, attributes and baggage in a program. The plethora of attributes they have is nevertheless too much for the oscilloscope.

To get the equivalent of spans for the oscilloscope, the adapter needs to be called at the starting events of a probe to start an instance of a probe, and at the ending events to

end the outcome instance and send the data to the oscilloscope. The name given with "start_span" is the name of the probe.

```
% Start the outcome instance of probe. The call to dqsd_otel starts an  
→ OpenTelemetry span, as it contains a call to ?start_span(Name)  
{ProbeCtx, ProbePid} = dqsd_otel:start_span(<<"probe">>),  
% Do work here ...  
%Example of with_span, the call to OpenTelemetry ?with_span is inside  
→ the adapter function, the function fun() -> ok end is executed  
→ inside dqsd_otel.  
dqsd_otel:with_span(<<"worker_2">>, fun() -> ok end),  
%End the outcome instance of probe. This ends the OpenTelemetry span  
→ aswell. If the outcome instance has already timed out, the  
→ oscilloscope receives no message where the status is successful.  
→ Otherwise, this sends a message with startTime, endTime, the name  
→ "probe" and success status.  
dqsd_otel:end_span(ProbeCtx, ProbePid),
```

3.3 Oscilloscope side

3.3.1 Server

The server is responsible for receiving the messages containing the outcome instances from the adapter. The server forwards the instances to the oscilloscope.

3.3.2 ΔQ Oscilloscope

The oscilloscope is a C++ graphical application which implements a dashboard to observe ΔQ s of probes inserted in the system under test [25]. It receives the instances corresponding to probes from the server and adds them to the time series of the probes whose instance is being received. The oscilloscope has a graphical interface which allows the user to create an outcome diagram of the system under test, display real time graphs which show detail about the execution of the system and allow the user to set custom timeouts for probes. It can also display snapshots of the system as if it was frozen in time

3.3.3 Inserting probes in the oscilloscope

Probes are automatically inserted in the oscilloscope when creating outcome diagrams. They are inserted on the outcomes observables, operators and to the sub-outcome diagrams (probes that observe the causal links of multiple outcomes/operators), we will see later on how they can be defined and how an outcome diagram can be created.

The names that are given to outcome, operators and sub-outcome diagrams are the names of the probes that observe them. Giving these probes a name allows the oscilloscope to match the outcome instances to the probes' time series.

In the system below, which is equal to the one defined above, probes are automatically attached to outcomes o_1, o_2 . The user who wants to observe the result of the sequential composition can insert probes at the start and end of the routine.

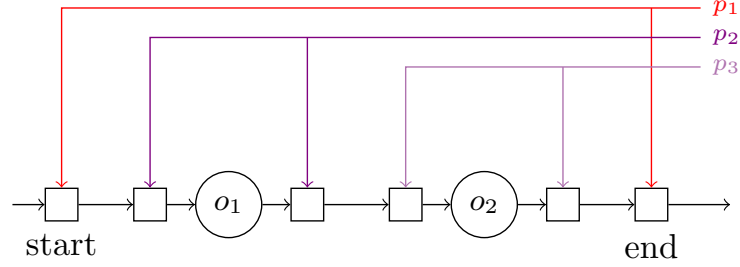


Figure 3.3: Probes inserted in the outcome diagram of the previous component diagram in Fig. 3.1.

As for operators, probes are automatically attached to the components inside them and to the start event and end events of the operators.

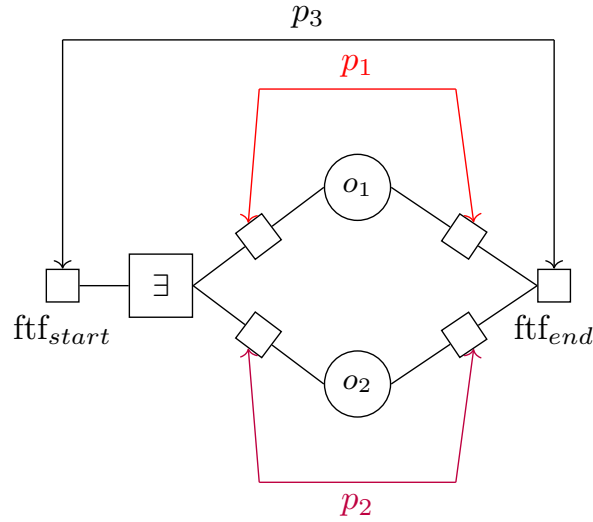


Figure 3.4: Probes inserted into an operator.

The **observed** ΔQ for the first-to-finish operator is the ΔQ for the observables (**start**, **end**). The **calculated** ΔQ is the ΔQ which is the result of the first-to-finish operator being applied on o_1, o_2

3.4 Triggers

Much like an oscilloscope that has a trigger mechanism to capture periodic signals or investigate a transient event [26], the ΔQ *oscilloscope* has a similar mechanism that can recognise when an observed ΔQ violates certain conditions regarding required behaviour and record snapshots of the system.

Each time an observed ΔQ is calculated, it is checked against the requirements set by the user. If these requirements are not met, a trigger is fired and a snapshot of the system is saved to be shown to the user.

3.4.1 Snapshot

A snapshot of the system gives insights into the system before and after a trigger was fired. It gives the user a still of the system, as if it was frozen in time. All the ΔQ s which are calculated during the system's execution are stored away. Then, if no trigger is fired, older ΔQ s are removed. Otherwise, the oscilloscope keeps recording ΔQ s without removing older ones, to allow the user to look at the state of the system before and after the trigger.

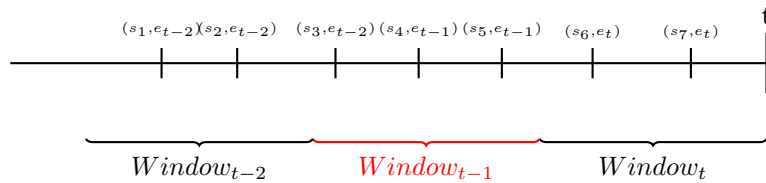
3.5 Sliding execution windows

There are two important windows that we consider in our oscilloscope, the *sampling window* and the *polling window*.

3.5.1 Sampling window

Suppose we are at time t , the observed (and calculated, if applicable) ΔQ s at time t we will display are the ΔQ s obtained from the outcome instances who ended within a sampling window in the **window of time** $(t-1)_l - (t-1)_u$, with $t-1$ equal to $t-x$, and x the sampling rate. The sampling rate is how often ΔQ s are calculated.

This is to account for various overheads that need to be taken into consideration. They could be network overhead, the adapter overhead, C++ latency ... Imagine multiple outcome instances that are ended at a time slightly lower but close to t , and due to the overheads the messages arrive at a time slightly higher but close to t , the outcome instance would not be taken into consideration for the calculation of a ΔQ .



The sampling window then advances every x seconds, setting the new window:

$$\begin{aligned} \text{From: } (t-1)_l, (t-1)_u &\xrightarrow{t+1} t_l, t_u. \\ \text{Where: } t_l &= (t-1)_u \text{ and } t_u = (t-1)_u + x \end{aligned}$$

3.5.2 Polling window

The polling window is the window of ΔQ s which are stored to keep a snapshot of the system over time and over which confidence bounds are calculated.

Suppose we are at time $t = 0$, the polling window will have 0 ΔQ s. As the sampling window advances, more ΔQ s are sampled, which in turn are added to the snapshot and to the confidence bounds.

The limit of ΔQ s for a polling window (subsequently snapshots and confidence bounds) is 30 ΔQ s. At $t = 31$, the older ΔQ s will be removed from the polling window and in turn from the snapshots and confidence bounds. Newer sampled ΔQ s will be added, keeping the limit of ΔQ s in a polling window to 30.

Chapter 4

Oscilloscope: User level concepts

The following chapter gives insights on the user level concepts of ΔQSD in the oscilloscope. They are the concepts needed by the user to understand how the oscilloscope works.

- We first provide insights into how ΔQSD was implemented in the oscilloscope, the parameters that define a probe's ΔQ , its representation and what can be done with ΔQ s. We show how probe's $\Delta Q(s)$ will be shown in the oscilloscope.
- We then provide a language to write outcome diagrams based on an already existing syntax.
- Lastly, we explain the different controls present on the oscilloscope dashboard.

4.1 ΔQSD concepts

Originally, $\Delta Q(x)$ denotes the probability that an outcome occurs in a time $t \leq x$, defining then the "intangible mass" of such IRV as $1 - \lim_{x \rightarrow \infty} \Delta Q(x)$. [6] We then extend the original definition to fit real time constraints, needing to calculate ΔQ s continuously.

For a given probe, $\Delta Q(t_l, t_u, dMax)$ is the probability that its instances with end time $t_l \leq t_e \leq t_u$ occur in time $t \leq dMax$.

4.1.1 Representation of a ΔQ

We provide a class to calculate the ΔQ of a probe between a lower time bound t_l and an upper time bound t_u . It can be calculated in two ways:

Observed ΔQ The first way is by having n collected outcome instances between t_l and t_u , calculating its *probability density function* (PDF) and then calculating the *empirical cumulative distribution function* (ECDF) based on its PDF. This is called the **Observed ΔQ** .

Calculated ΔQ A ΔQ can also be calculated by performing operations on two or more observed ΔQ s (convolution, operators operations), the notion of outcome instances is then lost between calculations, as the interest shifts towards calculating the resulting PDFs and ECDFs. This is called the **Calculated ΔQ** . A simple outcome can **not** have a "calculated ΔQ ", we can only observe the delay from its observables.

If you recall Section 3.3.3, the probes p_2 and p_3 observe simple outcomes, they can only display the observed ΔQ s of o_2, o_3 . The probe p_1 instead observes the sequential composition of said outcomes. We can display its "observed ΔQ " from the execution from *start* to *end* and the "calculated ΔQ " as the convolution of the observed ΔQ s of o_1, o_2

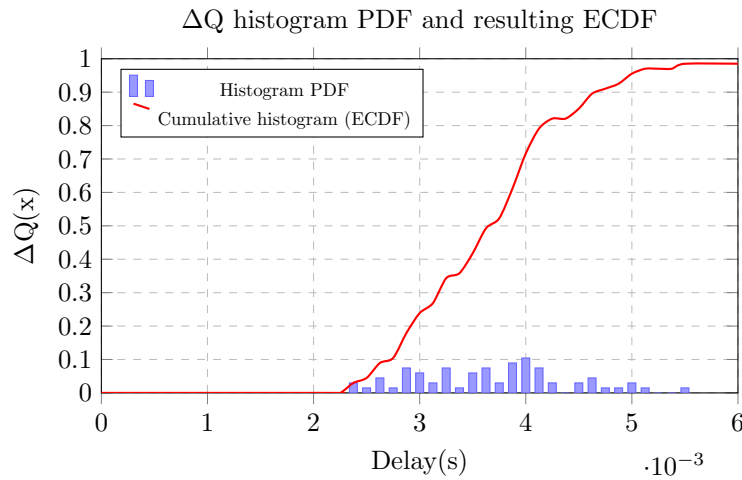


Figure 4.1: Blue bins: PDF of a sample ΔQ . Red: Resulting CDF of ΔQ PDF, the ECDF is displayed on the dashboard.

The PDFs of the observed and calculated ΔQ s($t_l, t_u, dMax$) are not shown on screen, only the ECDFs are plotted. Moreover, confidence bounds can then be calculated over multiple ΔQ s, we will see how in following sections.

4.1.2 $dMax = \Delta t \cdot N$

The key concept of ΔQSD is having a maximum delay after which we consider that the execution is timed out. This is represented in the oscilloscope as $dMax$. Understanding this equation is key to correctly using the oscilloscope and exploring tradeoffs

Setting a maximum delay for a probe is not a job that can be done one-off and blindly, it is something that is done with an underlying knowledge of the system inner-workings and must be thoroughly fine-tuned during the execution of the system by observing the resulting distributions of the obtained ΔQ s.

Let us explain the following equation:

$$dMax = \Delta t \cdot N \quad (4.1)$$

- $dMax$: The maximum delay, it represents the maximum delay that an outcome instance of a probe can have. The execution is considered "timed out" (failure) after $dMax$.
- Δt : The resolution of a ΔQ . It is the bin width of a bin in a probe's ΔQ .
- N : The precision of a ΔQ . It is the number of bins in a probe's ΔQ .

It can be informally described as a "two out of three" equation. If the user wants higher precision but the same $dMax$, the resolution must change, and so on for every parameter.

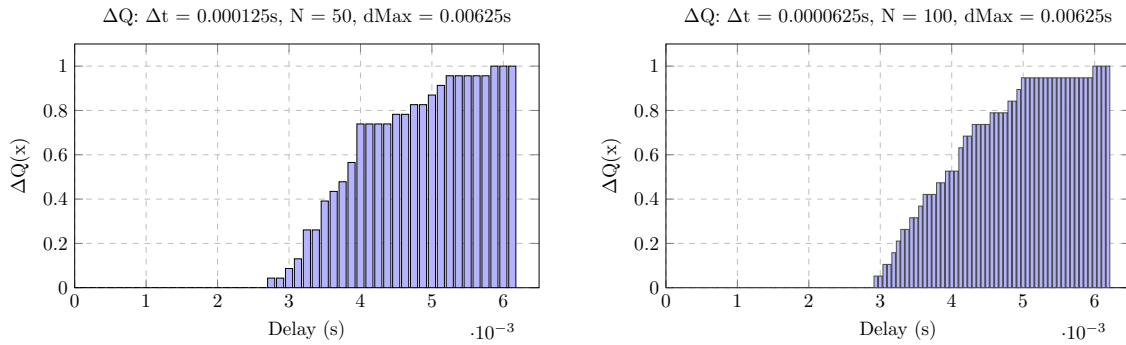


Figure 4.2: Left: Sample ΔQ with higher resolution but lower precision. Right: Sample ΔQ with lower resolution but higher precision. Both ΔQ s have the same $dMax$, but the amount of precise information they provide is far different.

Some tradeoffs must though be acknowledged when setting these parameters, a higher number of bins corresponds to a higher number of calculations and space complexity, a lower $dMax$ may correspond to more failures. The user must set these parameters carefully during execution y observing the shown plots.

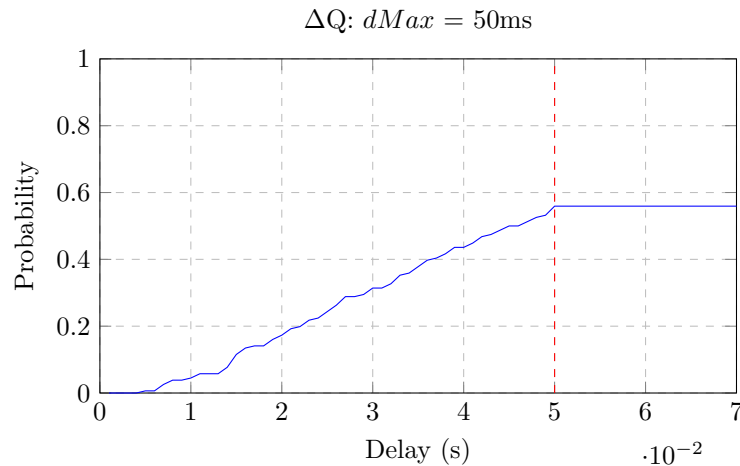


Figure 4.3: $\Delta Q: dMax = 50ms$, the ΔQ will stay constant when delay $> dMax$.

dMax limitation

dMax can **not** be lower than 1 millisecond and will be rounded to the **nearest** integer in the adapter, this is a limitation of Erlang `send_after` function which only accepts integers and milliseconds values. For example, if on the oscilloscope the *dMax* is equal to *1.56ms*, the adapter will fail spans after 2 ms.

4.1.3 QTA

A simplified QTA is defined for probes. We define 4 points for the step function at 25, 50, 75 percentiles and the maximum amount of failures accepted for an observable. An observed ΔQ will calculate that based on the samples collected.

4.1.4 Confidence bounds

To observe the stationarity of a probe we must observe its ΔQ s over a polling window and calculate confidence bounds over said ΔQ s. A single ΔQ may fluctuate, moreover, as it is an ECDF, it is not as precise as a CDF and, based on the number of bins, may not be precise. This is why we include the mean and confidence bounds of ΔQ s in the plot.

We first calculate a mean of the ΔQ s in the polling window, this gives an idea of how the probe has been behaving during the polling window.

Given this mean, we can calculate its confidence bounds, which give a probability range over which the true CDF of the ΔQ should fall. [27]

The bounds are updated dynamically by inserting or removing a ΔQ . Every time a new ΔQ is calculated, the oldest ΔQ in a window is removed if $\#\Delta Qs(\text{polling window}) > \text{limit}$. The new ΔQ is added to the calculation of the mean and confidence bounds as it is calculated.

This allows us to consider a small window of execution rather than observing the execution since the start for the bounds, this can help in observing stationarity of the system, where less sampled ΔQ s can help observe short term behaviour.

With a big window of ΔQ s, temporary overload may not greatly affect the mean and bounds, while, if we consider the current size of the polling window (30 ΔQ s), a few ΔQ s which deviate from stationary behaviour have a greater impact on the bounds and mean.

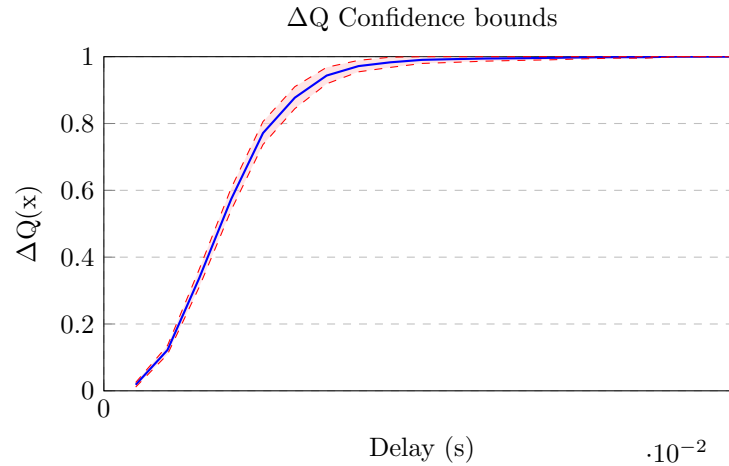


Figure 4.4: Upper and lower bounds (dashed, red) of the mean (blue) of multiple ΔQ s. In a system that behaves linearly, the bounds will be close to the mean, once the overload is approaching, or a system is showing behaviour that diverges from a linear one, the bounds will appear larger.

4.2 ΔQ display

Now that we have introduced the required concepts, we can put everything together to be plotted. In summary, a probe's displayed graph must contain the following functions:

- The observed ΔQ at time t , with a mean and confidence bounds of previous observed ΔQ s.
- If applicable, the calculated ΔQ from the causally linked components observed in a probe, with a mean and confidence bounds of previous calculated ΔQ s.
- Its QTA (if defined).

This allows for the user to observe if a ΔQ has deviated from normal execution, analyse its stationarity, nonlinearity and observe its execution.

In the photo below we can observe the multiple elements as they are displayed in real time in the oscilloscope.

- (1): The **Observed ΔQ** .
- (2): The **Calculated ΔQ** .
- (3): The **QTA**.
- Arrow, blue, below: The mean of the observed ΔQ s (yellow) with the confidence bounds of the mean. Upper bound (dark green) and lower bound (light green).
- Arrow, red, above: The mean of the calculated ΔQ s (ochre) with the confidence bounds of the mean. Upper bound (purple) and lower bound (magenta).

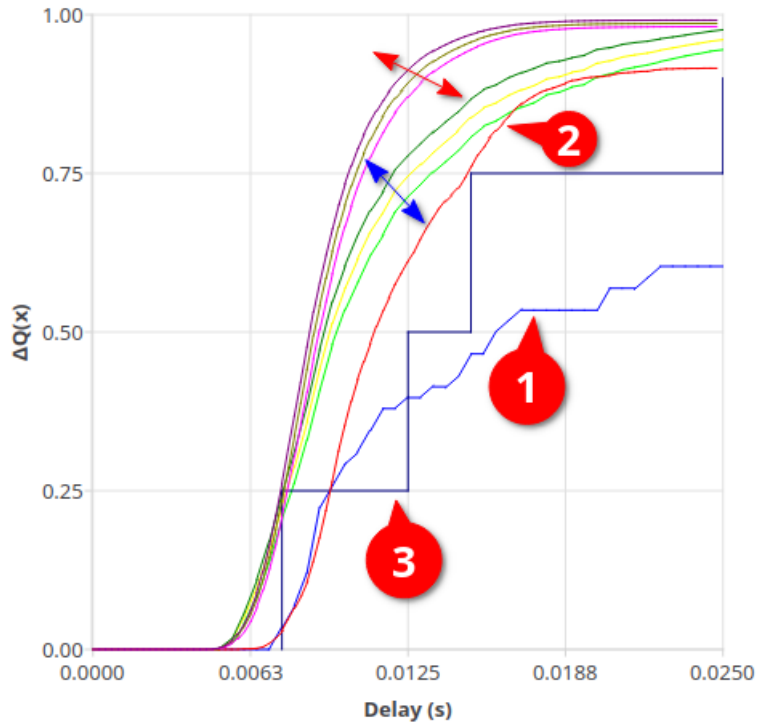


Figure 4.5: ΔQ s, confidence bounds, means and QTA for a probe observing the causal link of multiple components.

4.3 Outcome diagram

An abstract syntax for constructing outcome diagrams has already been defined in a previous paper [3], nevertheless, the oscilloscope needs a textual way to define an outcome diagram.

We define thus a grammar to create an outcome diagram in our oscilloscope, our grammar is a textual interpretation of the abstract syntax.

4.3.1 Causal link

A causal link between two components can be defined by a right arrow from `component_i` to `component_j`

```
component_i -> component_j
```

4.3.2 Sub-outcome diagrams

Multiple sub-outcome diagrams can be created for multiple parts of the system, these "sub-outcome diagrams" can then be linked together to form the global system outcome diagram. Recall Section 3.3.3, we defined a probe which observes the sequential composition of o_1, o_2 . The probe (sub-outcome diagram) p_1 can be defined as:

```
p_1 = o_1 -> o_2;
```

A probe is attached at the begin and end of p_1 , it will observe the whole system and the calculated ΔQ will be the convolution of o_1, o_2 .

The lines defining these diagrams must be semicolon terminated. Outcomes and operators cannot be defined on their own, they must be observed in a sub-outcome diagram.

Sub-outcome diagrams can be reused in other diagrams by adding **s:** (sub-outcome diagram) before they are used.

```
p_3 = s:p_1 -> s:p_2;
```

This allows for easy composition and reuse of different parts of the system, allowing for independent refining of diagrams.

4.3.3 Outcomes

To attach a probe to an outcome observables, it is enough to declare an outcome with its name inside a diagram.

```
... = outcomeName;
```

4.3.4 Operators

First-to-finish, all-to-finish and probabilistic choice operators must contain at least two components.

All-to-finish operator

An all-to-finish operator needs to be defined as follows:

```
a:name(component1, component2...)
```

First-to-finish operator

A first-to-finish operator needs to be defined as follows.

```
f:name(component1, component2...)
```

Probabilistic choice operator

A probabilistic choice operator needs to be defined as follows:

```
p:name[probability_1, probability_2, ... probability_i](component_1,
↪ component_2, ..., component_i)
```

In addition to being comma separated, the number of probabilities inside the brackets must match the number of components inside the parentheses. For n probabilities p_i , $0 < p_i < 1$, $\sum_{i=0}^n p_i = 1$

4.3.5 Limitations

Our system has a few limitations compared to the theoretical applications of ΔQ , namely, no cycles are allowed in the definition of outcome diagrams.

```
p_1 = s:p_2;  
p_2 = s:p_1;
```

The above example is not allowed and will raise an error when defined.

4.4 Dashboard

The dashboard is devised of multiple sections where the user can interact with the oscilloscope, create the system, observe the behaviour of its components, set triggers.

4.4.1 Sidebar

The sidebar has multiple tabs, we explain here the responsibility of each one.

System/Handle plots tab

System creation In this tab the user can create its system using the grammar defined before, he can save the text he used to define the system or load it, the system is saved to a file with any extension, we nevertheless define an extension to save the system to, the extension `.dq`. If the definition of the input is wrong, he will be warned with a pop-up giving the error the parser generator encountered in the creation of a system.

Adding a plot Once the system is defined, the user can choose the probes he wants to plot. They can select multiple probes per plot and display multiple plots on the oscilloscope window.

Sampling rate The user can choose the sampling rate of the system: How often ΔQ s are calculated and displayed in the oscilloscope.

Editing a plot By clicking onto a plot that is being shown, the user can choose to add or remove probes to and from it, thanks to the widget in the lower right corner. Multiple probes can be selected to either be removed or added.

Parameters tab

In this tab, the user can define parameters for the probes they have defined.

Set a QTA The user is given the choice to set a QTA for a given probe, they have 4 fields where they can fill in which correspond to the percentiles and the maximum amount of failures allowed, they can change this dynamically during execution.

dMax, bins The user can set the parameters we explained previously, Δt and N . When this information is saved by the user, the new $dMax$ is transmitted to the adapter and saved for the selected observable.

Triggers tab

In the triggers tab the user can set triggers and observe the snapshots of the system.

Set triggers The user can set which triggers to fire for the probes they desire, they are given checkboxes to decide which ones to set as active or not (by default, the triggers are deactivated).

Fired triggers Once a trigger is fired, the oscilloscope starts a timer, during which all probes start recording the observed ΔQ s (and the calculated ones if applicable) without discarding older ones. Once the timer expires, the snapshot is saved for the user in the triggers tab. In the dashboard, it indicates when the trigger was fired (timestamp) and the name of the probe which fired it.

Connection controls

Erlang controls The user can set the IP and the port where the ΔQ adapter is listening from. Two additional buttons communicate with the adapter by sending messages, they can start and stop the adapter's sending of outcome instances.

C++ server controls The user can set the IP and the port for the oscilloscope's server.

4.4.2 Plots window

To the left, the main window shows the plots of the probes being updated in real time.

4.5 Triggers

There are two available triggers which can be selected by the user, the triggers are evaluated on the **observed ΔQ** .

4.5.1 Load

A trigger on an observed ΔQ can be fired if the amount of outcome instances received for a probe in a sampling window is greater than what the user defines:

$$\#instances_{probe}(\Delta Q(t_l, t_u, dMax)) > \text{maxAllowedInstances}$$

4.5.2 QTA

A trigger on an observed ΔQ can be fired if:

$$\Delta Q_{obs} \not\prec \text{observableQTA}$$

Chapter 5

Oscilloscope: implementation

The following chapter gives a more technical description of the oscilloscope.

- We provide a more in-depth look at the ΔQ SD concepts introduced in the previous chapter.
- We then explain how the ΔQ adapter works, its API and the underlying mechanism that let us export outcome instances to the oscilloscope.
- Next we give a technical explanation of the parser generator we used to parse the outcome diagram syntax.
- Lastly, we briefly talk about the dashboard graphical framework.

5.1 ΔQ SD implementation

A probe's ΔQ can be represented internally by a PDF and displayed as an ECDF. Here is how both can be calculated given n outcome instances.

PDF

We approximate the PDF of the observed ΔQ via a histogram. We partition the values into N bins of equal width, Given $[x_i, x_{i+1}]$ the interval of a bin i , where $x_i = i\Delta t$, and $\hat{p}(x_i)$ the value of the PDF at bin i , for n bins:

$$\begin{cases} \hat{p}(i) = \frac{s_i}{n}, \text{ if } i \leq n \\ \hat{p}(i) = 0, \text{ if } i > n \end{cases} \quad (5.1)$$

Where s_i the number of successful outcome instances whose elapsed time is contained in the bin i , n the total number of instances. [28]

ECDF

The value $\hat{f}(x_i)$ of the ECDF at bin i with n bins can be calculated as:

$$\begin{cases} \hat{f}(i) = \sum_{j=1}^i \hat{p}(j), & \text{if } i \leq n \\ \hat{f}(i) = \hat{f}(x_n), & \text{if } i > n \end{cases} \quad (5.2)$$

5.1.1 dMax

We introduced $dMax$ in the previous section, we provide here the full equation that allows $dMax$ to be calculated:

$$dMax = \Delta_{tbase} * 2^n * N \quad (5.3)$$

Where:

- Δ_{tbase} represents the base width of a bin, equal to 1ms.
- N the number of bins.

5.1.2 Operations

In a previous section we talked about the possible operations that can be performed on and between Δ Qs, the time complexity of FTF, ATF and PC is trivially $\mathcal{O}(N)$ where N is the number of bins.

As to convolution, the naïve way of calculating convolution has a time complexity of $\mathcal{O}(N^2)$, this quickly becomes a problem as soon as the user wants to have a more fine-grained understanding of a component. Below we present two ways to perform convolution.

Convolution

Naïve convolution Given two Δ Q binned PDFs f and g , the result of the convolution $f \otimes g$ is given by [29]:

$$(f \otimes g)[n] = \sum_{m=0}^N f[m]g[n-m] \quad (5.4)$$

Fast Fourier Transform Convolution FFTW (Fastest Fourier Transform in the West) is a C subroutine library [30] for computing the discrete Fourier Transform in one or more dimensions, of arbitrary input size, and of both real and complex data. We use FFTW in our program to compute the convolution of Δ Qs. We adapt our script from an already existing one found on GitHub. [31]

Whilst the previous algorithm is far too slow to handle a high number of bins, convolution leveraging Fast Fourier Transform (FFT) allows us to reduce the amount of calculations to $\mathcal{O}(n \log n)$. This is why the naïve convolution algorithm is not used. We will analyse the time gains in a later chapter.

FFT and naïve convolution produce the same results in our program barring ε differences (around 10^{-18}) in bins whose result should be 0.

FFTs algorithms are plenty, the choice of the one to use is left up to the subroutine via the parameter `FFTW_ESTIMATE` [32].

Arithmetical operations

We can apply a set of arithmetical operations between Δ Qs ECDFs, and on a Δ Q.

Scaling (multiplication) A Δ Q can be scaled w.r.t. a constant $0 \leq j \leq 1$. It is equal to binwise multiplication on ECDF bins.

$$\hat{f}_r(i) = \hat{f}(i) \cdot j \quad (5.5)$$

Operations between Δ Qs Addition, subtraction and multiplication can be done between two Δ Q of equal bin width (but not forcibly of equal length) by calculating the operation between the two ECDFs of the Δ Qs:

$$\Delta Q_{AB}(i) = \hat{f}_A(i)[\cdot, +, -]\hat{f}_B(i) \quad (5.6)$$

5.1.3 Confidence bounds

Here is how we calculate the mean and lower/upper confidence for the Δ Qs ECDF at bin $i \quad \forall i < N$. [28]

For x_{ij} the value of an ECDF j at bin i , the mean of all ECDFs for the bin over a window is:

$$\mu_i = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij} \quad (5.7)$$

Its variance:

$$\sigma_i^2 = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij}^2 - \mu_i^2 \quad (5.8)$$

The confidence intervals CI_i for a bin i can then be calculated as:

$$CI_i = \mu_i \cdot \frac{\sigma_i}{\sqrt{n_i}} \quad (5.9)$$

5.1.4 Rebinning

Rebinning refers to the aggregation of multiple bins of a bin width i to another bin width j . Operations between Δ Qs can be done on Δ Qs that have the same bin width, this is why it is fundamental that all probes have a common Δ_{tbase} . This allows for fast rebinning to a common bin width.

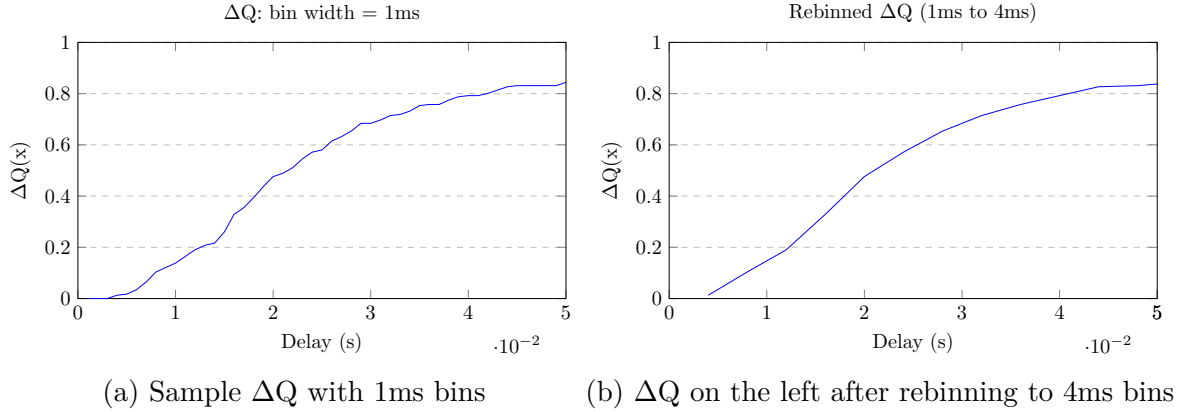
Given two Δ Qs $\Delta Q_i, \Delta Q_j$:

$$\Delta_{Tij} = \max \{ \Delta_{Ti}, \Delta_{Tj} \}$$

and the PDF of the rebinned ΔQ at bin b , from the original PDF of n bins, where $k = \frac{\Delta_{T_j}^i}{\Delta_{T_j}}$.

$$p'_b = \sum_{n=b \cdot k}^{b+1 \cdot k - 1} p_n, \quad b = 0, 1, \dots, \lceil \frac{N}{k} \rceil \quad (5.10)$$

We perform rebinning to a higher bin width for a simple reason, while this leads to loss of information for the ΔQ with the lowest bin width, rebinning to a lower bin width would imply inventing new values for the ΔQ with the highest bin width.



5.2 Adapter

The adapter, called `dqsd_otel` is a `rebar3` [33] application built to replace OpenTelemetry calls and create outcome instances, it is designed to be paired with the oscilloscope to observe an Erlang application.

5.2.1 API

The adapter functions to be used by the user are made to replace OpenTelemetry calls to macros as for `?start_span` and `?with_span` and `?end_span`. This is to make the adapter less of an encumbrance for the user.

Moreover, the adapter will always start OpenTelemetry spans but only start outcome instances if the adapter has been activated. The adapter can be activated by the oscilloscope by pressing the "start adapter" button and can be stopped via the "stop adapter" button.

start_span/1, start_span/2

```
start_span/1: -spec start_span(binary()) -> {opentelemetry:span_ctx(),
  ↳ pid() | ignore}.
start_span/2: -spec start_span(binary(), map()) ->
  ↳ {opentelemetry:span_ctx(), pid() | ignore}.
```


Parameters:

- Name: Binary name of the probe.
- Attributes: The OpenTelemetry span attributes (Only for `start_span/2`).

`start_span` incorporates OpenTelemetry `?start_span(Name)` macro.

Return: The function returns either:

- `{SpanCtx, span_process_PID}` if the adapter is active and the probe's *dMax* has been set.
- `{SpanCtx, ignore}` if one of the two previous conditions was not respected.

With `SpanCtx` being the context of the span created by OpenTelemetry.

with_span/1, with_span/2

```
with_span/1: -spec with_span(binary(), fun(() -> any())) -> any().  
with_span/2: -spec with_span(binary(), map(), fun(() -> any())) ->  
  ↳ any().
```

Parameters:

- Name: Binary name of the probe.
- Fun: Zero-arity function representing the code of block that should run inside the `?with_span` macro.
- Attributes: The OpenTelemetry span attributes (Only for `with_span/3`).

`with_span` incorporates OpenTelemetry `with_span` macro.

Return: `with_span` returns what `Fun` returns (`any()`).

end_span

```
-spec end_span(opentelemetry:span_ctx(), pid() | ignore) -> ok |  
  ↳ term().
```

Parameters:

- `SpanCtx`: The context of the span returned by `start_span`.
- `Pid`: `span_process_PID` || `ignore`.

As is the case for `start_span`, `end_span` incorporates an OpenTelemetry macro, in this case `?end_span(Ctx)`.

fail_span

```
-spec fail_span( pid() | ignore) -> ok | term().
```

Parameter:

- Pid: `ignore` || `span_process_PID`.

`fail_span` does not incorporate any `OpenTelemetry` macro, it is let up to the user to decide how to handle failures in execution.

span_process

`span_process` is the process, spawned by `start_span`, responsible for handling the `end_span`, `fail_span`, `timeout` messages.

Upon being spawned, the process starts a timer with time equal to the `dMax` set by an user for the probe being observed, thanks to `erlang:send_after`. When the timer runs out, it sends a `timeout` message to the process.

The process can receive three kinds of messages:

- `{end_span, end_time}`: This will send an outcome instance to the oscilloscope with the start and end time of the execution of the probe and success status.
- `{fail_span, end_time}`: This will send an outcome status to the oscilloscope indicating that an execution of a probe has failed.
- `{timeout, end_time(StartTime + dMax)}`: If the program hasn't ended the span before `dMax`, the timer will send a `timeout` message and it will send an outcome instance to the oscilloscope indicating that an execution of a probe has timed out.

The process is able to receive one and only message, if the execution times out and subsequently the span is ended, the oscilloscope will not be notified as the process is defunct. This is assured by Erlang documentation:

If the message signal was sent using a process alias that is no longer active, the message signal will be dropped. [34]

5.2.2 Handling outcome instances

To create outcome instances of a probe we must obtain three important informations:

- Its name.
- The time when the span was started.
- Its `dMax`.

They start time and end time are supplied by calling this function:

```
StartTime/EndTime = erlang:system_time(nanosecond).
```

The name is given when starting a span and the *dMax* is stored in a dictionary in the adapter.

The outcome instance is created only if two conditions are met: the adapter has been set as active and the user set a timeout for the probe, the functions will spawn a `span_process` process, passing along all the necessary informations.

Once the span is subsequently ended/timed out/failed, the function `send_span` creates a message carrying all the informations and sends it to the C++ server. The formatting of the messages is the following:

```
n:Observed name, b: Start time (beginning), e: End time (end or  
↪ deadline), s: The status
```

5.2.3 TCP connection

The adapter is composed of two `gen_server` which handle communication to and from the oscilloscope. This `gen_server` behaviour allows the adapter to send spans asynchronously to the oscilloscope.

TCP server

The TCP server is responsible for receiving commands from the oscilloscope. It can be run by setting its IP and port via:

```
-spec start_server(string() | binary() | tuple(), integer()) -> ok |  
↪ {error, Reason}
```

The oscilloscope can send commands to the adapter, these commands are:

- `start_stub`: This command sets the adapter as active, it can now send outcome instances to the oscilloscope if the probe's *dMaxs* are defined.
- `stop_stub`: This commands sets the adapter as inactive, it will no longer send outcome instances to the oscilloscope.
- `set_timeout;probeName;timeout`: This command indicates to the adapter to set the *dMax* = timeout for a probe, a limit of the adapter is that erlang:send_after does not accept floats as timeouts, so the timeout will be rounded to the nearest integer.

TCP client

The TCP client allows the adapter to send the spans to the oscilloscope. The client connects over TCP to the oscilloscope by connecting to the oscilloscope server's address and opens a socket where it can send the outcome instances.

```
-spec try_connect(string() | binary(), integer()) -> ok.
```

5.3 Parser

To parse the system, we use the C++ ANTLR4 (ANother Tool for Language Recognition) library.

5.3.1 ANTLR

ANTLR is a parser generator for reading, processing, executing or translating structured text files. ANTLR generates a parser that can build and walk parse trees [35].

ANTLR is just one of the many parsers generators available in C++ (flex/bison, lex, yacc). Although it presents certain limitations, its generated code is simpler to handle and less convoluted with respect to the other possibilities.

ANTLR uses Adaptive LL(*) (*ALL*(*)) parser, namely, it will move grammar analysis to parse-time, without the use of static grammar analysis. [36]

5.3.2 Grammar

ANTLR provides a yacc-like metalanguage [36] to write grammars. Below, is the grammar for our system:

```
grammar DQGrammar;

PROBE_ID: 's';
BEHAVIOR_TYPE: 'f' | 'a' | 'p';
NUMBER: [0-9]+('.'[0-9]+)?;
IDENTIFIER: [a-zA-Z_] [a-zA-Z0-9_]*;
WS: [ \t\r\n]+ -> skip;

start: definition* system? EOF;

definition: IDENTIFIER '=' component_chain ' ';

component_chain : component ('->' component)*;

component : behaviorComponent | probeComponent | outcome;

behaviorComponent : BEHAVIOR_TYPE ':' IDENTIFIER ('[' probability_list
    ↪ ']' )? '(' component_list ')';

probeComponent : PROBE_ID ':' IDENTIFIER;

probability_list: NUMBER (',' NUMBER)+;
component_list: component_chain (',' component_chain)+;
outcome: IDENTIFIER;
```

Limitations

A previous version was implemented in Lark [37], a python parsing toolkit. The python version was quickly discarded due to a more complicated integration between Python and C++. Lark provided Earley(SPPF) strategy which allowed for ambiguities to be resolved, which is not possible in ANTLR.

For example the following system definition presents a few errors:

```
probe = s -> a -> f -> p;
```

While Lark could correctly guess that everything inside was an outcome, ANTLR expects ":" after "s, a, f" and "p", thus, one can not name an outcome by these characters, as the parser generator thinks that an operator or a probe will be next.

5.4 Oscilloscope GUI

Our oscilloscope graphical interface has been built using the QT framework for C++. Qt is a cross-platform application development framework for creating graphical user interfaces. [38] We chose Qt as we believe that it is the most documented and practical library for GUI development in C++, using Qt allows us to create usable interfaces quickly, while being able to easily pair the backend code of C++ to the frontend.

The interface is composed of a main window, where widgets can be attached to it easily. Everything that can be seen is customisable widgets. This allows for easy reusability, modification and removal without great refactoring due in other parts of the system.

Chapter 6

Application on synthetic programs

This section aims to provide an example of how the oscilloscope could be used to instrument an application, in this case, a synthetic one. We explain how the Δ QSD paradigm can be applied to explore tradeoffs in design and to gain more insights into a running system.

6.1 System with sequential composition

We model a first system with two sequentially composed component. We choose two model the two components as M/M/1/K queues.

Why M/M/1/K queues? An average component in a distributed system can be modeled as an M/M/1/K, due to the exponential inter-arrival rate of messages λ , the exponential distribution of the execution delay μ , the buffer size of messages K of a component and the failure rate f . [1]

Let us first provide a refresher about M/M/1/K queues:

- λ : The arrival rate.
- s : The service time, is the time it takes to serve a message.
- μ : The service **rate** and $E[s] = \frac{1}{\mu}$
- Offered load: $\rho = \frac{\lambda}{\mu}$

We will control λ to show its effects on the offered load. The offered load can tell much about the system:

- At low load ($\rho < 0.8$) the failure will tend to 0, the system is behaving correctly and the Δ Q will show that, the delay will tend to 1.
- Once ρ is approaching high load ($\rho > 0.8$) we can observe the failure increasing quickly, but we can observe the system starting to get bad after $\rho > 0.5$! [1]

6.1.1 System composition

The system has two components **worker_1**, **worker_2**. Each individual component is made of a buffer queue of size $K = 1000$ and a worker process.

The system sends n messages per second following a Poisson distribution to **worker_1**'s queue, the queue then reduces its available buffer size.

The buffer notifies its worker, which then does N loops, which are defined upon start, of fictional work. The worker then passes a message to **worker_2**'s queue, which has another queue of same size, who passes the message to **worker_2**'s worker, which does the same amount of loops. When a worker completes its work, it notifies the queue, freeing one "message" from its buffer size and allowing the next message to be executed.

If the queue's buffer is overloaded, it will drop the incoming message and consider the execution a failure.

A probe "probe" is defined, which observes the execution from when the message is sent to **worker_1** up until **worker_2** is done.

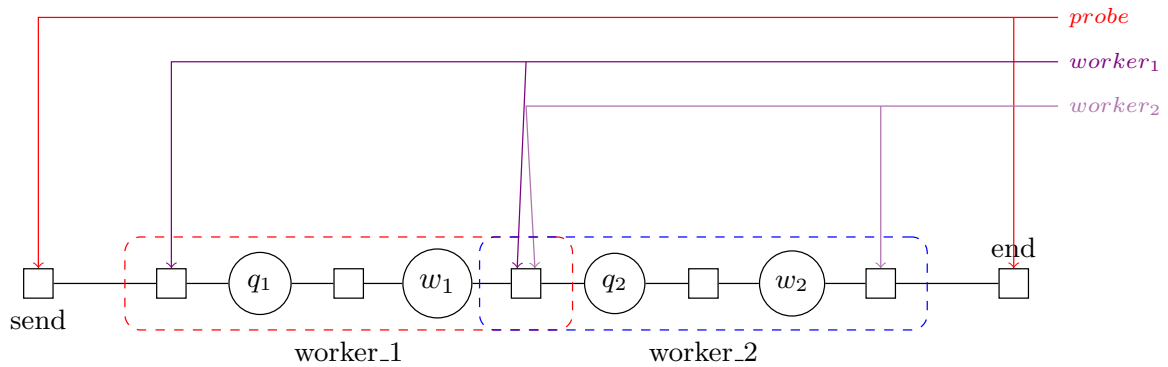


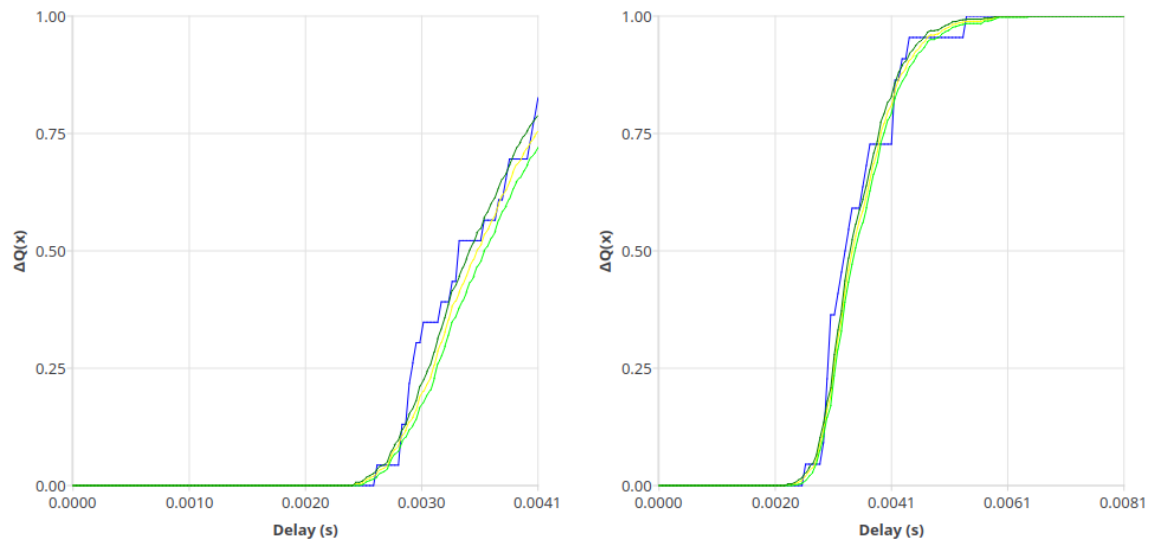
Figure 6.1: Outcome diagram of the M/M/1/K queue with the colored lines representing the probes that were inserted.

6.1.2 Determining parameters dynamically

We stated previously that determining parameters is something that must be done with an underlying knowledge of the system. The oscilloscope can provide knowledge of the system, here is an example of **worker_1** and **worker_2** as observed in the oscilloscope.

Imagine the engineer supposes the workers executions should a maximum of 4 ms to complete, but doesn't actually know how long the executions should take. The engineer, after having set the required parameters observes in the following graph in the oscilloscope Section 6.1.2.

The oscilloscope shows the engineer that their assumptions do not correspond to the actual system ΔQ , the user can then modify the parameters to observe the actual system's behaviour. By setting $dMax$ to 8 ms, they can observe the worker's ΔQ s failure approaching 0.



(a) worker_1 ΔQ with confidence bounds plot with 4 ms $dMax$. (b) worker_1 ΔQ with confidence bounds plot with 8 ms $dMax$.

On the other hand, the engineer's assumption could have been what he truly expected from the system, in this case, the oscilloscope tells him that the system is not behaving as expected.

Low Load

Let's first observe the system at low load, we will send 50 messages per second to observe the system under test to get key properties.

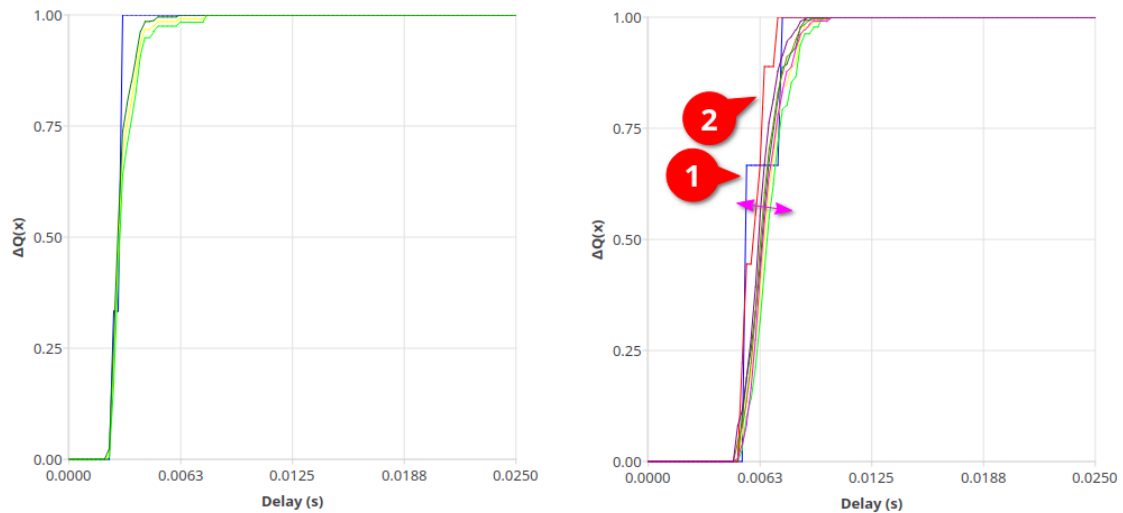


Figure 6.3: Left: worker_1 ΔQ , in blue, the observed ΔQ , in green, the confidence bounds. Right: probe ΔQ , in blue, the observed ΔQ , in red the calculated ΔQ . In magenta, the two ΔQ s confidence bounds overlapping.

We first observe the worker's ΔQ , we can observe that the average execution takes $\approx 30\text{ms}$. We then have $\mu_{worker} = \frac{1}{0.0033} \approx 300 \text{ req/s}$. Thus $\rho = \frac{50}{322} = 0.16$, we are in nice grounds!

At low load, we can observe in the oscilloscope the probe **observed ΔQ** and **calculated ΔQ** confidence bounds overlap.

Early signs of overload

We can see how even at load = 0.5 the system is starting to show bad behaviour. Let us observe what happens when $\lambda = 150 \rightarrow \rho = 0.5$.

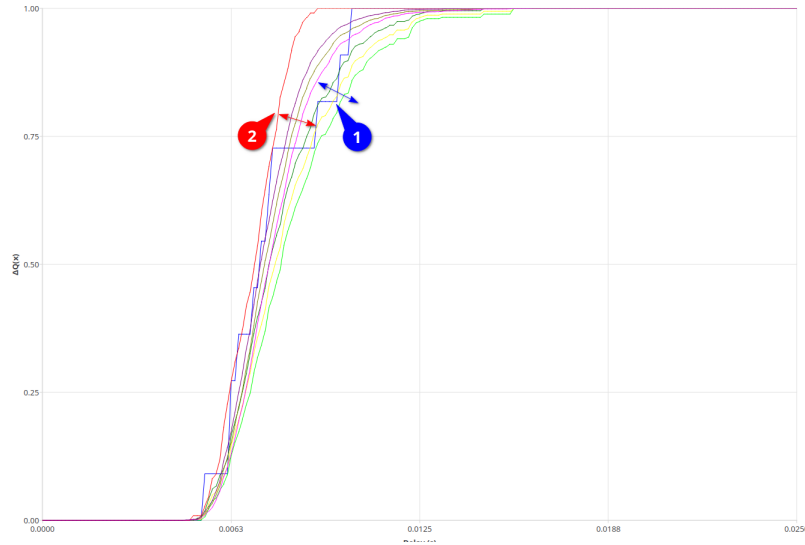
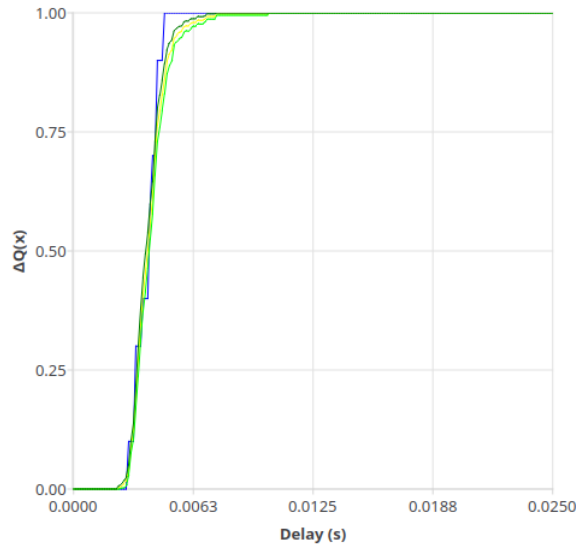


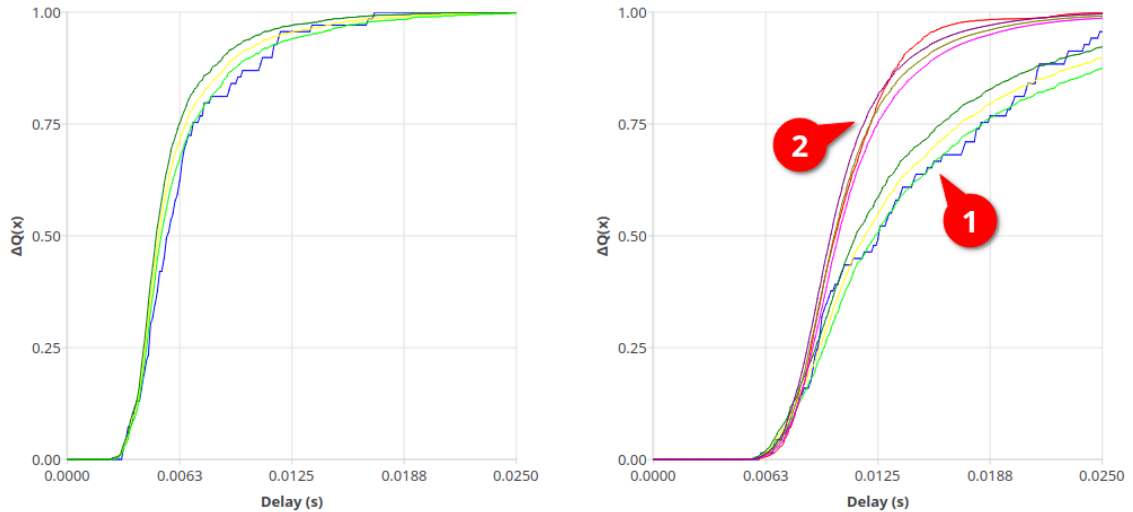
Figure 6.4: probe ΔQ s, in blue (1), the observed ΔQ , in red (2) the calculated ΔQ . Arrow, above, blue: Observed ΔQ confidence bounds. Arrow, below, red: Calculated ΔQ confidence bounds.

Figure 6.5: worker_1 ΔQ (blue) and its confidence bounds

Recall 2.11, we can start to observe early signs of dependency! At load 0.5 the calculated ΔQ is deviating from the observed one. This is a sign that the performance is degrading. Worker_1 is slowing down, but we nevertheless do not observe failures in probe's ΔQ s.

High load

Performance at 0.5 offered load are already remarkable, the ΔQ s are shifting. We can go even further and observe the system under high load situations. We set $\lambda = 200 \rightarrow \rho = 0.83$, just above the high load threshold.

Figure 6.6: Left: worker_1 ΔQ . Right: probe ΔQ , in blue (1), the observed ΔQ with its confidence bounds, in red (2) the calculated ΔQ with its confidence bounds.

This is what we expected previously, and confirms what is expected by queueing theory, ΔQ is capable of observing the basic observation requirements and capable of recognising dependency. While what is expected by the execution of the queue (observed ΔQ) is a nice normally distributed CDF with little to no failure. What we can actually observe is a degraded performance in both workers and the probe observed execution.

The workers CDF has completely degraded, with the average request taking almost double the time as under normal queueing conditions.

Further degradation can be observed by increasing $\lambda = 300, 350 \rightarrow \rho \geq 1$.

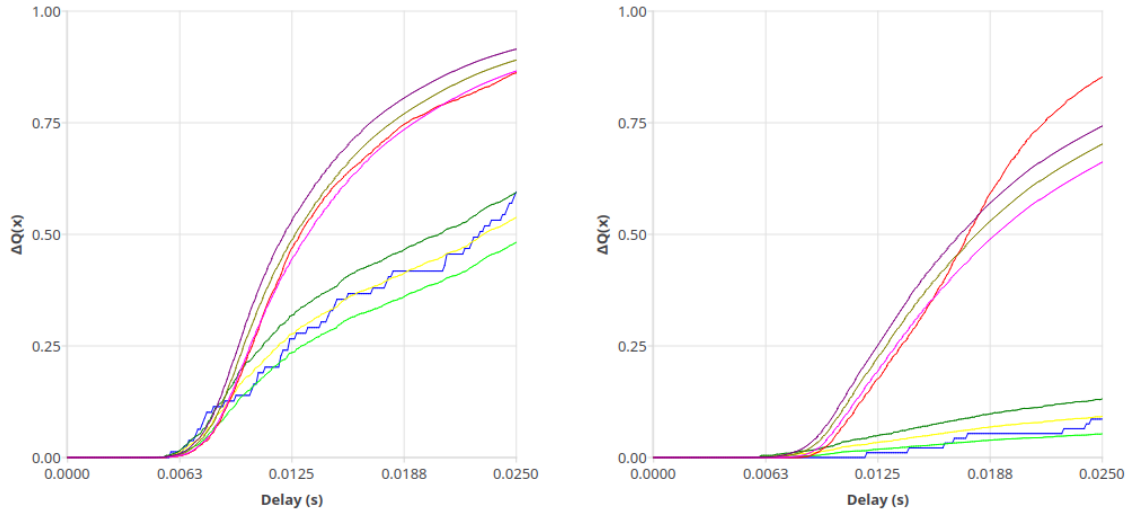


Figure 6.7: Left: probe ΔQ s at $\lambda = 300$. Right: probe ΔQ s at $\lambda = 350$

The system degrading clear, the ΔQ s show how almost all messages are being dropped or take $> dMax$. Let us look at triggers and how they can be useful to diagnose such cases.

Triggers

By observing the system under test in high load cases, we can set the load trigger by setting the sampling window to 1 second and trigger when outcome instances $\gtrapprox 150$. We can also set a trigger based on observation of the running system.

QTA trigger By observing the system, we create a QTA for the probe with: 25% = 0.0075 s, 50% = 0.0125 s, 75% = 0.015s and minimum intangible mass = 0.9.

By setting the trigger to fire for $\Delta Q_{obs} < QTA$. We captured a handful of snapshots. Here, $\lambda = 150$.

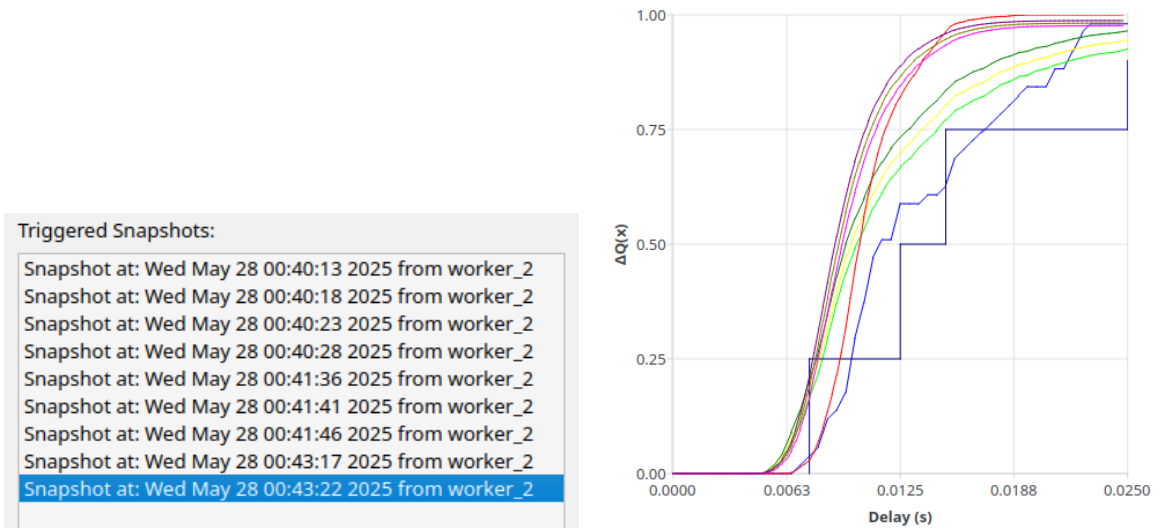
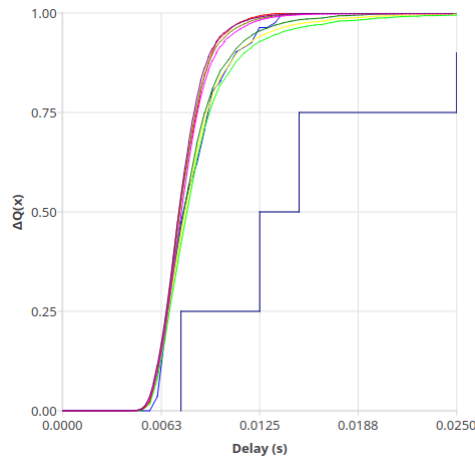


Figure 6.8: Left: Fired triggers. Right: Sample QTA violation.

QTA triggers can help to detect overhead even before high load becomes evident!

Instances trigger By knowing the inner details of the system, setting a QTA on the number of instances can be useful. Here is an example of a fired trigger on the number of instances.

Even though the QTA requirement isn't being violated, the number of instances fires a trigger, where the user can observe that the system is showing early signs of overload.



6.2 Detecting slower workers in workers

6.2.1 First to finish application

Next, we provide a synthetic application modeling an application that can be modeled by a first to finish operator

Why first to finish? Recall the previous FTF graph Fig. 2.5. Assume a send request to "the cloud" that waits for a response or a timeout, it is modeled by a FTF operator.

Using the wrong operator

What happens if the wrong operator is chosen to represent the causal relationships between the outcomes? What if the user believes that the system diagram is the one we presented before Fig. 6.1? The result on the oscilloscope will clearly show that something is wrong!

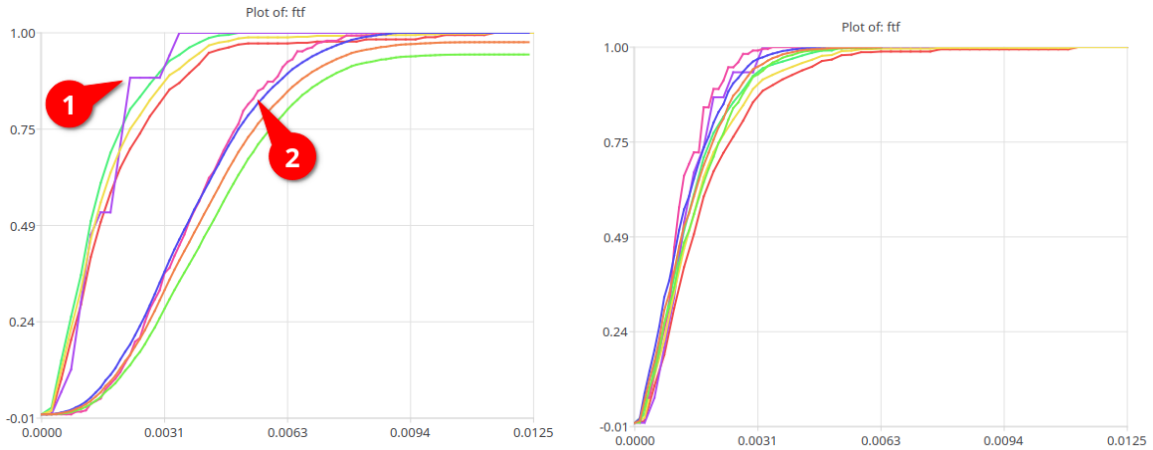


Figure 6.9: (Left) FTF plot **with wrong outcome diagram definition** as shown in the oscilloscope. (1) Observed ΔQ . (2) Calculated ΔQ . (Right) FTF plot **with correct outcome diagram definition** as shown in the oscilloscope. Observed ΔQ and calculated ΔQ overlapping.

On the left, we can observe how the **calculated ΔQ** (2) is clearly greater than the **observed ΔQ** (1). A difference this drastic tells us that the proposed outcome diagram does not correctly represent the actual system. On the right, if no dependencies are present and the correct operator is chosen, the two graphs will overlap.

Introducing a slower component

Let us introduce a slower worker into the system, we introduce an artificial delay into worker_2 (about 20ms). If the oscilloscope works correctly, the paradigm operations are sound and no dependencies are present in the system, we should not see any difference in the observed and calculated ΔQ s of the FTF operator.

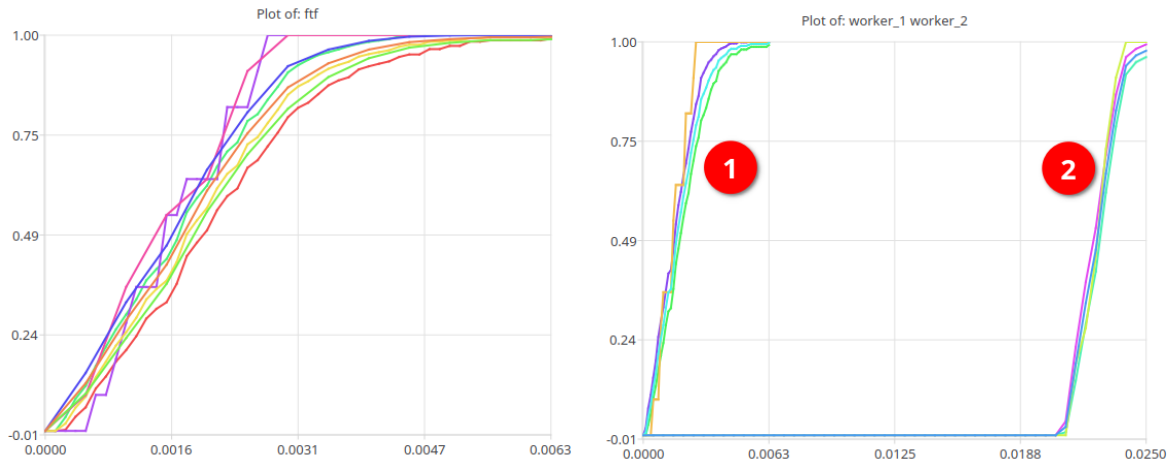


Figure 6.10: (Left) FTF plot of worker_1 and worker_2, observed and calculated ΔQ overlapping.

(Right) worker_1 (1) and worker_2 (2) ΔQ s.

The FTF plot correctly displays how worker_2 does not have an effect on the ftf plot.

6.2.2 All to finish application

We can extend the previous application to an all-to-finish operator, this operator can for instance parallel work, a task that requires a lot of computation and can be done in separate pieces by separate workers. [1]

Introducing a slower component

Like we did for the FTF operator, let's introduce a slower work into the mix. We introduce a slight delay to show how even a few milliseconds can be noticeable right away by a keen eye (or by triggers, which avoids having to look constantly at the graphs). The delay is a 2ms sleep on worker_2.

Worker's performance We present here the worker's graphs as shown on the oscilloscope.

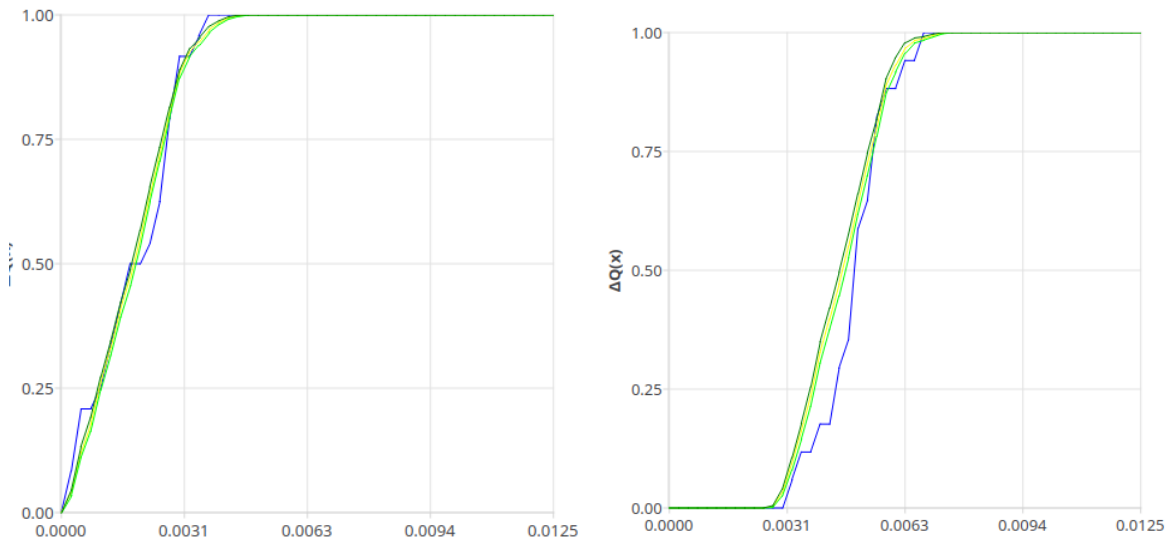
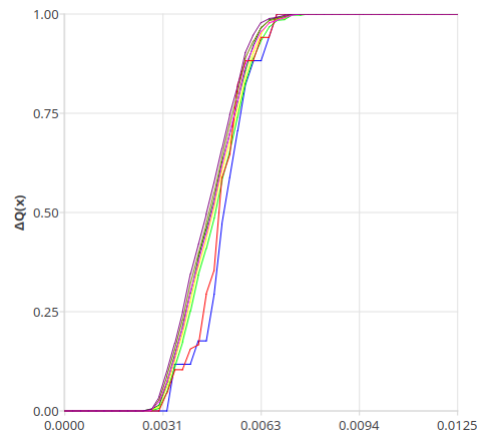


Figure 6.11: Left: worker_1 plot. Right: worker_2 plot.

The difference in the worker's ΔQ can be noticed with $\Delta Q_w 2 > \Delta Q_w 1$. The difference can then be observed in the all-to-finish plot, where the operator's ΔQ s (both observed and calculated) can be overlaid on top of worker_2 ΔQ , showing once again that the ΔQSD foundation is sound.



These plots show the usefulness of ΔQSD , the system can be decomposed to understand which part of the system is showing hazards, furthermore, the causal relationships can be observed to determine the behaviour of a part down to the single component.

Chapter 7

Performance study

This chapter evaluates the components and operations we introduced in previous sections, analysing their performances.

- We first evaluate the convolutions algorithms we previously introduced. We stated that naïve convolution would have $\mathcal{O}(n^2)$ time complexity, while FFT convolution would have $\mathcal{O}(n \log n)$ complexity. We will evaluate to see if what we observe corresponds to theory.
- We then evaluate the ΔQ adapter API performances, to see the overhead it introduces into a system.
- Lastly, we want to evaluate the QT framework plotting performances, we believe it is the weakest link in the oscilloscope and thus want to evaluate the QtCharts class when plotting ΔQ s.

7.1 Convolution performance

We implemented two versions of the convolution algorithm as described before, the naïve version and the FFT version. We compared their performance when performing convolution on two ΔQ s of equal bins. In theory, we should observe the naïve version delay quickly grow, while the FFT version have a log-linear growth.

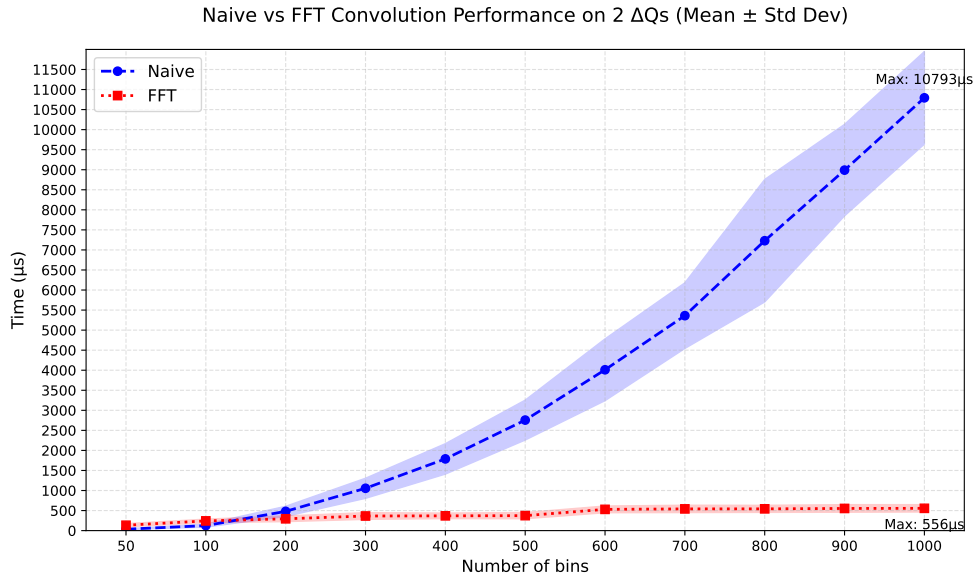


Figure 7.1: Performance comparison of two convolution algorithms

As expected, the naïve version has a time complexity of $\mathcal{O}(n^2)$ and quickly scales with the number of bins, this is clearly inefficient, as a more precise ΔQ will result in a much slower program.

As for the FFT algorithm, it is slightly slower when the number of bins is lower than 100. This is due to the FFTW3 routine having slightly higher overhead. Moreover, if we look closer at the FFT graph, we can observe slight increases after we surpass powers of 2 (e.g at $600 > 512$, $300 > 256 \dots$). This is because the algorithm is based on ΔQ s which are zero-padded to the nearest power of 2, this is to make the calculation more accurate.

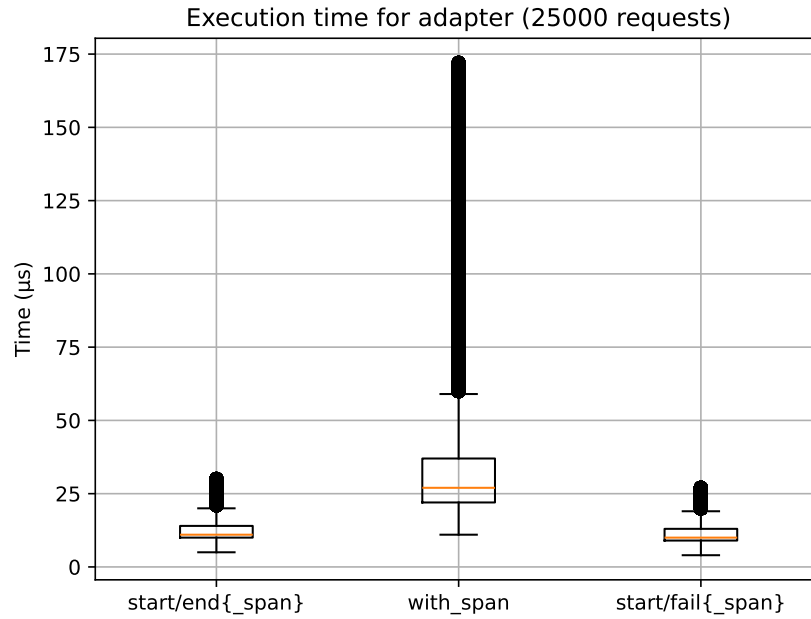
While we limit the number of bins to 1000 right now, this limit could potentially be increased as the convolution algorithm is very efficient (0.5 ms for 1024 bins).

7.2 ΔQ adapter performance

We evaluated the performance of the adapter to measure its impact in a normal execution, namely we tested the following calls which represent a normal usage of the adapter.

- `start_span` \rightarrow `end_span`.
- `with_span` with the following function: `fun()` \rightarrow `ok`.
- `start_span` \rightarrow `fail_span`.

We ran the simulation for 25000 subsequent iterations, these are the results.

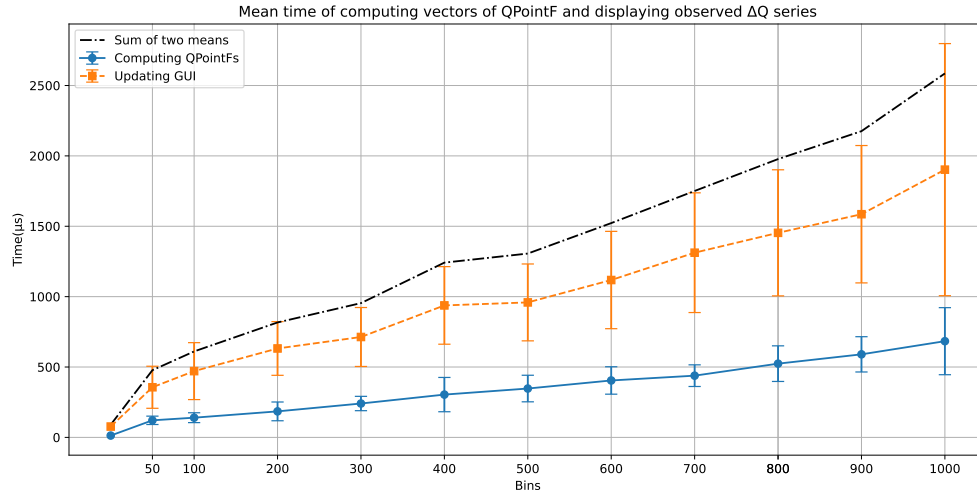


The overhead is minimal, around 10 microseconds on average to start and end/fail a span. The same cannot be said about with span, the increased overhead is nevertheless due to a function needing to be called inside it for it to record a span.

7.3 GUI plotting performance

We evaluated the performance of the GUI plotting routine for an observed ΔQ . The routine first prepares the ΔQ , creating vectors `QPointF` (a Qt class representing a point for a `QtChart`), representing the x and y values of the ΔQ s CDF. The vectors are created for the lower bound, the upper bound the mean of the window of ΔQ s and the observed ΔQ .

Then, once the vectors are prepared, Qt replaces the old points with the new points for every element being plotted.



The result scales up to 2 ms for 1000 bins, since multiple plots have to be shown, if the probe observes the causal relationships of outcome and operators, the calculated ΔQ will need to be plotted too, the time increase will be twofold. If a user decides to have finer grained representations and multiple plots at once, decreasing (decreasing or increasing?) the polling rate will avoid the plots frame skipping.

Chapter 8

Conclusions and future work

As we introduced the thesis, its background and current problems we think exist in observability tools, we set out a clear goal: design a graphical dashboard, the **ΔQ oscilloscope**, to observe running Erlang applications and plot the system behaviour in real time thanks to the ΔQSD paradigm to diagnose performance issues and to optimise performance over lifetime.

While we can not say that we are finished with the development of the oscilloscope, we can clearly say that a **working prototype** that reflects the theoretical findings of the paradigm and fulfills the initial goals was created.

This was successfully achieved thanks to multiple important implementations which make it fast, reliable and moreover capable of accurately detecting deviations from required behaviour.

The ΔQ adapter, named `dqsd_otel`, an Erlang application which is able to work alongside OpenTelemetry to add the notion of failure according to ΔQSD to spans. The adapter can communicate data about outcome instances from a running system directly to the oscilloscope and can directly receive commands from the oscilloscope. The `gen_server` behaviour allows for this to be done fast and asynchronously.

The ΔQ oscilloscope, a fully fledged Qt dashboard application that is able to observe running systems and provide real time plotting capabilities to the user. Moreover, it provides full control over to the user of the outcome diagrams, the parameters of probes, their QTAs, the triggers they want to set for a given probe, a system of snapshots which allows observing the system as if it was frozen in time. In fine, the FFT convolution algorithm allows us to scale down from $\mathcal{O}(n^2)$ complexity to $\mathcal{O}(n \log n)$, bringing the time to provide precise insights significantly down.

The synthetic applications further prove the oscilloscope's usefulness in detecting early signs of overload and dependent behaviour. This reinforces the solid theoretical basis of ΔQSD , which we remind has already been applied in many industrial projects.

Many crucial features are still missing from the dashboard, and it could require less code modifications in the Erlang side. The next important step of the oscilloscope is its trial in a true distributed application. Thise

8.1 Future improvements

We believe the oscilloscope and the Erlang application can be drastically improved, the size of the project and its intended goal is too big to be encompassed in a single master thesis. We list here some improvements which could be made to both the oscilloscope and the adapter.

8.1.1 Oscilloscope improvements

- The oscilloscope could be turned into a **web app**, we feel that a C++ oscilloscope is a good prototype and proof of concept, but its usability would be greater in a browser context. It would be great as a plugin for already existing observability platforms like Grafana.
- A wider selection of **triggers**, as of writing this thesis, only the QTA trigger and load are available, this is a limitation due to time constraints. Nevertheless, triggers can be easily implemented in the available codebase.
- **Better communication between stub - server - oscilloscope.** The current way of sending outcome instances may be a limiting factor under high load, if hundred of thousands of spans were to be sent, the current way the server and oscilloscope are tied together may throttle communications. TCP socket connections could quickly become the chokepoint which makes the oscilloscope temporarily unusable.

Future improvements on the server side could implement epoll system server calls to make the server more efficient; **Detaching server from client**, as of right now, the oscilloscope and the server are tied together, using ZeroMQ to assure real time server-client communications could be an interesting solution to explore.

- **Improve real time graphs.** The class QtCharts does not perform correctly with high frequencies update. Moreover, since we are plotting multiple series (from a minimum 4 to a maximum of 9) per probe, which allows up to 1000 bins per probe, the performance quickly degrades with more probes being displayed. A better graphing class for Qt could definitely improve the experience.
- **Saving probe parameters:** As of writing this thesis, there is no way to save the parameters one may have set.
- **Deconvolution:** An important aspect of Δ QSD, which was not introduced in this paper is deconvolution. It is used to check for infeasability in system desing. Since convolution has already been implemented, this could be integrated using the FFTW3 library.
- **Exporting graphs:** The graphs can only be observed in the oscilloscope and have no way to be exported to other programs via standard formats.
- **Many more:** This oscilloscope is just a start, if we were to list everything we may want to add, it would take many pages. What we provide is a sufficient enough basis to provide possibilities to observe a running system and understand

the power of Δ QSD in analysing its behaviour.

8.1.2 Adapter improvements

- As suggested by Bryan Naegele, a member of the observability group of Erlang, the adapter, instead of working on top of OpenTelemetry, could be directly included inside the context of a span by using the `ctx` library [39], which provides deadlines for contexts, propagating the value in `otel_ctx`, making it available to the OpenTelemetry span processor. Leveraging `erlang:send_after` as we already do, we could create outcome instances with telemetry events to handle successful executions and timeouts. The span processor will then be responsible for creating outcome instances, without creating the need for custom functions in the adapter, like we have now.

8.1.3 Real applications

A flaw of the oscilloscope and adapter is that they have not been tested on real applications, while their usefulness has been proven on synthetic applications, the lack of real life applications is a weakness.

8.1.4 Licensing limitations

Lastly, a notable limitation is created by **Qt**, namely, QtCharts. The usage of Qt does not allow us to release our project under BSD/MIT licenses, but rather a GPLv3 one (we cannot release it under LGPL due to QtCharts). [40]

Bibliography

- [1] Peter Van Roy and Seyed Hossein Haeri. *The ΔQSD Paradigm: Designing Systems with Predictable Performance at High Load. Full-day tutorial*. 15th ACM/SPEC International Conference on Performance Engineering. 2024. URL: <https://webperso.info.ucl.ac.be/~pvr/ICPE-2024-deltaQSD-full.pdf>.
- [2] Peter Thompson and Rudy Hernandez. *Quality Attenuation Measurement Architecture and Requirements*. Tech. rep. MSU-CSE-06-2. Sept. 2020. URL: <https://www.broadband-forum.org/pdfs/tr-452.1-1-0-0.pdf>.
- [3] Seyed Hossein Haeri et al. “Algebraic Reasoning About Timeliness”. In: *Proceedings 16th Interaction and Concurrency Experience, ICE 2023, Lisbon, Portugal, 19th June 2023*. Ed. by Clément Aubert et al. Vol. 383. EPTCS. 2023, pp. 35–54. DOI: 10.4204/EPTCS.383.3. URL: <https://doi.org/10.4204/EPTCS.383.3>.
- [4] Erlang/OTP. *What is Erlang*. Accessed: (26/05/2025). 2025. URL: <https://www.erlang.org/faq/introduction.html>.
- [5] Francesco Nieri. *Master thesis presentation poster UCLouvain - ΔQ oscilloscope*. 2024.
- [6] Seyed H. Haeri et al. “Mind Your Outcomes: The ΔQSD Paradigm for Quality-Centric Systems Development and Its Application to a Blockchain Case Study”. In: *Comput.* 11.3 (2022), p. 45. DOI: 10.3390/COMPUTERS11030045. URL: <https://doi.org/10.3390/computers11030045>.
- [7] Peter Van Roy. *LINFO2345 lessons on ΔQSD* . Accessed: (19/05/2025). UCLouvain, 2023. URL: <https://www.youtube.com/watch?v=tF7fbU9Gce8>.
- [8] Neil J. Davies and Peter W. Thompson. *ΔQSD workbench - GitHub*. Accessed: (19/05/2025). 2022. URL: <https://github.com/DeltaQ-SD/dqsd-workbench>.
- [9] Erlang programming language. *Erlang tracing*. Accessed: (19/05/2025). 2024. URL: <https://www.erlang.org/doc/apps/erts/tracing.html>.
- [10] OpenTelemetry. *OpenTelemetry in Erlang/Elixir*. Accessed: (19/05/2025). 2025. URL: <https://opentelemetry.io/docs/languages/erlang/>.
- [11] Isaak D. Mayergoyz and W. Lawson (Auth.) *Basic Electric Circuit Theory. A One-Semester Text*. pg. 116. Academic Press, 1996. ISBN: 9780080572284; 0080572286; 9780124808652; 0124808654.
- [12] OpenTelemetry. *What is OpenTelemetry?* Accessed: (19/05/2025). 2025. URL: <https://opentelemetry.io/docs/what-is-opentelemetry/>.
- [13] Hazel Weakly. *OpenTelemetry Challenges: Handling Long-Running Spans*. Accessed: (21/05/2025). 2024. URL: <https://thenewstack.io/opentelemetry-challenges-handling-long-running-spans/>.

- [14] OpenTelemetry. *Instrumentation for OpenTelemetry Erlang/Elixir*. Accessed: (19/05/2025). 2025. URL: <https://opentelemetry.io/docs/languages/erlang/instrumentation/>.
- [15] Erlang Ecosystem Foundation. *Observability Working Group*. Accessed: (28/05/2025). 2025. URL: <https://erlef.org/wg/observability>.
- [16] OpenTelemetry Authors. *Observability primer - Distributed traces*. Accessed: (28/05/2025). 2024. URL: <https://opentelemetry.io/docs/concepts/observability-primer/>.
- [17] OpenTelemetry. *OpenTelemetry - Traces*. Accessed: (19/05/2025). 2025. URL: <https://opentelemetry.io/docs/concepts/signals/traces/>.
- [18] OpenTelemetry Authors. *Exporters*. Accessed: (28/05/2025). 2025. URL: <https://opentelemetry.io/docs/languages/erlang/exporters/>.
- [19] Datadog. *Datadog - Product*. Accessed: (28/05/2025). 2025. URL: <https://www.datadoghq.com/product/>.
- [20] Dotan Horovits. *From Distributed Tracing to APM: Taking OpenTelemetry & Jaeger Up a Level*. Accessed: (19/05/2025). 2021. URL: <https://logz.io/blog/monitoring-microservices-opentelemetry-jaeger/>.
- [21] Sampath Siva Kumar Boddeti. *Tracing Made Easy: A Beginner's Guide to Jaeger and Distributed Systems*. Accessed: (19/05/2025). 2024. URL: <https://openobserve.ai/blog/tracing-made-easy-a-beginners-guide-to-jaeger-and-distributed-systems/>.
- [22] OpenTelemetry. *Active spans, C++ Instrumentation*. Accessed: (19/05/2025). 2025. URL: <https://opentelemetry.io/docs/languages/cpp/instrumentation/>.
- [23] OpenTelemetry authors. *Error handling*. Accessed: (29/05/2025). 2022. URL: <https://opentelemetry.io/docs/specs/otel/error-handling/>.
- [24] Francesco Nieri. *dqsd_otel*. Accessed: (28/05/2025). 2025. URL: https://github.com/fnieri/dqsd_otel.
- [25] Francesco Nieri. *ΔQ Oscilloscope*. Accessed: (28/05/2025). URL: <https://github.com/fnieri/DeltaQOscilloscope>.
- [26] KeySight. *What is an Oscilloscope Trigger?* Accessed: (23/05/2025). 2022. URL: <https://www.keysight.com/used/id/en/knowledge/glossary/oscilloscopes/what-is-an-oscilloscope-trigger>.
- [27] Alex Tsun. *Statistical inference - Confidence intervals*. Accessed: (29/05/2025). 2020. URL: https://courses.cs.washington.edu/courses/cse312/20su/files/student_drive/8.1.pdf.
- [28] FM Dekking et al. *A modern introduction to probability theory and statistics*. Vol. Springer texts in statistics. Springer, 2005. ISBN: 1-85233-896-2.
- [29] Steven B. Damelin and Willard Miller Jr. *The Mathematics of Signal Processing*. USA: Cambridge University Press, 2012. ISBN: 1107601045.
- [30] FFTW3. *Fastest Fourier Transform in The West*. Accessed: (19/05/2025). 2025. URL: <https://www.fftw.org/>.
- [31] Jeremy Fix. *FFTConvolution*. Accessed: (21/05/2025). 2013. URL: https://github.com/jeremyfix/FFTConvolution/blob/master/Convolution/src/convolution_fftw.h.

- [32] *Planning-rigor flags*. Accessed: (23/05/2025). URL: <https://www.fftw.org/doc/Planner-Flags.html>.
- [33] Rebar3. *Rebar3 Basic Usage*. Accessed: (25/05/2025). 2025. URL: https://rebar3.org/docs/basic_usage/.
- [34] Erlang System Documentation. *Signals/Sending signals*. Accessed: (24/05/2025). 2025. URL: https://www.erlang.org/doc/system/ref_man_processes.
- [35] ANTLR. *What is ANTLR4?* Accessed: (19/05/2025). 2025. URL: <https://www.antlr.org/>.
- [36] Terence Parr and Kathleen Fisher. “LL(*): the foundation of the ANTLR parser generator”. In: *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*. Ed. by Mary W. Hall and David A. Padua. ACM, 2011, pp. 425–436. DOI: 10.1145/1993498.1993548. URL: <https://doi.org/10.1145/1993498.1993548>.
- [37] lark-parser. *Lark - A parsing toolkit for Python*. Accessed: (24/05/2025). 2025. URL: <https://github.com/lark-parser/lark>.
- [38] Wikipedia. *Qt (software) Wikipedia, The Free Encyclopedia*. Accessed: (24/05/2025). 2025. URL: [https://en.wikipedia.org/wiki/Qt_\(software\)](https://en.wikipedia.org/wiki/Qt_(software)).
- [39] Tristan Slougher. *ctx*. Accessed: (21/05/2025). 2023. URL: <https://github.com/tslougher/ctx>.
- [40] The Qt Company. *Add-ons available under Commercial Licenses, or GNU General Public License v3*. Accessed: (23/05/2025). 2025. URL: <https://doc.qt.io/qt-5/qtmodules.html>.

Appendix A

Tests specifications

The tests were run on a laptop with the following specifications.

- OS: Manjaro Linux x86_64, Kernel: 6.12.11-1-MANJARO
- CPU: Intel i5-6300U (4) @ 3.000GHz
- GPU: Intel Skylake GT2 [HD Graphics 520]
- RAM: 16 GB DDR4 SDRAM—2133MHz

Appendix B

User manual

B.1 Sidebar: Outcome diagram and plots

B.1.1 Creating the system

You need to provide a textual description of your outcome diagram following the syntax which was defined previously. You can create your system by clicking on the **Create or edit system** button. If the parser successfully parsed the text, your system will be created and you can start setting the parameters for your probes.

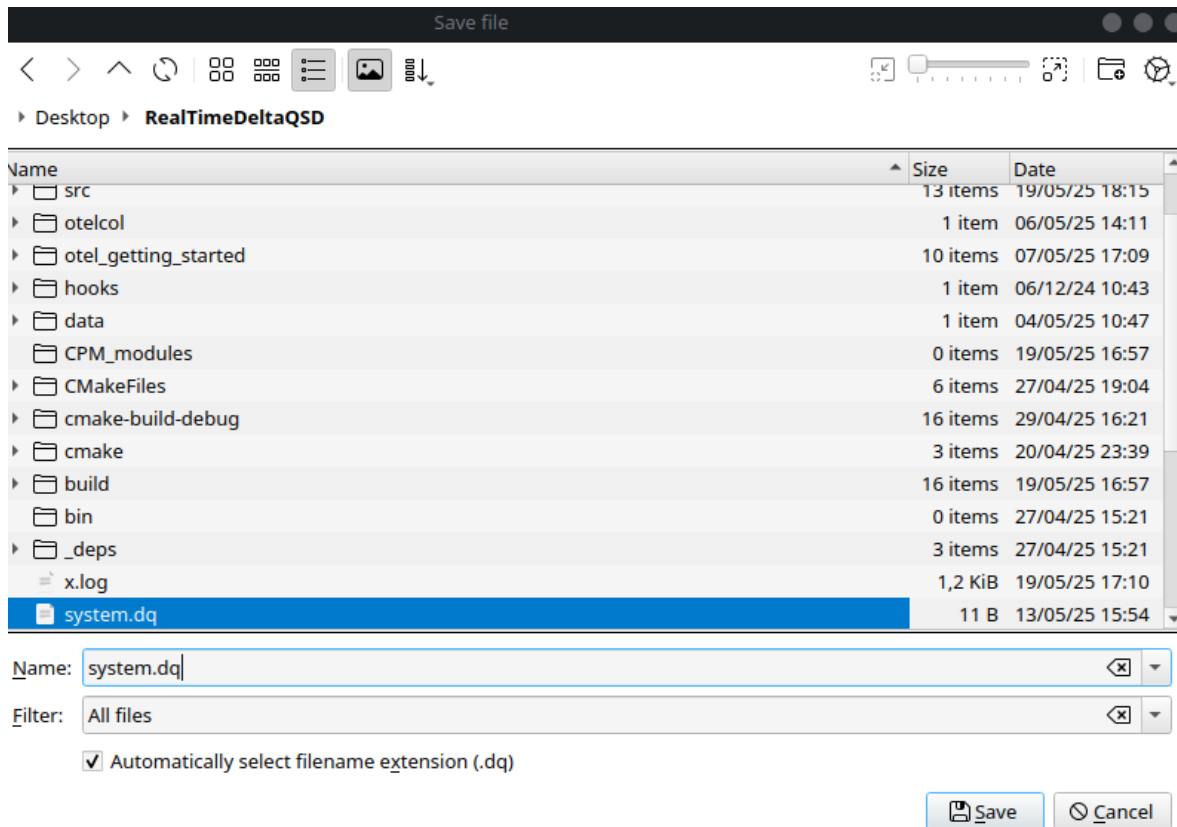
If the parser does not correctly parse the text, it will show a popup indicating the line where the error was produced and what it was expecting.

B.1.2 Saving the system definition

The button **Save system to** gives you the possibility to save the textual definition of the outcome diagram to a file, you can save the file to any extension, but preferably the file will have the **.dq** extension.

B.1.3 Loading the system definition

The button **Load system to** gives you the possibility to load the textual definition of the outcome diagram you may have previously saved to a file, as for saving, you can load any file extension.



B.1.4 Managing the plots

Once you have your system defined you can start adding plots of the ΔQ s of the probes you inserted in your system.

Adding a plot

Multiple probes can be added at once in your plot, you can select the probes you want to add to a new plot by selecting them in the **"Add a new plot"** section. Then by clicking the **"Add plot"** button, the selected probes will be added to the plot.

Editing a plot

You can remove the probes you have added to the plot by first clicking to it, this sets the plot as the **selected plot**. Once you have clicked the plot, a section will pop up beneath the rest of the controls on the sidebar.

In the section there are two subsections, one which shows the selected components which form the plot (those you have added previously), and the available components. You can select the probes you want to remove in the **"selected components"** zone, by clicking **"Remove selected components"** the components will be removed from the plot. Inversely, in the **"Available components"** section, you can select the plots you want to add, and by clicking **"Add selected components"** you can add the selected components to the selected plot.

B.1.5 Removing a plot

By left-clicking the plot, a popup appears, you can click **"Remove plot"**, this removes it from the selected plot.

B.2 Sidebar: Probes settings

Now that you have create your outcome diagram, you can modify the probes settings to set the $dMax$ you want and the QTA you desire.

B.2.1 Setting a QTA

You can set a QTA in the **"Set a QTA"** section, there you are presented with the possibility to:

- Select the probe for which you want to set the QTA.
- Set the QTA at the three percentiles (25%, 50%, 75%), the text to the left indicates which percentile is which. You need to set the delay in seconds.
- The minimum amount of successful events you can allow, which is bigger than 75%.

Of course, for the delay of the QTA at three percentiles, the delay at the percentile must be higher or equal than the delay at the previous percentile and higher than 0.

By pressing *"Save QTA settings"* you will save the QTA for the defined probe.

B.2.2 Setting the parameters of a probe

You can set the parameters for a probe in its section. To the left you can select the probe you want to set the parameter for. We provide a slider (which goes from -10 to 10) for the n parameters, to the right of the slider, you can select the number of bins for the probe. The maximum delay calculated will be shown below.

Once you press **"Save delay"**, a message to the erlang adapter will be sent, which will set the maximum delay you have set in the adapter.

Sidebar

Probes settings

Triggers

Set QTA for a probe:

Probe: probe

25th Percentile (s): 0.1

50th Percentile (s): 0.25

75th Percentile (s): 0.5

Max. allowed failure (0-1): 0.95

Save QTA Settings

Set the parameters for a probe.

probe

1000

Max delay is: 62.50 ms

Save delay

Figure B.1: Probes settings tab. Above: QTA settings. Below: Probe parameters settings.

B.3 Triggers

In the triggers section you can define which triggers to apply for a given probe, once selected, they will be automatically activated.

Once a trigger is triggered, the oscilloscope will keep recording the system for a few seconds. Once stopped, under the "snapshots" section, you can click the snapshots to view them in a separate section, there, you can observe the ΔQ s of all the probes before and after the trigger was triggered. You can discard the snapshot by left clicking on it and clicking *"Deleted snapshot"*.

B.4 Instrumenting the Erlang application

B.4.1 Including the adapter

The Erlang project you need to instrument needs to include the adapter in its dependencies, to do that, you need to include it in your dependencies.

```
%your_app.app.src
{application, otel_getting_started, [
    ...
    {applications, [
        kernel,
        stdlib,
        opentelemetry,
        opentelemetry_api,
        opentelemetry_exporter,
        dqsdtel
    ]},
    ...
]}.

{deps, [
    {opentelemetry, "~> 1.3"},
    {opentelemetry_api, "~> 1.2"},
    {opentelemetry_exporter, "~> 1.0"},
    {dqsdtel, {git, "https://github.com/fnieri/dqsdtel.git", {tag,
        ↪ "the_latest_version_on_git"}}}
]}.

```

Once you have the dependencies set up you can begin creating outcome instances for the oscilloscope.

(**Note:** If the project were to change name, you can still find the project in <https://github.com/fnieri/>).

B.4.2 Starting spans

To start spans you need to call:

```
{ProbeCtx, Pid} = dqsdtel:start_span(<<"probe">>, #{attributes =>
  ↳ [{attr, <<"my_attr_5o5s10">>}]})
```

This will give you the OpenTelemetry context of the probe and the Pid of the process to call upon end. It is left up to you to decide how to carry both in the execution. The function calls OpenTelemetry `?start_span` macro, effectively replacing it.

B.4.3 Ending spans

To end spans you need to call:

```
dqsdtel:end_span(ProbeCtx, ProbePid)
```

This will end the span on the OpenTelemetry side and end the outcome instance if it hasn't timedout. The function calls OpenTelemetry `?end_span` macro, effectively replacing it.

B.4.4 Failing outcome instances

To fail **custom** spans you need to call:

```
dqsdtel:fail_span(WorkerPid),
```

Contrary to the other methods, this does not end OpenTelemetry spans, it is let up to you to decide how to handle failure in spans.

B.5 Establishing connection to the oscilloscope

WIP

Part I

Source Code Appendix

Appendix C

C++ Source Files

C.1 Root

C.1.1 Application.cpp

```
1 #include "Application.h"
2
3 Application::Application()
4 {
5 }
6 Application &Application::getInstance()
7 {
8     static Application instance;
9     return instance;
10 }
11
12 std::shared_ptr<System> Application::getSystem()
13 {
14     return system;
15 }
16
17 void Application::addObserver(std::function<void()> callback)
18 {
19     observers.push_back(callback);
20 }
21
22 void Application::notifyObservers()
23 {
24     for (auto &observer : observers) {
25         observer();
26     }
27 }
28
29 void Application::setServer(Server *s)
30 {
31     server = s;
32 }
33
34 void Application::sendDelayChange(std::string &name, double newDelay)
```

```

35 {
36     server->sendToErlang("set_timeout;" + name + ';' + std::to_string(
newDelay));
37 }
38
39 void Application::setStubRunning(bool running)
40 {
41     if (running)
42         server->sendToErlang("start_stub");
43     else
44         server->sendToErlang("stop_stub");
45 }
46
47 SystemDiff Application::diffWith(System &newSystem)
48 {
49     SystemDiff diff;
50
51     auto oldSystem = getSystem();
52     if (!oldSystem) {
53         // No system yet: everything is new
54         for (const auto &[name, _] : newSystem.getProbes())
55             diff.addedProbes.push_back(name);
56         for (const auto &[name, _] : newSystem.getOperators())
57             diff.addedOperators.push_back(name);
58         for (const auto &[name, _] : newSystem.getOutcomes())
59             diff.addedOutcomes.push_back(name);
60         return diff;
61     }
62
63     auto &oldProbes = oldSystem->getProbes();
64     auto &newProbes = newSystem.getProbes();
65
66     for (const auto &[name, probe] : newProbes) {
67         if (!oldProbes.count(name)) {
68             diff.addedProbes.push_back(name);
69         } else if (componentsDiffer(oldProbes.at(name), probe)) {
70             diff.changedProbes.push_back(name);
71         }
72     }
73     for (const auto &[name, _] : oldProbes) {
74         if (!newProbes.count(name))
75             diff.removedProbes.push_back(name);
76     }
77
78     auto &oldOps = oldSystem->getOperators();
79     auto &newOps = newSystem.getOperators();
80
81     for (const auto &[name, op] : newOps) {
82         if (!oldOps.count(name)) {
83             diff.addedOperators.push_back(name);
84         } else if (componentsDiffer(oldOps.at(name), op)) {
85             diff.changedOperators.push_back(name);
86         }
87     }
88     for (const auto &[name, _] : oldOps) {
89         if (!newOps.count(name))

```

```

90         diff.removedOperators.push_back(name);
91     }
92
93     auto &oldOut = oldSystem->getOutcomes();
94     auto &newOut = newSystem.getOutcomes();
95
96     for (const auto &[name, out] : newOut) {
97         if (!oldOut.count(name)) {
98             diff.addedOutcomes.push_back(name);
99         } else if (componentsDiffer(oldOut.at(name), out)) {
100             diff.changedOutcomes.push_back(name);
101         }
102     }
103     for (const auto &[name, _] : oldOut) {
104         if (!newOut.count(name))
105             diff.removedOutcomes.push_back(name);
106     }
107
108     return diff;
109 }
110
111 bool Application::componentsDiffer(const std::shared_ptr<Observable> &
112 a, const std::shared_ptr<Observable> &b)
113 {
114     if (!a || !b || a->getName() != b->getName())
115         return true;
116
117     // For Probes, check causalLinks. For Operators, check type,
118     // probabilities, children.
119     if (auto pa = std::dynamic_pointer_cast<Probe>(a)) {
120         auto pb = std::dynamic_pointer_cast<Probe>(b);
121         if (!pb)
122             return true;
123
124         auto ca = pa->getCausalLinks();
125         auto cb = pb->getCausalLinks();
126
127         if (ca.size() != cb.size())
128             return true;
129         for (size_t i = 0; i < ca.size(); ++i) {
130             if (ca[i]->getName() != cb[i]->getName())
131                 return true;
132         }
133     }
134
135     if (auto oa = std::dynamic_pointer_cast<Operator>(a)) {
136         auto ob = std::dynamic_pointer_cast<Operator>(b);
137         if (!ob)
138             return true;
139
140         if (oa->getType() != ob->getType())
141             return true;
142
143         auto ca = oa->getCausalLinks();
144         auto cb = ob->getCausalLinks();

```

```

144         if (ca.size() != cb.size())
145             return true;
146         for (size_t i = 0; i < ca.size(); ++i) {
147             if (ca[i].size() == cb[i].size()) {
148                 for (size_t j = 0; j < ca[i].size(); ++j) {
149                     if (ca[i][j]->getName() != cb[i][j]->getName())
150                         return true;
151                 }
152             } else {
153                 return true;
154             }
155         }
156         auto probsA = oa->getProbabilities();
157         auto probsB = ob->getProbabilities();
158         if (probsA != probsB)
159             return true;
160     }
161 }
162
163 return false;
164 }
165
166 void Application::setSystem(System newSystem)
167 {
168     if (!system) {
169         system = std::make_shared<System>(newSystem);
170         notifyObservers();
171         return;
172     }
173     SystemDiff diff = diffWith(newSystem);
174
175     // Apply removals
176     for (const auto &name : diff.removedProbes) {
177         system->getProbes().erase(name);
178         system->getObservables().erase(name);
179     }
180     for (const auto &name : diff.removedOperators) {
181         system->getOperators().erase(name);
182         system->getObservables().erase(name);
183     }
184
185     for (const auto &name : diff.removedOutcomes) {
186         system->getOutcomes().erase(name);
187         system->getObservables().erase(name);
188     }
189
190     // Apply changes and additions
191     for (const auto &name : diff.changedProbes) {
192         system->getProbes()[name]->setCausalLinks(newSystem.getProbes()
193             .at(name)->getCausalLinks());
194         system->getObservables()[name] = newSystem.getProbes().at(name);
195     }
196     for (const auto &name : diff.addedProbes) {
197         system->getProbes()[name] = newSystem.getProbes().at(name);

```

```

197     system->getObservables()[name] = newSystem.getProbes().at(name
198 );
199 }
200 for (const auto &name : diff.changedOperators) {
201     system->getOperators()[name] = newSystem.getOperators().at(
202 name);
203     system->getObservables()[name] = newSystem.getOperators().at(
204 name);
205 }
206 for (const auto &name : diff.addedOperators) {
207     system->getOperators()[name] = newSystem.getOperators().at(
208 name);
209     system->getObservables()[name] = newSystem.getOperators().at(
210 name);
211 }
212 for (const auto &name : diff.changedOutcomes)
213     system->getOutcomes()[name] = newSystem.getOutcomes().at(name)
214 ;
215 for (const auto &name : diff.addedOutcomes) {
216     system->getOutcomes()[name] = newSystem.getOutcomes().at(name)
217 ;
218     system->getObservables()[name] = newSystem.getOutcomes().at(
219 name);
220 }
221 std::string defText = newSystem.getSystemDefinitionText();
222 system->setSystemDefinitionText(defText);
223
224 notifyObservers();
225 }
226
227 bool Application::startCppServer(const std::string &&ip, int port)
228 {
229     return server->startServer(ip, port);
230 }
231
232 void Application::stopCppServer()
233 {
234     server->stopServer();
235 }
236
237 void Application::setErlangEndpoint(const std::string &&ip, int port)
238 {
239     server->setErlangEndpoint(ip, port);
240 }

```

C.2 dashboard

C.2.1 ColorRegistry.cpp

```

1 #include "ColorRegistry.h"
2 #include <cmath>
3
4 std::unordered_map<std::string, QColor> ColorRegistry::colorMap;
5

```

```
6 QColor ColorRegistry::getColorFor(const std::string &name)
7 {
8     if (colorMap.count(name))
9         return colorMap[name];
10
11     int index = colorMap.size();
12     QColor color = generateDistinctColor(index);
13     colorMap[name] = color;
14     return color;
15 }
16
17 // Taken from
18 // https://martin.ankerl.com/2009/12/09/how-to-create-random-colors-
19 // programmatically/
20 QColor ColorRegistry::generateDistinctColor(int index)
21 {
22     const double golden_ratio_conjugate = 137.508; // degrees
23     double hue = std::fmod(index * golden_ratio_conjugate, 360.0);
24     QColor color;
25     color.setHsvF(hue / 360.0, 0.7, 0.95); // Saturation and Value
26     // tuned for brightness
27     return color;
28 }
```

C.2.2 CustomLegendEntry.cpp

```
1 #include "CustomLegendEntry.h"
2 #include <QHBoxLayout>
3 CustomLegendEntry::CustomLegendEntry(const QString &name, const QColor
4     &color, QWidget *parent)
5     : QWidget(parent)
6 {
7     layout = new QHBoxLayout(this);
8     colorBox = new QLabel;
9     colorBox->setFixedSize(12, 12);
10    colorBox->setStyleSheet(QString("background-color: %1").arg(color.
11        name()));
12    nameLabel = new QLabel(name);
13    layout->addWidget(colorBox);
14    layout->addWidget(nameLabel);
15    layout->addStretch();
16 }
```

C.2.3 CustomLegendPanel.cpp

```
1 #include "CustomLegendPanel.h"
2 #include "CustomLegendEntry.h"
3
4 CustomLegendPanel::CustomLegendPanel(QWidget *parent)
5     : QWidget(parent)
6 {
7     // Initialize scroll area and configure its behavior
8     scrollArea = new QScrollArea(this);
```

```

9     scrollArea->setWidgetResizable(true);
10    scrollArea->setVerticalScrollBarPolicy(Qt::ScrollBarAsNeeded);
11    scrollArea->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
12
13    // Create content widget for the scroll area
14    scrollContent = new QWidget();
15    scrollArea->setWidget(scrollContent);
16
17    // Set up layout for legend entries (aligned to top)
18    legendLayout = new QVBoxLayout(scrollContent);
19    legendLayout->setAlignment(Qt::AlignTop);
20
21    // Set up main layout and add scroll area
22    mainLayout = new QVBoxLayout(this);
23    mainLayout->addWidget(scrollArea);
24    setLayout(mainLayout);
25 }
26
27 /**
28  * @brief Adds a new entry to the legend panel.
29  * @param name The name for the entry.
30  * @param color The color for the entry.
31  */
32 void CustomLegendPanel::addEntry(const QString &name, const QColor &
    color)
33 {
34     auto entry = new CustomLegendEntry(name, color, this);
35     legendLayout->addWidget(entry);
36     legendEntries[name] = entry;
37 }
38
39 /**
40  * @brief Removes a specific entry from the legend by name.
41  * @param name The name of the entry to remove.
42  */
43 void CustomLegendPanel::removeEntry(const QString &name)
44 {
45     if (legendEntries.count(name)) {
46         QWidget *entry = legendEntries[name];
47         legendLayout->removeWidget(entry);
48         entry->deleteLater();
49         legendEntries.erase(name);
50     }
51 }
52
53 /**
54  * @brief Clears all entries from the legend panel.
55  */
56 void CustomLegendPanel::clear()
57 {
58     for (auto it = legendEntries.begin(); it != legendEntries.end();)
59     {
60         QWidget *entry = it->second;
61         legendLayout->removeWidget(entry);
62         entry->deleteLater();
63         it = legendEntries.erase(it);

```



```

63     }
64 }

```

C.2.4 DQPlotController.cpp

```

1
2 #include "DQPlotController.h"
3 #include "../Application.h"
4 #include <QMetaObject>
5 #include <QVector>
6 #include <QtConcurrent>
7 #include <algorithm>
8 #include <cstdlib>
9 #include <qcontainerfwd.h>
10 #include <qlineseries.h>
11 using namespace std::chrono;
12 DQPlotController::DQPlotController(DeltaQPlot *plot, const std::vector
    <std::string> &selectedItems)
13     : plot(plot)
14 {
15     auto system = Application::getInstance().getSystem();
16     for (const auto &name : selectedItems) {
17         addComponent(name, (system->hasOutcome(name)));
18     }
19     setTitle();
20 }
21
22 DQPlotController::~DQPlotController()
23 {
24     outcomes.clear();
25     probes.clear();
26     operators.clear();
27 }
28
29 bool DQPlotController::isEmptyAfterReset()
30 {
31     auto system = Application::getInstance().getSystem();
32
33     std::lock_guard<std::mutex> lock(resetMutex);
34
35     for (auto it = outcomes.begin(); it != outcomes.end(); ) {
36         if (!system->hasOutcome(it->first)) {
37             removeComponent(it->first);
38             it = outcomes.begin();
39         } else {
40             ++it;
41         }
42     }
43
44     for (auto it = probes.begin(); it != probes.end(); ) {
45         if (!system->hasProbe(it->first)) {
46             removeComponent(it->first);
47             it = probes.begin();
48         } else {
49             ++it;

```

```

50     }
51 }
52 for (auto it = operators.begin(); it != operators.end(); ) {
53     if (!system->hasOperator(it->first)) {
54         removeComponent(it->first);
55         it = operators.begin();
56     } else {
57         ++it;
58     }
59 }
60 return (outcomes.empty() && probes.empty() && operators.empty());
61 }
62
63 bool DQPlotController::containsComponent(std::string name)
64 {
65     return ((outcomes.find(name) != outcomes.end()) || (probes.find(
66         name) != probes.end()) || (operators.find(name) != operators.end()
67         ));
68 }
69
70 void DQPlotController::setTitle()
71 {
72     std::vector<std::string> existingItems = getComponents();
73     std::string title = "Plot of: ";
74     for (const auto &name : existingItems) {
75         title += name + " ";
76     }
77     plot->setTitle(QString::fromStdString(title));
78 }
79
80 void DQPlotController::editPlot(const std::vector<std::string> &
81     selectedItems)
82 {
83     std::vector<std::string> existingItems = getComponents();
84
85     for (const auto &name : existingItems) {
86         if (std::find(selectedItems.begin(), selectedItems.end(), name
87             ) == selectedItems.end()) {
88             removeComponent(name);
89         }
90     }
91
92     auto system = Application::getInstance().getSystem();
93
94     for (const auto &name : selectedItems) {
95         if (!containsComponent(name)) {
96             addComponent(name, system->hasOutcome(name));
97         }
98     }
99     setTitle();
100 }
101
102 void DQPlotController::addComponent(const std::string &name, bool
103     isOutcome)
104 {
105     auto system = Application::getInstance().getSystem();
106     if (isOutcome) {

```

```

101         addOutcomeSeries(name);
102     } else {
103         addExpressionSeries(name, system->hasProbe(name));
104     }
105 }
106
107 std::vector<std::string> DQPlotController::getComponents()
108 {
109     std::vector<std::string> components;
110     components.reserve(probes.size() + outcomes.size() + operators.
111 size());
112
113     for (const auto &kv : probes) {
114         components.push_back(kv.first);
115     }
116     for (const auto &kv : outcomes) {
117         components.push_back(kv.first);
118     }
119     for (const auto &kv : operators) {
120         components.push_back(kv.first);
121     }
122     return components;
123 }
124
125 QLineSeries *DQPlotController::createAndAddLineSeries(const std::
126 string &legendName)
127 {
128     auto series = new QLineSeries();
129     plot->addSeries(series, legendName);
130     return series;
131 }
132
133 void DQPlotController::addOutcomeSeries(const std::string &name)
134 {
135     auto system = Application::getInstance().getSystem();
136
137     auto lowerBoundSeries = createAndAddLineSeries(name + "(0) lower
138 bound");
139     auto upperBoundSeries = createAndAddLineSeries(name + "(0) upper
140 bound");
141     auto outcomeSeries = createAndAddLineSeries(name + "(0) observed");
142     ;
143     auto meanSeries = createAndAddLineSeries(name + "(0) mean");
144     auto qtaSeries = createAndAddLineSeries(name + "(0) qta");
145
146     OutcomeSeries series
147         = {.outcomeS = outcomeSeries, .lowerBoundS = lowerBoundSeries,
148 .upperBoundS = upperBoundSeries, .meanS = meanSeries, .qtaS =
149 qtaSeries};
150
151     outcomes[name] = {series, system->getOutcome(name)};
152 }
153
154 void DQPlotController::addExpressionSeries(const std::string &name,
155 bool isProbe)
156 {

```

```

149     auto system = Application::getInstance().getSystem();
150
151     auto obsLowerBoundSeries = createAndAddLineSeries(name + "(O
lower bound");
152     auto obsUpperBoundSeries = createAndAddLineSeries(name + " (O
upper bound");
153     auto obsSeries = createAndAddLineSeries(name + "(O) observed");
154     auto obsMeanSeries = createAndAddLineSeries(name + " (O) mean");
155
156     auto qtaSeries = createAndAddLineSeries(name + " qta");
157
158     auto calcSeries = createAndAddLineSeries(name + "(C) calculated");
159     auto calcLowerBoundSeries = createAndAddLineSeries(name + "(C
lower bound");
160     auto calcUpperBoundSeries = createAndAddLineSeries(name + "(C
upper bound");
161     auto calcMeanSeries = createAndAddLineSeries(name + "(C) mean");
162
163     ExpressionSeries series = {
164         .obsS = obsSeries,
165         .obsLowerBoundS = obsLowerBoundSeries,
166         .obsUpperBoundS = obsUpperBoundSeries,
167         .obsMeanS = obsMeanSeries,
168         .calcS = calcSeries,
169         .calcLowerBoundS = calcLowerBoundSeries,
170         .calcUpperBoundS = calcUpperBoundSeries,
171         .calcMeanS = calcMeanSeries,
172         .qtaS = qtaSeries,
173     };
174     if (isProbe)
175         probes[name] = {series, system->getProbe(name)};
176     else
177         operators[name] = {series, system->getOperator(name)};
178 }
179
180 void DQPlotController::removeOutcomeSeries(const std::string &name)
181 {
182     OutcomeSeries series = outcomes[name].first;
183
184     plot->removeSeries(series.outcomeS);
185     delete series.outcomeS;
186     series.outcomeS = NULL;
187
188     plot->removeSeries(series.lowerBoundS);
189     delete series.lowerBoundS;
190     series.lowerBoundS = NULL;
191
192     plot->removeSeries(series.upperBoundS);
193     delete series.upperBoundS;
194     series.upperBoundS = NULL;
195
196     plot->removeSeries(series.meanS);
197     delete series.meanS;
198     series.meanS = NULL;
199
200     plot->removeSeries(series.qtaS);

```

```
201     delete series.qtaS;
202     series.qtaS = NULL;
203
204     outcomes.erase(name);
205 }
206
207 void DQPlotController::removeExpressionSeries(const std::string &name,
208     bool isProbe)
209 {
210     ExpressionSeries series;
211     if (isProbe)
212         series = probes[name].first;
213     else
214         series = operators[name].first;
215
216     plot->removeSeries(series.obsLowerBoundS);
217     delete series.obsLowerBoundS;
218     series.obsLowerBoundS = NULL;
219
220     plot->removeSeries(series.obsUpperBoundS);
221     delete series.obsUpperBoundS;
222     series.obsUpperBoundS = NULL;
223
224     plot->removeSeries(series.obsS);
225     delete series.obsS;
226     series.obsS = NULL;
227
228     plot->removeSeries(series.obsMeanS);
229     delete series.obsMeanS;
230     series.obsMeanS = NULL;
231
232     plot->removeSeries(series.qtaS);
233     delete series.qtaS;
234     series.qtaS = NULL;
235
236     plot->removeSeries(series.calcS);
237     delete series.calcS;
238     series.calcS = NULL;
239
240     plot->removeSeries(series.calcLowerBoundS);
241     delete series.calcLowerBoundS;
242     series.calcLowerBoundS = NULL;
243
244     plot->removeSeries(series.calcUpperBoundS);
245     delete series.calcUpperBoundS;
246     series.calcUpperBoundS = NULL;
247
248     plot->removeSeries(series.calcMeanS);
249     delete series.calcMeanS;
250     series.calcMeanS = NULL;
251
252     if (isProbe)
253         probes.erase(name);
254     else
255         operators.erase(name);
256 }
```

```

256
257 void DQPlotController::removeComponent(const std::string &name)
258 {
259     if (outcomes.count(name)) {
260         removeOutcomeSeries(name);
261     }
262     removeExpressionSeries(name, probes.count(name));
263 }
264
265 void DQPlotController::update(uint64_t timeLowerBound, uint64_t
timeUpperBound)
266 {
267     std::lock_guard<std::mutex> lock(updateMutex);
268     double outcomeMax = 0;
269
270     for (auto &[name, seriesOutcome] : outcomes) {
271         double outcomeRange = updateOutcome(seriesOutcome.first,
seriesOutcome.second, timeLowerBound, timeUpperBound);
272         if (outcomeRange > outcomeMax) {
273             outcomeMax = outcomeRange;
274         }
275     }
276
277     double probeMax = 0;
278     for (auto &[name, seriesProbe] : probes) {
279         if (seriesProbe.second) {
280             double probeRange = updateProbe(probes[name].first, probes
[name].second, timeLowerBound, timeUpperBound);
281             if (probeRange > probeMax) {
282                 probeMax = probeRange;
283             }
284         }
285     }
286
287     double operatorMax = 0;
288     for (auto &[name, seriesOp] : operators) {
289         if (seriesOp.second) {
290             double opRange = updateOperator(operators[name].first,
operators[name].second, timeLowerBound, timeUpperBound);
291             if (opRange > operatorMax) {
292                 operatorMax = opRange;
293             }
294         }
295     }
296     plot->updateXRange(std::max({outcomeMax, probeMax, operatorMax}));
297 }
298
299 double DQPlotController::updateOutcome(OutcomeSeries &series, const
std::shared_ptr<Outcome> &outcome, uint64_t timeLowerBound,
uint64_t timeUpperBound)
300 {
301     auto ret = QtConcurrent::run([=]() {
302         double maxDelay = outcome->getMaxDelay();
303         DeltaQRepr repr = outcome->getObservedDeltaQRepr(
timeLowerBound, timeUpperBound);
304

```

```

305     DeltaQ deltaQ = repr.deltaQ;
306     std::vector<Bound> bounds = repr.bounds;
307     int size = deltaQ.getBins();
308     double binWidth = deltaQ.getBinWidth();
309
310     QVector<QPointF> deltaQData;
311     deltaQData.emplace_back(QPointF(0, 0));
312     deltaQData.reserve(size);
313
314     QVector<QPointF> lowerBoundData;
315     lowerBoundData.emplace_back(QPointF(0, 0));
316     lowerBoundData.reserve(size);
317
318     QVector<QPointF> upperBoundData;
319     upperBoundData.emplace_back(QPointF(0, 0));
320     upperBoundData.reserve(size);
321
322     QVector<QPointF> meanData;
323     meanData.emplace_back(QPointF(0, 0));
324     meanData.reserve(size);
325
326     QVector<QPointF> qtaData;
327     auto qta = outcome->getQTA();
328     for (int i = 0; i < size; ++i) {
329         double x = binWidth * (i + 1);
330
331         deltaQData.emplace_back(QPointF(x, deltaQ.cdfAt(i)));
332         lowerBoundData.emplace_back(QPointF(x, bounds[i].
lowerBound));
333         upperBoundData.emplace_back(QPointF(x, bounds[i].
upperBound));
334         meanData.emplace_back(QPointF(x, bounds[i].mean));
335     }
336
337     if (qta.defined) {
338         qtaData.reserve(8);
339         qtaData.emplace_back(qta.perc_25, 0);
340         qtaData.emplace_back(qta.perc_25, 0.25);
341         qtaData.emplace_back(qta.perc_50, 0.25);
342         qtaData.emplace_back(qta.perc_50, 0.5);
343         qtaData.emplace_back(qta.perc_75, 0.5);
344         qtaData.emplace_back(qta.perc_75, 0.75);
345         qtaData.emplace_back(maxDelay, 0.75);
346         qtaData.emplace_back(maxDelay, qta.cdfMax);
347     }
348     QMetaObject::invokeMethod(
349         plot,
350         [=]() {
351             //         auto guiStart = high_resolution_clock::now()
;
352
353             plot->setUpdatesEnabled(false);
354
355             plot->updateSeries(series.outcomeS, deltaQData);
356             plot->updateSeries(series.lowerBoundS, lowerBoundData)
;

```

```

357         plot->updateSeries(series.upperBoundS, upperBoundData)
358     ;
359     plot->updateSeries(series.meanS, meanData);
360     plot->updateSeries(series.qtaS, qtaData);
361
362     plot->setUpdatesEnabled(true);
363
364     //      auto guiEnd = high_resolution_clock::now();
365     //      qDebug() << "GUI update took" << duration_cast
366     <microseconds>(guiEnd - guiStart).count() << "";
367     },
368     Qt::QueuedConnection);
369 });
370 ret.waitForFinished();
371 return outcome->getMaxDelay();
372 }
373
374 double DQPlotController::updateOperator(ExpressionSeries &series, std
375 ::shared_ptr<Operator> &op, uint64_t timeLowerBound, uint64_t
376 timeUpperBound)
377 {
378     updateExpression(series, op->getObservedDeltaQRepr(timeLowerBound,
379 timeUpperBound), op->getCalculatedDeltaQRepr(timeLowerBound,
380 timeUpperBound),
381     op->getQTA(), op->getMaxDelay());
382
383     return op->getMaxDelay();
384 }
385
386 double DQPlotController::updateProbe(ExpressionSeries &series, std::
387 shared_ptr<Probe> &probe, uint64_t timeLowerBound, uint64_t
388 timeUpperBound)
389 {
390     updateExpression(series, probe->getObservedDeltaQRepr(
391 timeLowerBound, timeUpperBound), probe->getCalculatedDeltaQRepr(
392 timeLowerBound, timeUpperBound),
393     probe->getQTA(), probe->getMaxDelay());
394     return probe->getMaxDelay();
395 }
396
397 void DQPlotController::updateExpression(ExpressionSeries &series,
398 DeltaQRepr &&obsRepr, DeltaQRepr &&calcRepr, QTA &&qta, double
399 maxDelay)
400 {
401     auto ret = QtConcurrent::run( [=]() {
402         auto computeStart = high_resolution_clock::now();
403
404         DeltaQ obsDeltaQ = obsRepr.deltaQ;
405         std::vector<Bound> obsBounds = obsRepr.bounds;
406         DeltaQ calcDeltaQ = calcRepr.deltaQ;
407         std::vector<Bound> calcBounds = calcRepr.bounds;
408         int observedBins = obsDeltaQ.getBins();
409         double observedBinWidth = obsDeltaQ.getBinWidth();
410
411         // --- Prepare data ---
412         QVector<QPointF> obsDeltaQData;

```



```

401     obsDeltaQData.emplace_back(QPointF(0, 0));
402     obsDeltaQData.reserve(observedBins);
403
404     QVector<QPointF> obsLowerBoundData;
405     obsLowerBoundData.emplace_back(QPointF(0, 0));
406     obsLowerBoundData.reserve(observedBins);
407
408     QVector<QPointF> obsUpperBoundData;
409     obsUpperBoundData.emplace_back(QPointF(0, 0));
410     obsUpperBoundData.reserve(observedBins);
411
412     QVector<QPointF> obsMeanData;
413     obsMeanData.emplace_back(QPointF(0, 0));
414     obsMeanData.reserve(observedBins);
415
416     for (int i = 0; i < observedBins; ++i) {
417         double x = observedBinWidth * (i + 1);
418         obsDeltaQData.emplace_back(QPointF(x, obsDeltaQ.cdfAt(i)))
419     };
420     obsLowerBoundData.emplace_back(QPointF(x, obsBounds[i].
lowerBound));
421     obsUpperBoundData.emplace_back(QPointF(x, obsBounds[i].
upperBound));
422     obsMeanData.emplace_back(QPointF(x, obsBounds[i].mean));
423 }
424
425 QVector<QPointF> calcDeltaQData;
426 calcDeltaQData.emplace_back(QPointF(0, 0));
427 calcDeltaQData.reserve(observedBins);
428
429 QVector<QPointF> calcLowerBoundData;
430 calcLowerBoundData.emplace_back(QPointF(0, 0));
431 calcLowerBoundData.reserve(observedBins);
432
433 QVector<QPointF> calcUpperBoundData;
434 calcUpperBoundData.emplace_back(QPointF(0, 0));
435 calcUpperBoundData.reserve(observedBins);
436
437 QVector<QPointF> calcMeanData;
438 calcMeanData.emplace_back(QPointF(0, 0));
439 calcMeanData.reserve(observedBins);
440
441 QVector<QPointF> qtaData;
442
443 // Prepare calculatedDeltaQ data
444 int calculatedBins = calcDeltaQ.getBins();
445 double calculatedBinWidth = calcDeltaQ.getBinWidth();
446 for (int i = 0; i < calculatedBins; ++i) {
447     double x = calculatedBinWidth * (i + 1);
448     calcDeltaQData.emplace_back(QPointF(x, calcDeltaQ.cdfAt(i)
449 ));
450     calcLowerBoundData.emplace_back(QPointF(x, calcBounds[i].
lowerBound));
451     calcUpperBoundData.emplace_back(QPointF(x, calcBounds[i].
upperBound));
452     calcMeanData.emplace_back(QPointF(x, calcBounds[i].mean));

```

```

451     }
452     if (qta.defined) {
453         qtaData.reserve(8);
454         qtaData.emplace_back(qta.perc_25, 0);
455         qtaData.emplace_back(qta.perc_25, 0.25);
456         qtaData.emplace_back(qta.perc_50, 0.25);
457         qtaData.emplace_back(qta.perc_50, 0.5);
458         qtaData.emplace_back(qta.perc_75, 0.5);
459         qtaData.emplace_back(qta.perc_75, 0.75);
460         qtaData.emplace_back(maxDelay, 0.75);
461         qtaData.emplace_back(maxDelay, qta.cdfMax);
462     }
463     auto computeEnd = high_resolution_clock::now();
464     // std::cout << "comp," << observedBins << "," <<
calculatedBins << "," << duration_cast<microseconds>(computeEnd -
computeStart).count();
465
466     // --- Push results back to GUI thread ---
467
468     QMetaObject::invokeMethod(
469         plot,
470         [=]() {
471             auto guiStart = high_resolution_clock::now();
472
473             plot->setUpdatesEnabled(false);
474
475             plot->updateSeries(series.obsS, obsDeltaQData);
476             plot->updateSeries(series.obsLowerBoundS,
obsLowerBoundData);
477             plot->updateSeries(series.obsUpperBoundS,
obsUpperBoundData);
478             plot->updateSeries(series.obsMeanS, obsMeanData);
479
480             plot->updateSeries(series.calcS, calcDeltaQData);
481             plot->updateSeries(series.calcLowerBoundS,
calcLowerBoundData);
482             plot->updateSeries(series.calcUpperBoundS,
calcUpperBoundData);
483             plot->updateSeries(series.calcMeanS, calcMeanData);
484             plot->updateSeries(series.qtaS, qtaData);
485
486             plot->setUpdatesEnabled(true);
487
488             auto guiEnd = high_resolution_clock::now();
489             // std::cout << "gui," << observedBins << "," <<
calculatedBins << "," << duration_cast<microseconds>(guiEnd -
guiStart).count();
490         },
491         Qt::QueuedConnection);
492     });
493     ret.waitForFinished();
494 }

```

C.2.5 DQPlotList.cpp

```

1 #include "DQPlotList.h"
2 #include "../Application.h"
3 #include <QLabel>
4 #include <QListWidgetItem>
5 #include <qlistwidget.h>
6 DQPlotList::DQPlotList(DQPlotController *controller, QWidget *parent)
7     : QWidget(parent)
8     , controller(controller)
9 {
10     QVBoxLayout *layout = new QVBoxLayout(this);
11
12     // Selected items list
13     QLabel *selectedLabel = new QLabel("Selected probes:");
14     selectedList = new QListWidget(this);
15     layout->addWidget(selectedLabel);
16     layout->addWidget(selectedList);
17     selectedList->setSelectionMode(QAbstractItemView::MultiSelection);
18     removeButton = new QPushButton("Remove selected probes", this);
19     layout->addWidget(removeButton);
20
21     connect(removeButton, &QPushButton::clicked, this, &DQPlotList::
onRemoveSelection);
22     // Available items list
23     QLabel *availableLabel = new QLabel("Available probes:");
24     availableList = new QListWidget(this);
25     layout->addWidget(availableLabel);
26     layout->addWidget(availableList);
27     availableList->setSelectionMode(QAbstractItemView::MultiSelection)
;
28     // Confirm button
29     addButton = new QPushButton("Add selected probes", this);
30     connect(addButton, &QPushButton::clicked, this, &DQPlotList::
onConfirmSelection);
31     layout->addWidget(addButton);
32
33     updateLists();
34 }
35
36 bool DQPlotList::isEmptyAfterReset()
37 {
38     return controller->isEmptyAfterReset();
39 }
40
41
42 void DQPlotList::updateLists()
43 {
44     availableList->clear();
45     selectedList->clear();
46
47     auto plotComponents = controller->getComponents();
48     auto system = Application::getInstance().getSystem();
49     // Add components selected in a DeltaQPlot
50     for (auto &component : plotComponents) {
51         new QListWidgetItem(QString::fromStdString(component),
selectedList);

```

```

52     }
53
54     // Select all components that are available to be chosen to add
55     auto allComponents = system->getAllComponentsName();
56
57     auto pred
58         = [&plotComponents](const std::string &key) -> bool { return
std::find(plotComponents.begin(), plotComponents.end(), key) !=
plotComponents.end(); };
59
60     allComponents.erase(std::remove_if(allComponents.begin(),
allComponents.end(), pred), allComponents.end());
61     for (auto &component : allComponents) {
62         new QListWidgetItem(QString::fromStdString(component),
availableList);
63     }
64 }
65
66 void DQPlotList::onConfirmSelection()
67 {
68     QList<QListWidgetItem *> selected = availableList->selectedItems()
;
69     auto system = Application::getInstance().getSystem();
70     for (QListWidgetItem *item : selected) {
71         controller->addComponent(item->text().toStdString(), system->
hasOutcome(item->text().toStdString()));
72     }
73     updateLists();
74 }
75
76 void DQPlotList::onRemoveSelection()
77 {
78     QList<QListWidgetItem *> selected = selectedList->selectedItems();
79
80     for (QListWidgetItem *item : selected) {
81         controller->removeComponent(item->text().toStdString());
82     }
83     updateLists();
84 }

```

C.2.6 DelaySettingsWidget.cpp

```

1  /**
2   * @file DelaySettingsWidget.cpp
3   * @brief Implementation of the DelaySettingsWidget class.
4   */
5
6  #include "DelaySettingsWidget.h"
7  #include "../Application.h"
8  #include <qlabel.h>
9  #include <qpushbutton.h>
10
11 DelaySettingsWidget::DelaySettingsWidget(QWidget *parent)
12     : QWidget(parent)
13 {

```

```

14     mainLayout = new QVBoxLayout(this);
15     settingsLayout = new QHBoxLayout(this);
16
17     settingsLabel = new QLabel("Set the parameters for a probe.");
18     mainLayout->addWidget(settingsLabel);
19
20     observableComboBox = new QComboBox();
21     settingsLayout->addWidget(observableComboBox);
22
23     delaySlider = new QSlider(Qt::Horizontal);
24     delaySlider->setRange(-10, 10);
25     delaySlider->setTickInterval(1);
26     delaySlider->setTickPosition(QSlider::TicksBelow);
27     settingsLayout->addWidget(delaySlider);
28
29     binSpinBox = new QSpinBox();
30     binSpinBox->setRange(1, 1000);
31     binSpinBox->setValue(10);
32     settingsLayout->addWidget(binSpinBox);
33
34     mainLayout->addLayout(settingsLayout);
35
36     maxDelayLabel = new QLabel("Max delay is: ");
37     mainLayout->addWidget(maxDelayLabel);
38
39     saveDelayButton = new QPushButton("Save delay");
40     mainLayout->addWidget(saveDelayButton);
41
42     connect(delaySlider, &QSlider::valueChanged, this, &
DelaySettingsWidget::updateMaxDelay);
43     connect(binSpinBox, QOverload<int>::of(&QSpinBox::valueChanged),
this, &DelaySettingsWidget::updateMaxDelay);
44     connect(saveDelayButton, &QPushButton::clicked, this, &
DelaySettingsWidget::onSaveDelayClicked);
45     connect(observableComboBox, &QComboBox::currentTextChanged, this,
&DelaySettingsWidget::loadObservableSettings);
46
47     Application::getInstance().addObserver([this]() { this->
populateComboBox(); });
48 }
49
50 /**
51  * @brief Populates the combo box with available observables from the
52  * system.
53  */
54 void DelaySettingsWidget::populateComboBox()
55 {
56     auto system = Application::getInstance().getSystem();
57     if (!system)
58         return;
59
60     observableComboBox->clear();
61     for (const auto &[name, obs] : system->getObservables()) {
62         if (obs)
63             observableComboBox->addItem(QString::fromStdString(name));
64     }
65 }

```

```

64 }
65
66 /**
67  * @brief Loads delay settings for the currently selected observable.
68  */
69 void DelaySettingsWidget::loadObservableSettings()
70 {
71     auto system = Application::getInstance().getSystem();
72     if (!system)
73         return;
74
75     QString observableName = observableComboBox->currentText();
76     if (observableName.isEmpty())
77         return;
78
79     auto observable = system->getObservable(observableName.toStdString());
80
81     auto exponent = observable->getDeltaTExp();
82     auto bins = observable->getNBins();
83     delaySlider->setValue(exponent);
84     binSpinBox->setValue(bins);
85
86     updateMaxDelay();
87 }
88
89 /**
90  * @brief Computes the current maximum delay.
91  * @return Maximum delay value based on bin count and delay exponent.
92  */
93 double DelaySettingsWidget::getMaxDelayMs() const
94 {
95     int exponent = delaySlider->value();
96     int bins = binSpinBox->value();
97     return 1.0 * std::pow(2.0, exponent) * bins;
98 }
99
100 /**
101  * @brief Updates the label that shows the computed max delay value.
102  */
103 void DelaySettingsWidget::updateMaxDelay()
104 {
105     double delay = getMaxDelayMs();
106     maxDelayLabel->setText(QString("Max delay is: %1 ms").arg(delay,
107     0, 'f', 2));
108 }
109
110 /**
111  * @brief Saves the current delay settings to the system and emits a
112  * change signal.
113  */
114 void DelaySettingsWidget::onSaveDelayClicked()
115 {
116     auto system = Application::getInstance().getSystem();
117     if (!system)
118         return;

```

```

117
118     QString name = observableComboBox->currentText();
119     if (name.isEmpty())
120         return;
121
122     int exponent = delaySlider->value();
123     int bins = binSpinBox->value();
124     std::string nameString = name.toStdString();
125     system->setObservableParameters(nameString, exponent, bins);
126
127     Q_EMIT delayParametersChanged();
128 }

```

C.2.7 DeltaQPlot.cpp

```

1
2 #include "DeltaQPlot.h"
3 #include "ColorRegistry.h"
4 #include "CustomLegendPanel.h"
5 #include "DQPlotList.h"
6 #include <QChartView>
7 #include <QDebug>
8 #include <QHBoxLayout>
9 #include <QLineSeries>
10 #include <QMouseEvent>
11 #include <QRandomGenerator>
12 #include <QToolButton>
13 #include <QVBoxLayout>
14 DeltaQPlot::DeltaQPlot(const std::vector<std::string> &selectedItems,
15                        QWidget *parent)
16     : QWidget(parent)
17 {
18     // Create legend panel and toggle
19     legendPanel = new CustomLegendPanel(this);
20     QToolButton *toggleButton = new QToolButton(this);
21     toggleButton->setText("> Legend");
22     toggleButton->setCheckable(true);
23     toggleButton->setChecked(true);
24     toggleButton->setToolButtonStyle(Qt::ToolButtonTextBesideIcon);
25
26     connect(toggleButton, &QToolButton::toggled, legendPanel, &QWidget::setVisible);
27     connect(toggleButton, &QToolButton::toggled, [toggleButton](bool checked) { toggleButton->setText(checked ? "^ Legend" : "> Legend"); });
28
29     // Create chart and view
30     chart = new QChart();
31     chartView = new QChartView(chart, this);
32     chartView->setRenderHint(QPainter::Antialiasing);
33     chart->legend()->setVisible(false);
34
35     axisX = new QValueAxis();
36     axisY = new QValueAxis();

```

```
37     axisY->setRange(-0.01, 1.0);
38     axisX->setRange(0, 0.05);
39     chart->addAxis(axisX, Qt::AlignBottom);
40     chart->addAxis(axisY, Qt::AlignLeft);
41
42     controller = new DQPlotController(this, selectedItems);
43     plotList = new DQPlotList(controller, this);
44     // Right-side layout: toggle + legend
45     QVBoxLayout *rightLayout = new QVBoxLayout();
46     rightLayout->addWidget(toggleButton);
47     rightLayout->addWidget(legendPanel);
48     rightLayout->addStretch();
49
50     // Main layout: chart + right side
51     QHBoxLayout *mainLayout = new QHBoxLayout(this);
52     mainLayout->addWidget(chartView, 1);
53     mainLayout->addLayout(rightLayout);
54     mainLayout->setContentsMargins(0, 0, 0, 0);
55     mainLayout->setSpacing(5);
56     setLayout(mainLayout);
57     chartView->setRenderHint(QPainter::Antialiasing);
58 }
59
60 DeltaQPlot::~DeltaQPlot()
61 {
62     delete controller;
63     controller = nullptr;
64     delete plotList;
65     plotList = nullptr;
66 }
67
68 bool DeltaQPlot::isEmptyAfterReset()
69 {
70     if (!controller->isEmptyAfterReset()) {
71         plotList->updateLists();
72         return false;
73     }
74     return true;
75 }
76
77 void DeltaQPlot::setTitle(QString &&title)
78 {
79     chart->setTitle(title);
80 }
81
82 void DeltaQPlot::addSeries(QLineSeries *series, const std::string &
83     name)
84 {
85     chart->addSeries(series);
86     series->setName(QString::fromStdString(name));
87     series->attachAxis(axisX);
88     series->attachAxis(axisY);
89     QColor color = ColorRegistry::getColorFor(name);
90     legendPanel->addEntry(QString::fromStdString(name), color);
91     series->setColor(color);
92     series->setVisible(true);
```



```

92 }
93
94 void DeltaQPlot::update(uint64_t timeLowerBound, uint64_t
    timeUpperBound)
95 {
96     controller->update(timeLowerBound, timeUpperBound);
97 }
98
99 void DeltaQPlot::removeSeries(QAbstractSeries *series)
100 {
101     chart->removeSeries(series);
102     auto name = series->name();
103     legendPanel->removeEntry(name);
104 }
105
106 void DeltaQPlot::editPlot(const std::vector<std::string> &
    selectedItems)
107 {
108     controller->editPlot(selectedItems);
109 }
110
111 void DeltaQPlot::updateSeries(QLineSeries *series, const QVector<
    QPointF> &data)
112 {
113     series->replace(data);
114 }
115
116 void DeltaQPlot::updateXRange(double xRange)
117 {
118     axisX->setRange(0, xRange);
119 }
120
121 std::vector<std::string> DeltaQPlot::getComponents()
122 {
123     return controller->getComponents();
124 }
125
126 DQPlotList *DeltaQPlot::getPlotList()
127 {
128     return plotList;
129 }
130
131 void DeltaQPlot::mousePressEvent(QMouseEvent *event)
132 {
133     Q_EMIT plotSelected(this);
134 }

```

C.2.8 MainWindow.cpp

```

1 #include "MainWindow.h"
2 #include "../Application.h"
3 #include "../maths/DeltaQOperations.h"
4 #include "DQPlotList.h"
5 #include "DeltaQPlot.h"
6 #include "NewPlotList.h"

```

```

7  #include "ObservableSettings.h"
8  #include "Sidebar.h"
9  #include <QMenu>
10 #include <QMessageBox>
11 #include <QThread>
12 #include <QVBoxLayout>
13 #define MAX_P_ROW 2
14 #define MAX_P_COL 2
15
16 MainWindow::MainWindow(QWidget *parent)
17     : QMainWindow(parent)
18 {
19     // Set up central widget and main layout
20     centralWidget = new QWidget(this);
21     setCentralWidget(centralWidget);
22     mainLayout = new QHBoxLayout(centralWidget);
23
24     // Configure scroll area for plots
25     scrollArea = new QScrollArea(this);
26     scrollArea->setWidgetResizable(true);
27     scrollArea->setHorizontalScrollBarPolicy(Qt::ScrollBarAsNeeded);
28     scrollArea->setVerticalScrollBarPolicy(Qt::ScrollBarAsNeeded);
29     scrollArea->setBaseSize(800, 800);
30
31     // Set up plot container with grid layout
32     plotContainer = new QWidget();
33     plotLayout = new QGridLayout(plotContainer);
34     scrollArea->setWidget(plotContainer);
35     mainLayout->addWidget(scrollArea, 1);
36
37     // Initialize side panels
38     sidebar = new Sidebar(this);
39     triggersTab = new TriggersTab(this);
40     observableSettings = new ObservableSettings(this);
41     stubWidget = new StubControlWidget(this);
42
43     // Configure tabbed side panel
44     sideTabWidget = new QTabWidget(this);
45     sideTabWidget->addTab(sidebar, "System/Plots");
46     sideTabWidget->addTab(observableSettings, "Probes settings");
47     sideTabWidget->addTab(triggersTab, "Triggers");
48     sideTabWidget->addTab(stubWidget, "Connection controls");
49     sideTabWidget->setTabPosition(QTabWidget::West);
50     // Connect sidebar signals
51     connect(sidebar, &Sidebar::addPlotClicked, this, &MainWindow::
onAddPlotClicked);
52
53     // Set up side container layout
54     sideContainer = new QWidget(this);
55     sideLayout = new QVBoxLayout(sideContainer);
56     sideLayout->addWidget(sideTabWidget);
57     sideLayout->setStretch(1, 0); // Make tabs take up most space
58     mainLayout->addWidget(sideContainer, 0);
59
60     // Set up update timer in separate thread
61     timerThread = new QThread(this);

```

```

62     updateTimer = new QTimer();
63     updateTimer->moveToThread(timerThread);
64     connect(updateTimer, &QTimer::timeout, this, &MainWindow::
updatePlots, Qt::QueuedConnection);
65     connect(timerThread, &QThread::started, [this]() { updateTimer->
start(200); });
66     timerThread->start();
67
68     // Register system reset observer
69     Application::getInstance().addObserver([this] { this->reset(); });
70
71     // Initialize time bounds
72     auto now = std::chrono::system_clock::now();
73     auto adjustedTime = now - std::chrono::milliseconds(200);
74     timeLowerBound = std::chrono::duration_cast<std::chrono::
nanoseconds>(adjustedTime.time_since_epoch()).count();
75
76     // Connect polling rate changes
77     connect(sidebar, &Sidebar::onPollingRateChanged, this, [this](int
ms) {
78         qDebug() << "MainWindow received polling rate:" << ms;
79         pollingRate = ms;
80         QMetaObject::invokeMethod(updateTimer, [ms, this]() {
updateTimer->setInterval(ms); }, Qt::QueuedConnection);
81     });
82 }
83
84 /**
85  * @brief Cleans up empty plots during reset.
86  */
87 void MainWindow::reset()
88 {
89     std::lock_guard<std::mutex> lock(plotDelMutex);
90     auto it = plotContainers.begin();
91     while (it != plotContainers.end()) {
92         DeltaQPlot *plot = it.key();
93         QWidget *plotWidget = it.value();
94
95         if (plot->isEmptyAfterReset()) {
96             it = plotContainers.erase(it);
97             delete plot;
98             if (plotWidget) {
99                 delete plotWidget;
100             }
101             sidebar->hideCurrentPlot();
102         } else {
103             ++it;
104         }
105     }
106 }
107
108 /**
109  * @brief Updates all plots with new data from the system.
110  */
111 void MainWindow::updatePlots()
112 {

```

```

113     // Update time bounds
114     timeLowerBound += std::chrono::duration_cast<std::chrono::
nanoseconds>(std::chrono::milliseconds(pollingRate)).count();
115     uint64_t timeUpperBound = timeLowerBound + std::chrono::
duration_cast<std::chrono::nanoseconds>(std::chrono::milliseconds(
pollingRate)).count();
116
117     auto system = Application::getInstance().getSystem();
118     std::lock_guard<std::mutex> lock(plotDelMutex);
119
120     // Update probes
121     for (auto &[name, probe] : system->getProbes()) {
122         if (probe) {
123             probe->getObservedDeltaQ(timeLowerBound, timeUpperBound);
124             probe->calculateCalculatedDeltaQ(timeLowerBound,
timeUpperBound);
125         }
126     }
127
128     // Update operators
129     for (auto &[name, op] : system->getOperators()) {
130         if (op) {
131             op->getObservedDeltaQ(timeLowerBound, timeUpperBound);
132             op->calculateCalculatedDeltaQ(timeLowerBound,
timeUpperBound);
133         }
134     }
135
136     // Update outcomes
137     for (auto &[name, outcome] : system->getOutcomes()) {
138         if (outcome) {
139             outcome->getObservedDeltaQ(timeLowerBound, timeUpperBound)
;
140         }
141     }
142
143     // Update all plots
144     for (auto [plot, _] : plotContainers.asKeyValueRange()) {
145         plot->update(timeLowerBound, timeUpperBound);
146     }
147 }
148
149 /**
150  * @brief Destructor cleans up timer thread and resources.
151  */
152 MainWindow::~MainWindow()
153 {
154     if (timerThread) {
155         timerThread->quit();
156         timerThread->wait();
157         delete timerThread;
158     }
159 }
160
161 /**
162  * @brief Adds a new plot based on sidebar selection.

```

```

163  */
164  void MainWindow::onAddPlotClicked()
165  {
166      auto selectedItems = sidebar->getPlotList()->getSelectedItems();
167
168      if (selectedItems.empty()) {
169          QMessageBox::warning(this, "No Selection", "Please select
170 components before adding a plot.");
171          return;
172      }
173
174      // Create new plot container
175      auto *plotWidget = new QWidget(this);
176      auto *plotWidgetLayout = new QVBoxLayout(plotWidget);
177      auto *deltaQPlot = new DeltaQPlot(selectedItems, this);
178
179      // Set up connections and tracking
180      connect(deltaQPlot, &DeltaQPlot::plotSelected, this, &MainWindow::
181 onPlotSelected);
182      plotContainers[deltaQPlot] = plotWidget;
183
184      // Configure layout
185      plotWidgetLayout->addWidget(deltaQPlot);
186      plotWidget->setMaximumWidth(scrollArea->width() / MAX_P_ROW);
187      plotWidget->setMaximumHeight(scrollArea->height() / 2);
188
189      // Position in grid
190      int plotCount = plotContainers.size();
191      int row = (plotCount - 1) / MAX_P_ROW;
192      int col = (plotCount - 1) % MAX_P_COL;
193      plotLayout->addWidget(plotWidget, row, col);
194
195      // Update UI state
196      onPlotSelected(deltaQPlot);
197      sidebar->clearOnAdd();
198  }
199
200  /**
201   * @brief Handles window resize events to adjust plot sizes.
202   * @param event The resize event.
203   */
204  void MainWindow::resizeEvent(QResizeEvent *event)
205  {
206      QMainWindow::resizeEvent(event);
207      for (auto [plot, widget] : plotContainers.asKeyValueRange()) {
208          widget->setMaximumWidth(scrollArea->width() / MAX_P_COL);
209          widget->setMaximumHeight(scrollArea->height() / 2);
210      }
211  }
212
213  /**
214   * @brief Shows context menu for plot management.
215   * @param event The context menu event.
216   */
217  void MainWindow::contextMenuEvent(QContextMenuEvent *event)
218  {

```

```

217     QWidget *child = childAt(event->pos());
218     if (!child)
219         return;
220
221     // Find which plot was right-clicked
222     DeltaQPlot *selectedPlot = nullptr;
223     for (auto it = plotContainers.begin(); it != plotContainers.end();
224         ++it) {
225         if (it.value()->isAncestorOf(child)) {
226             selectedPlot = it.key();
227             break;
228         }
229     }
230
231     if (!selectedPlot)
232         return;
233
234     // Create and show context menu
235     QMenu contextMenu(this);
236     QAction *removeAction = contextMenu.addAction("Remove Plot");
237
238     QAction *selectedAction = contextMenu.exec(event->globalPos());
239
240     if (selectedAction == removeAction) {
241         onRemovePlot(selectedPlot);
242     }
243
244     /**
245     * @brief Removes a plot and cleans up resources.
246     * @param plot The plot to remove.
247     */
248     void MainWindow::onRemovePlot(DeltaQPlot *plot)
249     {
250         QWidget *plotWidget = plotContainers.value(plot, nullptr);
251         if (plotWidget) {
252             plotLayout->removeWidget(plotWidget);
253             plotWidget->deleteLater();
254         }
255         plotContainers.remove(plot);
256
257         // Reorganize remaining plots
258         int plotCount = 0;
259         for (auto it = plotContainers.begin(); it != plotContainers.end();
260             ++it) {
261             int row = plotCount / MAX_P_ROW;
262             int col = plotCount % MAX_P_COL;
263             plotLayout->addWidget(it.value(), row, col);
264             plotLayout->setRowStretch(row, 1);
265             plotLayout->setColumnStretch(col, 1);
266             plotCount++;
267         }
268
269         sidebar->hideCurrentPlot();
270     }

```

```
271 /**
272  * @brief Updates sidebar when a plot is selected.
273  * @param plot The newly selected plot.
274  */
275 void MainWindow::onPlotSelected(DeltaQPlot *plot)
276 {
277     if (!plot)
278         return;
279     DQPlotList *plotList = plot->getPlotList();
280     if (!plotList) {
281         return;
282     }
283     sidebar->setCurrentPlotList(plotList);
284 }
```

C.2.9 NewPlotList.cpp

```
1 /**
2  * @file NewPlotList.cpp
3  * @brief Implementation of the NewPlotList class, which provides a UI
4  *       list for selecting observables to create a new plot.
5  */
6 #include "NewPlotList.h"
7 #include "../Application.h"
8 #include <iostream>
9 #include <qlistwidget.h>
10
11 NewPlotList::NewPlotList(QWidget *parent)
12     : QListWidget(parent)
13 {
14     setSelectionMode(QAbstractItemView::MultiSelection);
15     Application::getInstance().addObserver([this]() { this->reset();
16     });
17 }
18 /**
19  * @brief Clears and repopulates the list with updated observables.
20  *
21  */
22 void NewPlotList::reset()
23 {
24     this->blockSignals(true);
25     while (count() != 0)
26         delete takeItem(0);
27     this->blockSignals(false);
28     addItem();
29 }
30
31 /**
32  * @brief Adds all observables from the system as items in the list.
33  */
34 void NewPlotList::addItem()
35 {
36     auto system = Application::getInstance().getSystem();
```

```

37     auto observables = system->getObservables();
38
39     for (const auto &obs : observables) {
40         if (obs.second) {
41             QListWidgetItem *item = new QListWidgetItem(QString::
fromStdString(obs.first), this);
42             item->setFlags(Qt::ItemIsSelectable | Qt::ItemIsEnabled);
43         }
44     }
45 }
46
47 /**
48  * @brief Returns a list of selected observable names.
49  * @return A vector of strings corresponding to selected item labels.
50  */
51 std::vector<std::string> NewPlotList::getSelectedItems()
52 {
53     std::vector<std::string> selectedItems;
54     QList<QListWidgetItem *> selected = this->selectedItems();
55
56     for (QListWidgetItem *item : selected) {
57         selectedItems.push_back(item->text().toStdString());
58     }
59     return selectedItems;
60 }
61
62 /**
63  * @brief Deselects all currently selected items in the list.
64  */
65 void NewPlotList::deselectAll()
66 {
67     clearSelection();
68 }

```

C.2.10 ObservableSettings.cpp

```

1  #include "ObservableSettings.h"
2  #include <qlabel.h>
3  #include <qwidget.h>
4
5  ObservableSettings::ObservableSettings(QWidget *parent)
6      : QWidget(parent)
7  {
8      layout = new QVBoxLayout(this);
9      layout->setAlignment(Qt::AlignTop);
10     layout->setSpacing(10);
11     layout->setContentsMargins(10, 10, 10, 10);
12
13     qtaInputWidget = new QTAInputWidget(this);
14     layout->addWidget(qtaInputWidget);
15
16     delaySettingsWidget = new DelaySettingsWidget(this);
17     layout->addWidget(delaySettingsWidget);
18     connect(delaySettingsWidget, &DelaySettingsWidget::
delayParametersChanged, qtaInputWidget, &QTAInputWidget::

```



```

19 }
    loadObservableSettings);

```

C.2.11 PollingRateWidget.cpp

```

1 #include "PollingRateWidget.h"
2
3 PollingRateWidget::PollingRateWidget(QWidget *parent)
4     : QWidget(parent)
5 {
6     // Available polling rate options in milliseconds
7     pollingRates = {100, 200, 300, 400, 500, 600, 700, 800, 900,
8                     1000, 2000, 5000, 10000};
9
10    // Configure rate selection slider
11    slider = new QSlider(Qt::Horizontal);
12    slider->setMinimum(0);
13    slider->setMaximum(pollingRates.size() - 1);
14    slider->setValue(1);
15    slider->setTickInterval(1);
16    slider->setTickPosition(QSlider::TicksBelow);
17
18    valueLabel = new QLabel(QString("Polling Rate: %1 ms").arg(
19        pollingRates[0]));
20
21    saveButton = new QPushButton("Save polling rate");
22
23    // Set up layout
24    auto *layout = new QVBoxLayout();
25    layout->addWidget(slider);
26    layout->addWidget(valueLabel);
27    layout->addWidget(saveButton);
28    setLayout(layout);
29
30    // Connect signals
31    connect(slider, &QSlider::valueChanged,
32        this, &PollingRateWidget::onSliderValueChanged);
33    connect(saveButton, &QPushButton::clicked,
34        this, &PollingRateWidget::onSaveClicked);
35
36    /**
37     * @brief Updates the displayed rate when slider changes.
38     * @param value The current slider index (0-based).
39     *
40     * Converts slider index to actual milliseconds and updates the
41     * display label.
42     */
43    void PollingRateWidget::onSliderValueChanged(int value)
44    {
45        int ms = pollingRates[value];
46        valueLabel->setText(QString("Polling Rate: %1 ms").arg(ms));
47    }
48    /**

```

```

49  * @brief Emits the selected polling rate when save is clicked.
50  *
51  * Gets the current slider value, converts it to milliseconds,
52  * and emits the onPollingRateChanged signal.
53  */
54 void PollingRateWidget::onSaveClicked()
55 {
56     int ms = pollingRates[slider->value()];
57     Q_EMIT onPollingRateChanged(ms);
58 }

```

C.2.12 QTAInputWidget.cpp

```

1  #include "QTAInputWidget.h"
2  #include "../Application.h"
3  #include <QDoubleValidator>
4  #include <QMessageBox>
5  #include <QPushButton>
6
7  QTAInputWidget::QTAInputWidget(QWidget *parent)
8      : QWidget(parent)
9  {
10     auto layout = new QFormLayout(this);
11
12     qtaLabel = new QLabel(this);
13     qtaLabel->setText("Set QTA for a probe:");
14     layout->addRow(qtaLabel);
15     observableComboBox = new QComboBox(this);
16     layout->addRow("Probe:", observableComboBox);
17
18     perc25Edit = new QLineEdit(this);
19     perc25Edit->setPlaceholderText("Seconds (s)");
20
21     perc50Edit = new QLineEdit(this);
22     perc50Edit->setPlaceholderText("Seconds (s)");
23
24     perc75Edit = new QLineEdit(this);
25     perc75Edit->setPlaceholderText("Seconds (s)");
26
27     cdfMaxEdit = new QLineEdit(this);
28     cdfMaxEdit->setPlaceholderText("Value between 0 and 1");
29
30     layout->addRow("25th Percentile (s):", perc25Edit);
31     layout->addRow("50th Percentile (s):", perc50Edit);
32     layout->addRow("75th Percentile (s):", perc75Edit);
33     layout->addRow("Max. allowed failure (0-1):", cdfMaxEdit);
34
35     connect(observableComboBox, &QComboBox::currentTextChanged, this,
36         &QTAInputWidget::loadObservableSettings);
37
38     Application::getInstance().addObserver([this]() { this->
39         populateComboBox(); });
40     saveButton = new QPushButton("Save QTA Settings", this);
41     layout->addRow(saveButton);

```

```

40     connect(saveButton, &QPushButton::clicked, this, &QTAInputWidget::
onSaveButtonClicked);
41
42     setLayout(layout);
43
44     Application::getInstance().addObserver([this]() { this->
populateComboBox(); });
45 }
46
47 void QTAInputWidget::populateComboBox()
48 {
49     auto system = Application::getInstance().getSystem();
50     if (!system)
51         return;
52
53     observableComboBox->clear();
54     for (const auto &[name, _] : system->getProbes()) {
55         observableComboBox->addItem(QString::fromStdString(name));
56     }
57     for (const auto &[name, _] : system->getOutcomes()) {
58         observableComboBox->addItem(QString::fromStdString(name));
59     }
60 }
61
62 void QTAInputWidget::loadObservableSettings()
63 {
64     auto system = Application::getInstance().getSystem();
65     if (!system)
66         return;
67     std::string observableName = observableComboBox->currentText().
toStdString();
68     auto observable = system->getObservable(observableName);
69     if (observable) {
70         auto qta = observable->getQTA();
71         perc25Edit->setText(QString::number(qta.perc_25, 'f', 6)); //
6 decimal places
72         perc50Edit->setText(QString::number(qta.perc_50, 'f', 6));
73         perc75Edit->setText(QString::number(qta.perc_75, 'f', 6));
74         cdfMaxEdit->setText(QString::number(qta.cdfMax, 'f', 6));
75     }
76 }
77
78 void QTAInputWidget::onSaveButtonClicked()
79 {
80     auto system = Application::getInstance().getSystem();
81     if (!system)
82         return;
83
84     std::string observableName = observableComboBox->currentText().
toStdString();
85     auto observable = system->getObservable(observableName);
86     if (!observable)
87         return;
88
89     try {

```

```

90     QTA newQTA = QTA::create(getPerc25(), getPerc50(), getPerc75()
91     , getCdfMax(), true);
92     observable->setQTA(newQTA);
93 } catch (std::exception &e) {
94     QMessageBox::warning(this, "Error", e.what());
95 }
96
97 double QTAInputWidget::getPerc25() const
98 {
99     return perc25Edit->text().toDouble();
100 }
101 double QTAInputWidget::getPerc50() const
102 {
103     return perc50Edit->text().toDouble();
104 }
105 double QTAInputWidget::getPerc75() const
106 {
107     return perc75Edit->text().toDouble();
108 }
109 double QTAInputWidget::getCdfMax() const
110 {
111     return cdfMaxEdit->text().toDouble();
112 }
113
114 QString QTAInputWidget::getSelectedObservable() const
115 {
116     return observableComboBox->currentText();
117 }

```

C.2.13 Sidebar.cpp

```

1  #include "Sidebar.h"
2
3  #include "NewPlotList.h"
4  #include "PollingRateWidget.h"
5  #include "SystemCreationWidget.h"
6  #include <QBoxLayout>
7  #include <QFileDialog>
8  #include <QLabel>
9  #include <QMessageBox>
10 #include <iostream>
11 #include <qboxlayout.h>
12 #include <qlabel.h>
13 #include <qlogging.h>
14 #include <qnamespace.h>
15 #include <qpushbutton.h>
16 #include <qsplitter.h>
17 #include <qtextedit.h>
18
19 Sidebar::Sidebar(QWidget *parent)
20     : QWidget(parent)
21 {
22     // Main layout setup
23     layout = new QVBoxLayout(this);

```

```

24     layout->setContentsMargins(0, 0, 0, 0);
25
26     mainSplitter = new QSplitter(Qt::Vertical, this);
27
28     // System creation section
29     systemCreationWidget = new SystemCreationWidget(this);
30     mainSplitter->addWidget(systemCreationWidget);
31
32     // New plot section
33     newPlotListWidget = new QWidget(this);
34     newPlotListLayout = new QVBoxLayout(newPlotListWidget);
35     newPlotListLayout->setContentsMargins(5, 5, 5, 5);
36
37     newPlotLabel = new QLabel("Select probes for a new plot:", this);
38     newPlotLabel->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::
Fixed);
39     addNewPlotButton = new QPushButton("Add plot");
40     newPlotList = new NewPlotList(this);
41     connect(addNewPlotButton, &QPushButton::clicked, this, &Sidebar::
onAddPlotClicked);
42
43     newPlotListLayout->addWidget(newPlotLabel);
44     newPlotListLayout->addWidget(newPlotList);
45     newPlotListLayout->addWidget(addNewPlotButton);
46
47     mainSplitter->addWidget(newPlotListWidget);
48
49     // Current plot section (initially hidden)
50     currentPlotWidget = new QWidget(this);
51     currentPlotLayout = new QVBoxLayout(currentPlotWidget);
52     currentPlotLabel = new QLabel("Modify current plot:", this);
53     currentPlotLabel->setSizePolicy(QSizePolicy::Preferred,
QSizePolicy::Fixed);
54     currentPlotLabel->hide();
55
56     currentPlotLayout->addWidget(currentPlotLabel);
57     mainSplitter->addWidget(currentPlotWidget);
58
59     // Polling rate section
60     pollingRateWidget = new PollingRateWidget(this);
61     mainSplitter->addWidget(pollingRateWidget);
62
63     connect(pollingRateWidget, &PollingRateWidget::
onPollingRateChanged, this, &Sidebar::handlePollingRateChanged);
64     layout->addWidget(mainSplitter);
65 }
66
67 /**
68  * @brief Sets the current plot list widget to display.
69  * @param plotList The DQPlotList widget to show in the current plot
    section.
70  */
71 void Sidebar::setCurrentPlotList(DQPlotList *plotList)
72 {
73     if (currentPlotList == plotList) {
74         return;

```

```
75     }
76
77     if (currentPlotList) {
78         layout->removeWidget(currentPlotList);
79         currentPlotList->hide();
80     }
81
82     if (plotList) {
83         currentPlotList = plotList;
84         layout->addWidget(currentPlotList);
85         currentPlotList->show();
86         currentPlotLabel->show();
87     }
88 }
89
90 /**
91  * @brief Handles polling rate change events from the
92  * PollingRateWidget.
93  * @param ms The new polling rate in milliseconds.
94  */
95 void Sidebar::handlePollingRateChanged(int ms)
96 {
97     Q_EMIT onPollingRateChanged(ms);
98 }
99
100 /**
101  * @brief Hides the current plot management section.
102  */
103 void Sidebar::hideCurrentPlot()
104 {
105     if (currentPlotList) {
106         layout->removeWidget(currentPlotList);
107         currentPlotList = nullptr;
108     }
109     currentPlotLabel->hide();
110 }
111
112 /**
113  * @brief Clears new plot selection after plot creation.
114  */
115 void Sidebar::clearOnAdd()
116 {
117     newPlotList->clearSelection();
118 }
119
120 /**
121  * @brief Handles the "Add plot" button click event.
122  * Validates selection and emits addPlotClicked() signal.
123  */
124 void Sidebar::onAddPlotClicked()
125 {
126     auto selectedItems = newPlotList->getSelectedItems();
127
128     if (selectedItems.empty()) {
129         QMessageBox::warning(this, "No Selection", "Please select
130 probes before adding a plot.");
131     }
132 }
```

```

129         return;
130     }
131
132     Q_EMIT addPlotClicked();
133 }

```

C.2.14 SnapshotViewerWindow.cpp

```

1  /**
2   * @file SnapshotViewerWindow.cpp
3   * @brief Implementation of the SnapshotViewerWindow class.
4   */
5
6  #include "SnapshotViewerWindow.h"
7
8  #include <QComboBox>
9  #include <QHBoxLayout>
10 #include <QLabel>
11 #include <QSlider>
12 #include <QVBoxLayout>
13 #include <QtCharts/QChartView>
14 #include <QtCharts/QLineSeries>
15 #include <QtCharts/QValueAxis>
16 #include <QtConcurrent>
17 #include <qnamespace.h>
18
19 /**
20 * @brief Constructs a new SnapshotViewerWindow.
21 */
22 SnapshotViewerWindow::SnapshotViewerWindow(std::vector<Snapshot> &
    snapshotList, QWidget *parent)
23     : QWidget(parent)
24     , observableSelector(new QComboBox(this))
25     , timeSlider(new QSlider(Qt::Horizontal, this))
26     , timeLabel(new QLabel(this))
27     , chartView(new QChartView(new QChart(), this))
28 {
29     auto *mainLayout = new QHBoxLayout(this);
30
31     // Chart area
32     chartView->setRenderHint(QPainter::Antialiasing);
33     mainLayout->addWidget(chartView, 3); // 3/4 of space
34
35     // Controls
36     auto *controlLayout = new QVBoxLayout(this);
37     controlLayout->addWidget(new QLabel("Select Observable:"));
38     controlLayout->addWidget(observableSelector);
39
40     controlLayout->addWidget(new QLabel("Select Time:"));
41     controlLayout->addWidget(timeSlider);
42     controlLayout->addWidget(timeLabel);
43     controlLayout->addStretch();
44     mainLayout->addLayout(controlLayout, 1); // 1/4 of space
45 }

```

```

46     connect(observableSelector, &QComboBox::currentTextChanged, this,
    &SnapshotViewerWindow::onObservableChanged);
47     connect(timeSlider, &QSlider::valueChanged, this, &
    SnapshotViewerWindow::onTimeSliderChanged);
48
49     setSnapshots(snapshotList);
50 }
51
52 /**
53  * @brief Sets the snapshot data and updates the UI.
54  */
55 void SnapshotViewerWindow::setSnapshots(std::vector<Snapshot> &
    snapshotList)
56 {
57     snapshots.clear();
58     observableSelector->clear();
59
60     for (auto &s : snapshotList) {
61         snapshots[s.getName()] = s;
62         observableSelector->addItem(QString::fromStdString(s.getName())
    ));
63     }
64
65     if (!snapshots.empty()) {
66         observableSelector->setCurrentIndex(0);
67         onObservableChanged(observableSelector->currentText());
68     }
69 }
70
71 /**
72  * @brief Handles the change of observable by updating the slider
    range and triggering a plot update.
73  */
74 void SnapshotViewerWindow::onObservableChanged(const QString &name)
75 {
76     currentObservable = name.toStdString();
77
78     const auto &snapshot = snapshots.at(currentObservable);
79     int count = static_cast<int>(snapshot.getObservedSize());
80     timeSlider->setRange(0, std::max(0, count - 1));
81     timeSlider->setValue(0);
82     onTimeSliderChanged(0);
83 }
84
85 /**
86  * @brief Handles changes in the time slider by updating the chart.
87  */
88 void SnapshotViewerWindow::onTimeSliderChanged(int value)
89 {
90     if (snapshots.empty() || !snapshots.count(currentObservable))
91         return;
92
93     const auto &snapshot = snapshots.at(currentObservable);
94
95     if (snapshot.getObservedSize() == 0 || value >= static_cast<int>(
    snapshot.getObservedSize()))

```



```

96         return;
97
98         const auto obs = snapshot.getObservedDeltaQs()[value];
99         const auto calc = snapshot.getCalculatedSize() > value ? std::
optional<DeltaQRepr>(snapshot.getCalculatedDeltaQs()[value]) : std
::nullopt;
100         const auto qta = snapshot.getQTAs()[value];
101         auto time = obs.time;
102
103         qint64 msTime = time / 1000000;
104         QDateTime timestamp = QDateTime::fromMSecsSinceEpoch(msTime);
105         timeLabel->setText(QString("Snapshot at: %1").arg(timestamp.
toString()));
106
107         auto ret = QtConcurrent::run( [=]() {
108             int bins = obs.deltaQ.getBins();
109             double binWidth = obs.deltaQ.getBinWidth();
110
111             QVector<QPointF> obsMean, obsLower, obsUpper, obsCdf, qtaData;
112             obsCdf.reserve(bins + 1);
113             obsMean.reserve(bins + 1);
114             obsLower.reserve(bins + 1);
115             obsUpper.reserve(bins + 1);
116
117             obsCdf.append(QPointF(0, 0));
118             obsMean.append(QPointF(0, 0));
119             obsLower.append(QPointF(0, 0));
120             obsUpper.append(QPointF(0, 0));
121
122             for (int i = 0; i < bins; ++i) {
123                 double x = binWidth * (i + 1);
124                 obsCdf.append(QPointF(x, obs.deltaQ.cdfAt(i)));
125                 obsLower.append(QPointF(x, obs.bounds[i].lowerBound));
126                 obsUpper.append(QPointF(x, obs.bounds[i].upperBound));
127                 obsMean.append(QPointF(x, obs.bounds[i].mean));
128             }
129
130             if (qta.defined) {
131                 double maxDelay = bins * binWidth;
132                 qtaData = {
133                     {qta.perc_25, 0},
134                     {qta.perc_25, 0.25},
135                     {qta.perc_50, 0.25},
136                     {qta.perc_50, 0.5},
137                     {qta.perc_75, 0.5},
138                     {qta.perc_75, 0.75},
139                     {maxDelay, 0.75},
140                     {maxDelay, qta.cdfMax}
141                 };
142             }
143
144             QVector<QPointF> calcCdf, calcMean, calcLower, calcUpper;
145             if (calc) {
146                 int cbins = calc->deltaQ.getBins();
147                 double cbinWidth = calc->deltaQ.getBinWidth();
148                 calcCdf.reserve(cbins + 1);

```

```

149         calcMean.reserve(cbins + 1);
150         calcLower.reserve(cbins + 1);
151         calcUpper.reserve(cbins + 1);
152
153         calcCdf.append(QPointF(0, 0));
154         calcMean.append(QPointF(0, 0));
155         calcLower.append(QPointF(0, 0));
156         calcUpper.append(QPointF(0, 0));
157
158         for (int i = 0; i < cbins; ++i) {
159             double x = cbinWidth * (i + 1);
160             calcCdf.append(QPointF(x, calc->deltaQ.cdfAt(i)));
161             calcLower.append(QPointF(x, calc->bounds[i].lowerBound
162 ));
163             calcUpper.append(QPointF(x, calc->bounds[i].upperBound
164 ));
165             calcMean.append(QPointF(x, calc->bounds[i].mean));
166         }
167     }
168
169     QMetaObject::invokeMethod(this, [=]() {
170         QChart *chart = chartView->chart();
171
172         // Clear previous content
173         chart->removeAllSeries();
174         const auto axes = chart->axes();
175         for (QAbstractAxis *axis : axes) {
176             chart->removeAxis(axis);
177             axis->deleteLater(); // Safe axis cleanup
178         }
179
180         chart->setTitle(QString::fromStdString(currentObservable)
181 + QString(" - Time Index: %1").arg(value));
182
183         auto addSeries = [&](const QVector<QPointF> &data, const
184 QString &name, const QColor &color) {
185             QLineSeries *series = new QLineSeries();
186             series->setName(name);
187             series->append(data);
188             series->setColor(color);
189             chart->addSeries(series);
190             return series;
191         };
192
193         addSeries(obsCdf, "Observed", Qt::blue);
194         addSeries(obsMean, "Obs Mean", Qt::yellow);
195         addSeries(obsLower, "Obs Lower", Qt::green);
196         addSeries(obsUpper, "Obs Upper", Qt::darkGreen);
197         addSeries(qtaData, "QTA", Qt::darkBlue);
198
199         if (!calcCdf.isEmpty()) {
200             addSeries(calcCdf, "Calculated", Qt::red);
201             addSeries(calcMean, "Calc Mean", Qt::darkYellow);
202             addSeries(calcLower, "Calc Lower", Qt::magenta);
203             addSeries(calcUpper, "Calc Upper", Qt::darkMagenta);
204         }

```

```

201
202     auto *axisX = new QValueAxis();
203     axisX->setTitleText("Delay (s)");
204     chart->addAxis(axisX, Qt::AlignBottom);
205
206     auto *axisY = new QValueAxis();
207     axisY->setTitleText("ΔQ(x)");
208     axisY->setRange(0, 1);
209     chart->addAxis(axisY, Qt::AlignLeft);
210
211     for (auto *series : chart->series()) {
212         series->attachAxis(axisX);
213         series->attachAxis(axisY);
214     }
215     });
216 });
217 }

```

C.2.15 StubControlWidget.cpp

```

1 #include "StubControlWidget.h"
2
3 StubControlWidget::StubControlWidget(QWidget *parent)
4     : QWidget(parent)
5 {
6     mainLayout = new QVBoxLayout(this);
7
8     // Erlang Control Group
9     QGroupBox *erlangGroup = new QGroupBox("Erlang Wrapper Control",
10     this);
11     QVBoxLayout *erlangMainLayout = new QVBoxLayout(erlangGroup);
12
13     // First row: IP and Port
14     QHBoxLayout *erlangIpPortLayout = new QHBoxLayout();
15     erlangIpPortLayout->addWidget(new QLabel("IP:"));
16     erlangReceiverIpEdit = new QLineEdit("127.0.0.1", this);
17     erlangIpPortLayout->addWidget(erlangReceiverIpEdit);
18     erlangIpPortLayout->addWidget(new QLabel("Port:"));
19     erlangReceiverPortEdit = new QLineEdit("8081", this);
20     erlangIpPortLayout->addWidget(erlangReceiverPortEdit);
21     erlangMainLayout->addLayout(erlangIpPortLayout);
22
23     // Second row: Set Endpoint Button
24     setErlangEndpointButton = new QPushButton("Set Erlang Endpoint",
25     this);
26     erlangMainLayout->addWidget(setErlangEndpointButton, 0, Qt::
27     AlignLeft);
28
29     // Third row: Start/Stop Wrapper
30     QHBoxLayout *erlangButtonsLayout = new QHBoxLayout();
31     stopErlangButton = new QPushButton("Stop Wrapper", this);
32     startErlangButton = new QPushButton("Start Wrapper", this);
33     erlangButtonsLayout->addWidget(stopErlangButton);
34     erlangButtonsLayout->addWidget(startErlangButton);
35     erlangMainLayout->addLayout(erlangButtonsLayout);

```

```

33
34 // Server Control Group
35 QGroupBox *serverGroup = new QGroupBox("C++ Server Control", this)
36 ;
37 QVBoxLayout *serverMainLayout = new QVBoxLayout(serverGroup);
38
39 // First row: IP and Port
40 QHBoxLayout *serverIpPortLayout = new QHBoxLayout();
41 serverIpPortLayout->addWidget(new QLabel("IP:"));
42 serverIpEdit = new QLineEdit("0.0.0.0", this);
43 serverIpPortLayout->addWidget(serverIpEdit);
44 serverIpPortLayout->addWidget(new QLabel("Port:"));
45 serverPortEdit = new QLineEdit("8080", this);
46 serverIpPortLayout->addWidget(serverPortEdit);
47 serverMainLayout->addLayout(serverIpPortLayout);
48
49 // Second row: Start/Stop Server
50 QHBoxLayout *serverButtonsLayout = new QHBoxLayout();
51 startServerButton = new QPushButton("Start Oscilloscope Server",
52 this);
53 stopServerButton = new QPushButton("Stop Oscilloscope Server",
54 this);
55 serverButtonsLayout->addWidget(startServerButton);
56 serverButtonsLayout->addWidget(stopServerButton);
57 serverMainLayout->addLayout(serverButtonsLayout);
58
59 // Add to main layout
60 mainLayout->addWidget(erlangGroup);
61 mainLayout->addWidget(serverGroup);
62 mainLayout->addStretch();
63 setLayout(mainLayout);
64 setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Maximum);
65
66 // Connect signals
67 connect(startErlangButton, &QPushButton::clicked, this, &
68 StubControlWidget::onStartErlangClicked);
69 connect(stopErlangButton, &QPushButton::clicked, this, &
70 StubControlWidget::onStopErlangClicked);
71 connect(startServerButton, &QPushButton::clicked, this, &
72 StubControlWidget::onStartServerClicked);
73 connect(stopServerButton, &QPushButton::clicked, this, &
74 StubControlWidget::onStopServerClicked);
75 connect(setErlangEndpointButton, &QPushButton::clicked, this, &
76 StubControlWidget::onSetErlangEndpointClicked);
77 }
78
79 void StubControlWidget::onStartErlangClicked()
80 {
81     Application::getInstance().setStubRunning(true);
82 }
83
84 void StubControlWidget::onStopErlangClicked()
85 {
86     Application::getInstance().setStubRunning(false);
87 }
88

```

```

81 void StubControlWidget::onStartServerClicked()
82 {
83     QString ip = serverIpEdit->text();
84     int port = serverPortEdit->text().toInt();
85
86     if (Application::getInstance().startCppServer(ip.toStdString(),
87     port)) {
88         startServerButton->setEnabled(false);
89         stopServerButton->setEnabled(true);
90     }
91 }
92 void StubControlWidget::onStopServerClicked()
93 {
94     Application::getInstance().stopCppServer();
95     startServerButton->setEnabled(true);
96     stopServerButton->setEnabled(false);
97 }
98
99 void StubControlWidget::onSetErlangEndpointClicked()
100 {
101     QString ip = erlangReceiverIpEdit->text();
102     int port = erlangReceiverPortEdit->text().toInt();
103
104     Application::getInstance().setErlangEndpoint(ip.toStdString(),
105     port);
106 }

```

C.2.16 SystemCreationWidget.cpp

```

1  #include "SystemCreationWidget.h"
2  #include "../Application.h"
3  #include "../parser/SystemParserInterface.h"
4  #include <QFileDialog>
5  #include <QMessageBox>
6  #include <fstream>
7
8  /**
9   * @brief Constructs the SystemCreationWidget and sets up the UI.
10  * @param parent Parent QWidget.
11  */
12 SystemCreationWidget::SystemCreationWidget(QWidget *parent)
13     : QWidget(parent)
14 {
15     mainLayout = new QVBoxLayout(this);
16     mainLayout->setAlignment(Qt::AlignTop);
17     mainLayout->setSpacing(10);
18     mainLayout->setContentsMargins(0, 0, 0, 0);
19
20     systemLabel = new QLabel("Create or edit your system here");
21     systemTextEdit = new QTextEdit();
22
23     buttonLayout = new QHBoxLayout();
24     updateSystemButton = new QPushButton("Create or edit system");
25     saveSystemButton = new QPushButton("Save system to");

```

```

26     loadSystemButton = new QPushButton("Load system from");
27
28     buttonLayout->addWidget(updateSystemButton);
29     buttonLayout->addWidget(saveSystemButton);
30     buttonLayout->addWidget(loadSystemButton);
31
32     mainLayout->addWidget(systemLabel);
33     mainLayout->addWidget(systemTextEdit);
34     mainLayout->addLayout(buttonLayout);
35
36     connect(updateSystemButton, &QPushButton::clicked, this, &
SystemCreationWidget::onUpdateSystem);
37     connect(saveSystemButton, &QPushButton::clicked, this, &
SystemCreationWidget::saveSystemTo);
38     connect(loadSystemButton, &QPushButton::clicked, this, &
SystemCreationWidget::loadSystem);
39 }
40
41 /**
42  * @brief Gets the text currently in the system text editor.
43  * @return System text as a std::string.
44  */
45 std::string SystemCreationWidget::getSystemText() const
46 {
47     return systemTextEdit->toPlainText().toStdString();
48 }
49
50 /**
51  * @brief Sets the content of the system text editor.
52  * @param text The new system text.
53  */
54 void SystemCreationWidget::setSystemText(const std::string &text)
55 {
56     systemTextEdit->setText(QString::fromStdString(text));
57 }
58
59 /**
60  * @brief Parses the system text and updates the application system if
        valid.
61  */
62 void SystemCreationWidget::onUpdateSystem()
63 {
64     std::string text = getSystemText();
65
66     try {
67         auto system = SystemParserInterface::parseString(text);
68         if (system.has_value()) {
69             Application::getInstance().setSystem(system.value());
70             system->setSystemDefinitionText(text);
71             Q_EMIT systemUpdated();
72         }
73     } catch (const std::exception &e) {
74         QMessageBox::critical(this, "Parsing error", e.what());
75     }
76 }
77

```

```

78 /**
79  * @brief Opens a dialog to save the current system to a file.
80  */
81 void SystemCreationWidget::saveSystemTo()
82 {
83     QFileDialog dialog(this);
84     dialog.setFileMode(QFileDialog::AnyFile);
85     QString filename = dialog.getSaveFileName(this, "Save file", "", "
All files (*.dq)");
86
87     if (!filename.isEmpty()) {
88         std::string systemText = getSystemText();
89         auto system = SystemParserInterface::parseString(systemText);
90
91         if (system.has_value()) {
92             std::ofstream outFile(filename.toStdString());
93             if (outFile.is_open()) {
94                 outFile << systemText;
95                 outFile.close();
96                 QMessageBox::information(this, "Success", "File saved
successfully.");
97                 Q_EMIT systemSaved();
98             } else {
99                 QMessageBox::critical(this, "Error", "Could not open
file for writing.");
100             }
101         } else {
102             QMessageBox::warning(this, "Error", "System parsing failed
. File not saved.");
103         }
104     }
105 }
106
107 /**
108  * @brief Opens a dialog to load a system from a file, parses it, and
updates the editor.
109  */
110 void SystemCreationWidget::loadSystem()
111 {
112     QFileDialog dialog(this);
113     std::string filename = dialog.getOpenFileName(this, "Select file",
" ", "All files (*.dq)").toStdString();
114
115     auto system = SystemParserInterface::parseFile(filename);
116     if (system.has_value()) {
117         Application::getInstance().setSystem(system.value());
118         std::ifstream file(filename);
119         std::string str;
120         std::string fileContents;
121         while (std::getline(file, str)) {
122             fileContents += str;
123             fileContents.push_back('\n');
124         }
125
126         system->setSystemDefinitionText(fileContents);
127         setSystemText(fileContents);

```

```

128         Q_EMIT systemLoaded();
129     }
130 }

```

C.2.17 TriggersTab.cpp

```

1  #include "TriggersTab.h"
2  #include "SnapshotViewerWindow.h"
3  #include <QDateTime>
4  #include <QLabel>
5  #include <QString>
6  #include <QTimer>
7  #include <cstdlib>
8  #include <sstream>
9  #include <string>
10 #include <unordered_set>
11
12 TriggersTab::TriggersTab(QWidget *parent)
13     : QWidget(parent)
14 {
15     // Main layout setup
16     mainLayout = new QVBoxLayout(this);
17     formLayout = new QFormLayout();
18
19     // Observable selection dropdown
20     observableComboBox = new QComboBox(this);
21     connect(observableComboBox, &QComboBox::currentTextChanged, this,
22     &TriggersTab::onObservableChanged);
23     formLayout->addRow("Probe:", observableComboBox);
24
25     // Sample limit controls
26     sampleLimitCheckBox = new QCheckBox("Sample Limit >", this);
27     sampleLimitSpinBox = new QSpinBox(this);
28     sampleLimitSpinBox->setRange(1, 100000);
29     sampleLimitSpinBox->setValue(sampleLimitThreshold);
30
31     sampleLimitLayout = new QHBoxLayout();
32     sampleLimitLayout->addWidget(sampleLimitCheckBox);
33     sampleLimitLayout->addWidget(sampleLimitSpinBox);
34
35     sampleLimitWidget = new QWidget(this);
36     sampleLimitWidget->setLayout(sampleLimitLayout);
37     formLayout->addRow(sampleLimitWidget);
38
39     // QTA bounds violation checkbox
40     qtaBoundsCheckBox = new QCheckBox("QTA Bound Violation", this);
41     formLayout->addRow(qtaBoundsCheckBox);
42
43     // Connect signals
44     connect(sampleLimitCheckBox, &QCheckBox::checkStateChanged, this,
45     &TriggersTab::onTriggerChanged);
46     connect(sampleLimitSpinBox, QOverload<int>::of(&QSpinBox::
47     valueChanged), this, &TriggersTab::onTriggerChanged);
48     connect(qtaBoundsCheckBox, &QCheckBox::checkStateChanged, this, &
49     TriggersTab::onTriggerChanged);

```



```

46
47     mainLayout->addLayout(formLayout);
48
49     // Triggered events list
50     triggeredList = new QListWidget(this);
51     triggeredList->setSelectionMode(QAbstractItemView::SingleSelection
52 );
53     triggeredList->setMinimumHeight(150);
54     connect(triggeredList, &QListWidget::itemClicked, this, &
55 TriggersTab::onTriggeredItemClicked);
56
57     mainLayout->addWidget(new QLabel("Triggered Snapshots:"));
58     mainLayout->addWidget(triggeredList);
59
60     // Set up system observer
61     Application::getInstance().addObserver([this]() { this->
62 populateObservables(); });
63
64     populateObservables();
65 }
66
67 TriggersTab::~TriggersTab()
68 {
69     delete mainLayout;
70 }
71
72 /**
73  * @brief Populates the observable dropdown with available probes and
74  * outcomes.
75  */
76 void TriggersTab::populateObservables()
77 {
78     try {
79         auto system = Application::getInstance().getSystem();
80         observableComboBox->clear();
81         for (const auto &[name, obs] : system->getObservables()) {
82             if (obs) {
83                 observableComboBox->addItem(QString::fromStdString(
84 name));
85             }
86         }
87     } catch (std::exception &) {
88         return;
89     }
90 }
91
92 /**
93  * @brief Handles observable selection changes.
94  * @param name The newly selected observable name (unused, signal
95  * requires parameter).
96  */
97 void TriggersTab::onObservableChanged(const QString &)
98 {
99     updateCheckboxStates();
100 }

```

```

96 /**
97  * @brief Updates triggers when conditions change.
98  */
99 void TriggersTab::onTriggerChanged()
100 {
101     try {
102         auto observable = getCurrentObservable();
103         std::string name = observable->getName();
104
105         // Update sample limit trigger
106         if (sampleLimitCheckBox->isChecked()) {
107             observable->addTrigger(
108                 TriggerType::SampleLimit,
109                 TriggerDefs::Conditions::SampleLimit(
110 sampleLimitSpinBox->value()),
111                 [this, name](const DeltaQ &, const QTA &, std::
112 uint64_t time) {
113                     this->captureSnapshots(time, name);
114                 },
115                 true,
116                 sampleLimitSpinBox->value());
117         } else {
118             observable->removeTrigger(TriggerType::SampleLimit);
119         }
120
121         // Update QTA bounds trigger
122         if (qtaBoundsCheckBox->isChecked()) {
123             observable->addTrigger(
124                 TriggerType::QTAViolation,
125                 TriggerDefs::Conditions::QTABounds(),
126                 [this, name](const DeltaQ &, const QTA &, std::
127 uint64_t time) {
128                     this->captureSnapshots(time, name);
129                 },
130                 true,
131                 std::nullopt);
132         } else {
133             observable->removeTrigger(TriggerType::QTAViolation);
134         }
135     } catch (std::exception &) {
136         return;
137     }
138 }
139
140 /**
141  * @brief Captures snapshots when triggers are activated.
142  * @param time The timestamp of the trigger event.
143  * @param name The name of the observable that triggered.
144  */
145 void TriggersTab::captureSnapshots(std::uint64_t time, const std::
146 string &name)
147 {
148     auto system = Application::getInstance().getSystem();
149     if (!system->isRecording()) {
150         system->setRecording(true);
151         QMetaObject::invokeMethod(

```

```

148         this,
149         [this, name, system, time]() {
150             auto *timer = new QTimer(this);
151             timer->setSingleShot(true);
152
153             connect(timer, &QTimer::timeout, this, [=]() {
154                 system->getObservablesSnapshotAt(time);
155
156                 // Format timestamp for display
157                 qint64 msTime = time / 1000000;
158                 QDateTime timestamp = QDateTime::
fromMSEcsSinceEpoch(msTime);
159
160                 // Create display string
161                 QString timestampStr = timestamp.toString();
162                 std::ostringstream oss;
163                 oss << "Snapshot at: " << timestampStr.toStdString
() << " from " << name;
164                 QString snapshotString = QString::fromStdString(
oss.str());
165
166                 // Add to triggered list
167                 auto *item = new QListWidgetItem(snapshotString);
168                 item->setData(Qt::UserRole, static_cast<qulonglong
>(time));
169                 triggeredList->addItem(item);
170                 timer->deleteLater();
171
172                 system->setRecording(false);
173             });
174
175             timer->start(5000);
176         },
177         Qt::QueuedConnection);
178     }
179 }
180
181 /**
182  * @brief Updates checkbox states based on current triggers.
183  */
184 void TriggersTab::updateCheckboxStates()
185 {
186     try {
187         auto observable = getCurrentObservable();
188         auto &manager = observable->getTriggerManager();
189
190         // Get active trigger types
191         auto all = manager.getAllTriggers();
192         std::unordered_set<TriggerType> activeTypes;
193         for (const auto &trigger : all) {
194             if (trigger.enabled)
195                 activeTypes.insert(trigger.type);
196         }
197
198         // Update UI to match active triggers

```

```

199     sampleLimitCheckBox->setChecked(activeTypes.count(TriggerType
200     ::SampleLimit));
201     qtaBoundsCheckBox->setChecked(activeTypes.count(TriggerType::
202     QTAViolation));
203
204     // Update sample limit value if trigger exists
205     for (const auto &t : all) {
206         if (t.type == TriggerType::SampleLimit && t.
207         sampleLimitValue) {
208             sampleLimitSpinBox->setValue(*t.sampleLimitValue);
209         }
210     }
211     catch (std::exception &) {
212         sampleLimitCheckBox->setChecked(false);
213         qtaBoundsCheckBox->setChecked(false);
214     }
215 }
216
217 /**
218  * @brief Gets the currently selected observable.
219  * @return current observable.
220  * @throws std::runtime_error if system or observable doesn't exist.
221  */
222 std::shared_ptr<Observable> TriggersTab::getCurrentObservable()
223 {
224     auto system = Application::getInstance().getSystem();
225     if (!system)
226         throw std::runtime_error("System does not exist");
227
228     std::string name = observableComboBox->currentText().toStdString();
229     ;
230     auto observable = system->getObservable(name);
231     if (!observable)
232         throw std::runtime_error("Observable does not exist");
233
234     return observable;
235 }
236
237 /**
238  * @brief Handles triggered item clicks to show snapshots.
239  * @param item The clicked list item containing snapshot data.
240  */
241 void TriggersTab::onTriggeredItemClicked(QListWidgetItem *item)
242 {
243     if (!item)
244         return;
245
246     // Retrieve timestamp from item data
247     qulonglong timestamp = item->data(Qt::UserRole).toULongLong();
248     if (timestamp == 0)
249         return;
250
251     // Find and display corresponding snapshot
252     auto system = Application::getInstance().getSystem();
253     const auto &snapshots = system->getAllSnapshots();
254     auto it = snapshots.find(timestamp);

```

```
251     if (it != snapshots.end()) {
252         const auto &snapshotList = it->second;
253
254         auto *viewer = new SnapshotViewerWindow(const_cast<std::vector<
255             <Snapshot> &>(snapshotList));
256         viewer->setAttribute(Qt::WA_DeleteOnClose);
257         viewer->resize(800, 600);
258         viewer->show();
259     }
```

C.3 diagram

C.3.1 Observable.cpp

```
1  #include "Observable.h"
2  #include <algorithm>
3  #include <cmath>
4  #include <cstdint>
5  #include <iostream>
6
7  #define MAX_DQ 30
8  Observable::Observable(const std::string &name)
9      : observedInterval(50)
10      , name(name)
11  {
12      observableSnapshot.setName(name);
13  }
14
15  void Observable::addSample(const Sample &sample)
16  {
17      samples.emplace_back(sample);
18      sorted = false;
19  }
20
21  std::vector<Sample> Observable::getSamplesInRange(std::uint64_t
22      lowerTime, std::uint64_t upperTime)
23  {
24      std::lock_guard<std::mutex> lock(samplesMutex);
25      if (!sorted) {
26          std::sort(samples.begin(), samples.end(), [](const Sample &a,
27              const Sample &b) { return a.endTime < b.endTime; });
28          sorted = true;
29      }
30
31      std::vector<Sample> selectedSamples;
32
33      auto lower = std::lower_bound(samples.begin(), samples.end(),
34          lowerTime, [](const Sample &s, long long time) { return s.endTime
35              < time; });
36
37      auto upper = std::upper_bound(samples.begin(), samples.end(),
38          upperTime, [](long long time, const Sample &s) { return time < s.
39              endTime; });
```

```

34
35     for (auto it = lower; it != upper; ++it) {
36         selectedSamples.emplace_back(*it);
37     }
38     samples.erase(std::remove_if(samples.begin(), samples.end(), [
upperTime](const Sample &s) { return s.endTime < upperTime; }),
samples.end());
39
40     return selectedSamples;
41 }
42
43 DeltaQ Observable::calculateObservedDeltaQ(std::uint64_t
timeLowerBound, std::uint64_t timeUpperBound)
44 {
45     auto samplesInRange = getSamplesInRange(timeLowerBound,
timeUpperBound);
46     if (samplesInRange.empty()) {
47         DeltaQ deltaQ = DeltaQ();
48         updateSnapshot(timeLowerBound, deltaQ);
49         return deltaQ;
50     }
51
52     DeltaQ deltaQ {getBinWidth(), samplesInRange, nBins};
53     std::cout << deltaQ.toString() << "\n";
54     updateSnapshot(timeLowerBound, deltaQ);
55
56     triggerManager.evaluate(deltaQ, qta, timeLowerBound);
57     return deltaQ;
58 }
59
60 void Observable::updateSnapshot(uint64_t timeLowerBound, DeltaQ &
deltaQ)
61 {
62     if (!(deltaQ == DeltaQ())) {
63         observedInterval.addDeltaQ(deltaQ);
64         confidenceIntervalHistory.push_back(deltaQ);
65
66         if (confidenceIntervalHistory.size() > MAX_DQ) {
67             observedInterval.removeDeltaQ(confidenceIntervalHistory.
front());
68             confidenceIntervalHistory.pop_front();
69         }
70     }
71     observableSnapshot.addObservedDeltaQ(timeLowerBound, deltaQ,
observedInterval.getBounds());
72     observableSnapshot.addQTA(timeLowerBound, qta);
73     if (!recording && (confidenceIntervalHistory.size() > MAX_DQ)) {
74         observableSnapshot.resizeTo(MAX_DQ);
75     }
76 }
77
78 DeltaQ Observable::getObservedDeltaQ(uint64_t timeLowerBound, uint64_t
timeUpperBound)
79 {
80     std::lock_guard<std::mutex> lock(observedMutex);

```

```

81     auto deltaQRepr = observableSnapshot.getObservedDeltaQAtTime(
timeLowerBound);
82     if (!deltaQRepr.has_value()) {
83         calculateObservedDeltaQ(timeLowerBound, timeUpperBound);
84         deltaQRepr = observableSnapshot.getObservedDeltaQAtTime(
timeLowerBound);
85     }
86     return deltaQRepr.value().deltaQ;
87 }
88
89 DeltaQRepr Observable::getObservedDeltaQRepr(uint64_t timeLowerBound,
uint64_t timeUpperBound)
90 {
91     std::lock_guard<std::mutex> lock(observedMutex);
92     auto deltaQRepr = observableSnapshot.getObservedDeltaQAtTime(
timeLowerBound);
93     if (!deltaQRepr.has_value()) {
94         calculateObservedDeltaQ(timeLowerBound, timeUpperBound);
95         deltaQRepr = observableSnapshot.getObservedDeltaQAtTime(
timeLowerBound);
96     }
97     return deltaQRepr.value();
98 }
99
100 void Observable::setQTA(const QTA &newQTA)
101 {
102     if (newQTA.perc_25 > maxDelay || newQTA.perc_50 > maxDelay ||
newQTA.perc_75 > maxDelay)
103         throw std::invalid_argument("Percentages should not be bigger
than maximum delay " + std::to_string(maxDelay));
104     qta = newQTA;
105 }
106
107 void Observable::addTrigger(TriggerType type, TriggerDefs::Condition
condition, TriggerDefs::Action action, bool enabled, std::optional
<int> sampleLimitVal)
108 {
109     auto &tm = triggerManager;
110     tm.addTrigger(type, condition, action, enabled);
111     auto &trigger = tm.getTriggersByType(type).back();
112     trigger.sampleLimitValue = sampleLimitVal;
113 }
114
115 void Observable::removeTrigger(TriggerType type)
116 {
117     triggerManager.removeTriggersByType(type);
118 }
119
120 void Observable::setRecording(bool isRecording)
121 {
122     if (recording && !isRecording) {
123         recording = isRecording;
124         observableSnapshot.resizeTo(30); // FIXME magic numb
125     }
126     recording = isRecording;
127 }

```

```

128
129 Snapshot Observable::getSnapshot()
130 {
131     return observableSnapshot;
132 }
133
134 double Observable::setNewParameters(int newExp, int newNBins)
135 {
136     std::lock_guard<decltype(paramMutex)> lock(paramMutex);
137
138     bool binsChanged = (newNBins != nBins);
139     bool expChanged = (newExp != deltaTExp);
140
141     nBins = newNBins;
142     deltaTExp = newExp;
143
144     maxDelay = DELTA_T_BASE * std::pow(2, deltaTExp) * nBins;
145
146     if (qta.perc_25 > maxDelay || qta.perc_50 > maxDelay || qta.
147     perc_75 > maxDelay) {
148         qta = QTA::create(0, 0, 0, qta.cdfMax, false);
149     }
150
151     if (binsChanged || expChanged) {
152         observedInterval.setNumBins(nBins);
153         confidenceIntervalHistory.clear();
154     }
155
156     return maxDelay;
157 }
158
159 std::string Observable::getName() const &
160 {
161     return name;
162 }

```

C.3.2 Operator.cpp

```

1  #include "Operator.h"
2
3  #include <numeric>
4
5  #include "../maths/DeltaQOperations.h"
6  #include "Observable.h"
7  #include "OperatorType.h"
8  #include <cmath>
9  #include <limits>
10 #define OP_EPSILON std::numeric_limits<double>::epsilon()
11
12 Operator::Operator(const std::string &name, OperatorType type)
13     : Observable(name)
14     , type(type)
15     , calculatedInterval(50)
16 {
17 }

```



```

18
19 Operator::~~Operator() = default;
20
21 void Operator::setProbabilities(const std::vector<double> &probs)
22 {
23     if (type != OperatorType::PRB) {
24         throw std::invalid_argument("Only probabilistic operators
25         accept probabilities");
26     }
27
28     double result = std::reduce(probs.begin(), probs.end());
29
30     if (std::fabs(1 - result) > OP_EPSILON)
31         throw std::logic_error("Result should approximate to 1");
32
33     probabilities = probs;
34 }
35
36 DeltaQ Operator::calculateCalculatedDeltaQ(uint64_t timeLowerBound,
37     uint64_t timeUpperBound)
38 {
39     if (observableSnapshot.getCalculatedDeltaQAtTime(timeLowerBound).
40     has_value()) {
41         return observableSnapshot.getCalculatedDeltaQAtTime(
42         timeLowerBound).value().deltaQ;
43     }
44     std::vector<DeltaQ> deltaQs;
45
46     // Get DeltaQ for all children
47     for (auto &childrenLinks : causalLinks) {
48         std::vector<DeltaQ> childrenDeltaQs;
49
50         // For each children in causal link, get their DeltaQ
51         for (auto &component : childrenLinks) {
52             childrenDeltaQs.push_back(component->getObservedDeltaQ(
53             timeLowerBound, timeUpperBound));
54         }
55         // Get the convolution of the components in a children
56         deltaQs.push_back(convolveN(childrenDeltaQs));
57     }
58     DeltaQ result;
59
60     // Choose appropriate operation
61     if (type == OperatorType::FTF)
62         result = firstToFinish(deltaQs);
63     else if (type == OperatorType::PRB)
64         result = probabilisticChoice(probabilities, deltaQs);
65     else
66         result = allToFinish(deltaQs);
67
68     int calculatedBins = result.getBins();
69
70     if (calculatedBins != calculatedInterval.getBins()) {
71         calculatedInterval.setNumBins(calculatedBins);
72         calculatedDeltaQHistory.clear();
73     }
74 }

```

```

69
70     calculatedInterval.addDeltaQ(result);
71     calculatedDeltaQHistory.push_back(result);
72
73     if (calculatedDeltaQHistory.size() > MAX_DQ) {
74         calculatedInterval.removeDeltaQ(calculatedDeltaQHistory.front
75         ());
76         calculatedDeltaQHistory.pop_front();
77     }
78
79     observableSnapshot.addCalculatedDeltaQ(timeLowerBound, result,
80     calculatedInterval.getBounds());
81
82     if (!recording && calculatedDeltaQHistory.size() > MAX_DQ) {
83         observableSnapshot.removeOldestCalculatedDeltaQ();
84     }
85     return result;
86 }
87
88 DeltaQRepr Operator::getCalculatedDeltaQRepr(uint64_t timeLowerBound,
89 uint64_t timeUpperBound)
90 {
91     std::lock_guard<std::mutex> lock(calcMutex);
92     auto deltaQRepr = observableSnapshot.getCalculatedDeltaQAtTime(
93     timeLowerBound);
94     if (!deltaQRepr.has_value()) {
95         calculateCalculatedDeltaQ(timeLowerBound, timeUpperBound);
96         deltaQRepr = observableSnapshot.getCalculatedDeltaQAtTime(
97         timeLowerBound);
98     }
99     return deltaQRepr.value();
100 }

```

C.3.3 Outcome.cpp

```

1 #include "Outcome.h"
2
3 Outcome::Outcome(const std::string &name)
4     : Observable(name)
5 {
6 }
7
8 Outcome::~Outcome() = default;

```

C.3.4 Probe.cpp

```

1 #include "Probe.h"
2 #include "../maths/ConfidenceInterval.h"
3 #include "../maths/DeltaQOperations.h"
4 #include <iostream>
5 #include <memory>
6 #include <mutex>
7 Probe::Probe(const std::string &name)

```

```

8      : Observable(name)
9      , calculatedInterval(50)
10 {
11 }
12
13 Probe::Probe(const std::string &name, std::vector<std::shared_ptr<
    Observable>> causalLinks)
14     : Observable(name)
15     , causalLinks(causalLinks)
16     , calculatedInterval(50)
17 {
18 }
19
20 Probe::~~Probe() = default;
21
22 DeltaQ Probe::calculateCalculatedDeltaQ(uint64_t timeLowerBound,
    uint64_t timeUpperBound)
23 {
24     if (observableSnapshot.getCalculatedDeltaQAtTime(timeLowerBound).
        has_value()) {
25         return observableSnapshot.getCalculatedDeltaQAtTime(
            timeLowerBound).value().deltaQ;
26     }
27     std::vector<DeltaQ> deltaQs;
28     for (const auto &component : causalLinks) {
29
30         deltaQs.push_back(component->getObservedDeltaQ(timeLowerBound,
            timeUpperBound));
31     }
32
33     DeltaQ result = convolveN(deltaQs);
34     int calculatedBins = result.getBins();
35
36     if (calculatedBins != calculatedInterval.getBins()) {
37         calculatedInterval.setNumBins(calculatedBins);
38         calculatedDeltaQHistory.clear();
39     }
40
41     calculatedInterval.addDeltaQ(result);
42     calculatedDeltaQHistory.push_back(result);
43
44     if (calculatedDeltaQHistory.size() > MAX_DQ) {
45         calculatedInterval.removeDeltaQ(calculatedDeltaQHistory.front(
            ));
46         calculatedDeltaQHistory.pop_front();
47     }
48
49     observableSnapshot.addCalculatedDeltaQ(timeLowerBound, result,
        calculatedInterval.getBounds());
50
51     if (!recording && calculatedDeltaQHistory.size() > MAX_DQ) {
52         observableSnapshot.removeOldestCalculatedDeltaQ();
53     }
54     return result;
55 }
56 std::vector<Bound> Probe::getBounds() const

```

```

57 {
58     return observedInterval.getBounds();
59 }
60
61 std::vector<Bound> Probe::getObservedBounds() const
62 {
63     return observedInterval.getBounds();
64 }
65
66 std::vector<Bound> Probe::getCalculatedBounds() const
67 {
68     return calculatedInterval.getBounds();
69 }
70
71 DeltaQRepr Probe::getCalculatedDeltaQRepr(uint64_t timeLowerBound,
72     uint64_t timeUpperBound)
73 {
74     std::lock_guard<std::mutex> lock(calcMutex);
75     auto deltaQRepr = observableSnapshot.getCalculatedDeltaQAtTime(
76         timeLowerBound);
77     if (!deltaQRepr.has_value()) {
78         calculateCalculatedDeltaQ(timeLowerBound, timeUpperBound);
79         deltaQRepr = observableSnapshot.getCalculatedDeltaQAtTime(
80             timeLowerBound);
81     }
82     return deltaQRepr.value();
83 }

```

C.3.5 System.cpp

```

1  /**
2   * @author Francesco Nieri
3   * @date 26/10/2024
4   * Class representing a DeltaQ system
5   */
6
7  #include "System.h"
8  #include "../Application.h"
9  #include <utility>
10
11 void System::setOutcomes(std::unordered_map<std::string, std::
12     shared_ptr<Outcome>> outcomesMap)
13 {
14     outcomes = outcomesMap;
15     for (auto &[name, outcome] : outcomes) {
16         observables[name] = outcome;
17     }
18 }
19
20 void System::setProbes(std::unordered_map<std::string, std::shared_ptr
21     <Probe>> probesMap)
22 {
23     probes = probesMap;
24     for (auto &[name, probe] : probes) {
25         observables[name] = probe;
26     }
27 }

```

```
24     }
25 }
26
27 void System::setOperators(std::unordered_map<std::string, std::
    shared_ptr<Operator>> operatorsMap)
28 {
29     operators = operatorsMap;
30     for (auto &[name, op] : operators) {
31         observables[name] = op;
32     }
33 }
34 std::shared_ptr<Outcome> System::getOutcome(const std::string &name)
35 {
36     return outcomes[name];
37 }
38
39 void System::setObservableParameters(std::string &componentName, int
    exponent, int numBins)
40 {
41     if (auto it = observables.find(componentName); it != observables.
        end()) {
42         double maxDelay = it->second->setNewParameters(exponent,
            numBins);
43         Application::getInstance().sendDelayChange(componentName,
            maxDelay * 1000);
44     }
45 }
46
47 void System::addSample(std::string &componentName, Sample &sample)
48 {
49     if (auto it = observables.find(componentName); it != observables.
        end()) {
50         it->second->addSample(sample);
51     }
52 }
53
54 DeltaQ System::calculateDeltaQ()
55 {
56     return DeltaQ();
57 }
58
59 [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<Outcome
    >> &System::getOutcomes()
60 {
61     return outcomes;
62 }
63
64 [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<Operator
    >> &System::getOperators()
65 {
66     return operators;
67 }
68
69 [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<Probe>>
    &System::getProbes()
70 {
```

```
71     return probes;
72 }
73
74 [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<
75     Observable>> &System::getObservables()
76 {
77     return observables;
78 }
79
80 bool System::hasOutcome(const std::string &name)
81 {
82     return outcomes.find(name) != outcomes.end();
83 }
84
85 bool System::hasProbe(const std::string &name)
86 {
87     return probes.find(name) != probes.end();
88 }
89
90 bool System::hasOperator(const std::string &name)
91 {
92     return operators.find(name) != operators.end();
93 }
94
95 std::shared_ptr<Operator> System::getOperator(const std::string &name)
96 {
97     return operators[name];
98 }
99
100 std::shared_ptr<Probe> System::getProbe(const std::string &name)
101 {
102     return probes[name];
103 }
104
105 std::vector<std::string> System::getAllComponentsName()
106 {
107     std::vector<std::string> names;
108     names.reserve(observables.size());
109
110     for (auto &[name, obs] : observables) {
111         if (obs)
112             names.emplace_back(name);
113     }
114
115     return names;
116 }
117
118 void System::setSystemDefinitionText(std::string &text)
119 {
120     systemDefinitionText = text;
121 }
122
123 std::string System::getSystemDefinitionText()
124 {
125     return systemDefinitionText;
126 }
```

```

126
127 std::shared_ptr<Observable> System::getObservable(const std::string &
    name)
128 {
129     return observables[name];
130 }
131
132 void System::setRecording(bool isRecording)
133 {
134     if (recordingTrigger && isRecording) {
135         return;
136     }
137
138     recordingTrigger = isRecording;
139     for (auto &obs : observables) {
140         if (obs.second) {
141             obs.second->setRecording(isRecording);
142         }
143     }
144 }
145
146 bool System::isRecording() const
147 {
148     return recordingTrigger;
149 }
150
151 void System::getObservablesSnapshotAt(std::uint64_t time)
152 {
153     std::vector<Snapshot> result;
154     for (auto &[name, observable] : observables) {
155         if (observable)
156             result.push_back(observable->getSnapshot());
157     }
158
159     snapshots[time] = std::move(result); // store the snapshots by
    timestamp
160 }
161
162 std::map<std::uint64_t, std::vector<Snapshot>> System::getAllSnapshots
    ()
163 {
164     return snapshots;
165 }

```

C.4 maths

C.4.1 ConfidenceInterval.cpp

```

1 #include "ConfidenceInterval.h"
2 #include <iostream>
3 #include <math.h>
4 #include <stdexcept>
5 #include <vector>
6

```

```
7 ConfidenceInterval::ConfidenceInterval()
8 : numBins(0)
9 {
10 }
11
12 ConfidenceInterval::ConfidenceInterval(int numBins)
13 : numBins(numBins)
14 , cdfSum(numBins, 0.0)
15 , cdfSumSquares(numBins, 0.0)
16 , cdfSampleCounts(numBins, 0)
17 , bounds(numBins)
18 {
19 }
20
21 void ConfidenceInterval::addDeltaQ(const DeltaQ &deltaQ)
22 {
23     const auto &cdf = deltaQ.getCdfValues();
24     if (deltaQ == DeltaQ()) {
25         return;
26     }
27
28     if (cdf.size() != numBins) {
29         std::cerr << "CDF size mismatch in addDeltaQ, have" << deltaQ.
30         getBins() << " expected" << numBins << "\n";
31         return;
32     }
33     for (size_t i = 0; i < cdf.size(); ++i) {
34         const double cdfValue = cdf[i];
35
36         cdfSum[i] += cdfValue;
37         cdfSumSquares[i] += cdfValue * cdfValue;
38         cdfSampleCounts[i] += 1;
39     }
40
41     updateConfidenceInterval();
42 }
43
44 void ConfidenceInterval::removeDeltaQ(const DeltaQ &deltaQ)
45 {
46     if (deltaQ == DeltaQ()) {
47         return;
48     }
49
50     const auto &cdf = deltaQ.getCdfValues();
51     if (cdf.size() != numBins) {
52         return; // Returning a previous DeltaQ which had different
53         bins
54     }
55
56     for (size_t i = 0; i < cdf.size(); ++i) {
57         const double cdfValue = cdf[i];
58
59         cdfSum[i] -= cdfValue;
60         cdfSumSquares[i] -= cdfValue * cdfValue;
61
62         if (cdfSampleCounts[i] == 0) {
```



```
61         return; // Removing more than needed
62     }
63
64     cdfSampleCounts[i] -= 1;
65 }
66
67 updateConfidenceInterval();
68 }
69
70 void ConfidenceInterval::updateConfidenceInterval()
71 {
72     for (size_t i = 0; i < bounds.size(); ++i) {
73         unsigned int n = cdfSampleCounts[i];
74         if (cdfSampleCounts[i] == 0) {
75             bounds[i].lowerBound = 0.0;
76             bounds[i].upperBound = 0.0;
77             continue;
78         }
79
80         double mean = cdfSum[i] / n;
81         double meanSquare = cdfSumSquares[i] / n;
82         double variance = meanSquare - (mean * mean);
83         double stddev = std::sqrt(std::max(variance, 0.0));
84         double marginOfError = z * stddev / std::sqrt(n);
85         bounds[i].lowerBound = std::max(0.0, mean - marginOfError);
86         bounds[i].upperBound = std::min(1.0, mean + marginOfError);
87         bounds[i].mean = mean;
88     }
89 }
90
91 void ConfidenceInterval::reset()
92 {
93     bounds = std::vector<Bound>();
94     bounds.resize(numBins);
95     cdfSum = std::vector<double>();
96     cdfSum.resize(numBins);
97     cdfSumSquares = std::vector<double>();
98     cdfSumSquares.resize(numBins);
99     cdfSampleCounts = std::vector<unsigned int>();
100    cdfSampleCounts.resize(numBins);
101 }
102
103 void ConfidenceInterval::setNumBins(int newNumBins)
104 {
105     numBins = newNumBins;
106     reset();
107 }
108
109 std::vector<Bound> ConfidenceInterval::getBounds() const
110 {
111     return bounds;
112 }
113
114 unsigned int ConfidenceInterval::getBins()
115 {
116     return numBins;
117 }
```

117 }

C.4.2 DeltaQ.cpp

```
1
2
3 #include "DeltaQ.h"
4
5 #include <algorithm>
6 #include <cmath>
7 #include <functional>
8 #include <iomanip>
9 #include <iostream>
10
11 #include <chrono>
12 #include <iostream>
13 DeltaQ::DeltaQ(const double binWidth)
14     : binWidth(binWidth)
15     , bins(0)
16     , qta()
17 {
18 }
19
20 DeltaQ::DeltaQ(const double binWidth, const std::vector<double> &
    values, const bool isPdf)
21     : binWidth(binWidth)
22     , bins(values.size()) // Values is binned data
23     , qta()
24 {
25     if (isPdf) {
26         pdfValues = values;
27         calculateCDF();
28     } else {
29         cdfValues = values;
30         calculatePDF();
31     }
32 }
33 DeltaQ::DeltaQ(double binWidth, std::vector<Sample> &samples)
34     : binWidth(binWidth)
35     , bins {50}
36     , qta()
37 {
38     calculateDeltaQ(samples);
39 }
40
41 DeltaQ::DeltaQ(double binWidth, std::vector<Sample> &samples, int bins
    )
42     : binWidth(binWidth)
43     , bins(bins)
44     , qta()
45 {
46     calculateDeltaQ(samples);
47 }
48
49 void DeltaQ::calculateDeltaQ(std::vector<Sample> &outcomeSamples)
```

```

50 {
51
52     double size = binWidth * bins;
53     if (outcomeSamples.empty() || binWidth <= 0) {
54         bins = 0;
55         return;
56     }
57     std::vector<double> histogram(bins, 0.0);
58     totalSamples = outcomeSamples.size();
59     long long successfulSamples = 0;
60
61     std::sort(outcomeSamples.begin(), outcomeSamples.end(), [](const
Sample &a, const Sample &b) { return a.elapsedTime < b.elapsedTime
; });
62
63     for (const auto &sample : outcomeSamples) {
64         if (sample.status != Status::SUCCESS) {
65             continue; // Exclude failed samples from histogram but
count them
66         }
67
68         double elapsed = sample.elapsedTime;
69
70         if (elapsed < 0 || std::isnan(elapsed) || std::isinf(elapsed))
{
71             std::cerr << "Warning: Invalid sample value: " << elapsed
<< std::endl;
72             continue;
73         }
74         double invBinWidth = 1.0 / binWidth;
75         int bin = static_cast<int>(elapsed * invBinWidth);
76         if (bin < 0) {
77             std::cerr << "Warning: Negative bin value: " << bin << std
::endl;
78             continue;
79         }
80         if (bin >= bins) {
81             if (elapsed > size) {
82                 continue;
83             }
84             bin = bins - 1;
85         }
86
87         successfulSamples++;
88         histogram[bin] += 1.0;
89     }
90     // Calculate PDF
91     for (double &val : histogram) {
92         val /= totalSamples;
93     }
94     pdfValues = std::move(histogram);
95
96     calculateCDF();
97     calculateQuartiles(outcomeSamples);
98 }
99

```

```

100 void DeltaQ::calculateQuartiles(std::vector<Sample> &outcomeSamples)
101 {
102     if (outcomeSamples.empty()) {
103         return;
104     }
105     const size_t n = outcomeSamples.size();
106
107     auto getElapsedAt = [&](size_t index) -> double { return
outcomeSamples[index].elapsedTime; };
108
109     auto getPercentile = [&](double p) -> double {
110         double pos = p * (n - 1);
111         auto idx = static_cast<size_t>(pos);
112         double frac = pos - idx;
113
114         if (idx + 1 < n) {
115             // Linear interpolation between samples[idx] and samples[
idx + 1]
116             return getElapsedAt(idx) * (1.0 - frac) + getElapsedAt(idx
+ 1) * frac;
117         }
118         return getElapsedAt(idx);
119     };
120     qta = QTA::create(getPercentile(0.25), getPercentile(0.50),
getPercentile(0.75), ((cdfAt(bins - 1)) > 1) ? 1 : cdfAt(bins - 1)
);
121 }
122
123 void DeltaQ::calculateCDF()
124 {
125     cdfValues.clear();
126     cdfValues.reserve(bins);
127     double cumulativeSum = 0;
128     for (const double &pdfValue : pdfValues) {
129         cumulativeSum += pdfValue;
130         cdfValues.push_back(cumulativeSum);
131     }
132 }
133
134 void DeltaQ::calculatePDF()
135 {
136     pdfValues.clear();
137
138     double previous = 0;
139     for (const double &cdfValue : cdfValues) {
140         pdfValues.push_back(cdfValue - previous);
141         previous = cdfValue;
142     }
143 }
144
145 QTA DeltaQ::getQTA() const
146 {
147     return qta;
148 }
149
150 const std::vector<double> &DeltaQ::getPdfValues() const

```

```
151 {
152     return pdfValues;
153 }
154
155 const std::vector<double> &DeltaQ::getCdfValues() const
156 {
157     return cdfValues;
158 }
159
160 const unsigned int DeltaQ::getTotalSamples() const
161 {
162     return totalSamples;
163 }
164
165 void DeltaQ::setBinWidth(double newWidth)
166 {
167     binWidth = newWidth;
168 }
169
170 double DeltaQ::getBinWidth() const
171 {
172     return binWidth;
173 }
174
175 int DeltaQ::getBins() const
176 {
177     return bins;
178 }
179
180 double DeltaQ::pdfAt(int x) const
181 {
182     if (bins != 0) {
183         if (x >= bins) {
184             return 0.0;
185         }
186         return pdfValues.at(x);
187     }
188     return 0;
189 }
190
191 double DeltaQ::cdfAt(int x) const
192 {
193     if (bins != 0) {
194         if (x >= bins) {
195             return cdfValues.at(bins - 1);
196         }
197         return cdfValues.at(x);
198     }
199     return 0;
200 }
201
202 bool DeltaQ::operator<(const DeltaQ &other) const
203 {
204     return this->bins < other.bins;
205 }
206
```

```
207 bool DeltaQ::operator>(const DeltaQ &other) const
208 {
209     return this->bins > other.bins;
210 }
211
212 bool DeltaQ::operator==(const DeltaQ &deltaQ) const
213 {
214     return pdfValues == deltaQ.getPdfValues();
215 }
216
217 DeltaQ operator*(const DeltaQ &deltaQ, double constant)
218 {
219     DeltaQ result(deltaQ.binWidth);
220     result.bins = deltaQ.bins;
221
222     result.pdfValues = deltaQ.pdfValues;
223     std::transform(result.pdfValues.begin(), result.pdfValues.end(),
224 result.pdfValues.begin(), [constant](double value) { return value
225 * constant; });
226
227     result.cdfValues = deltaQ.cdfValues;
228     std::transform(result.cdfValues.begin(), result.cdfValues.end(),
229 result.cdfValues.begin(), [constant](double value) { return value
230 * constant; });
231
232     return result;
233 }
234
235 template <typename BinaryOperation>
236 DeltaQ applyBinaryOperation(const DeltaQ &lhs, const DeltaQ &rhs,
237 BinaryOperation op)
238 {
239     const DeltaQ &highestDeltaQ = (lhs > rhs) ? lhs : rhs;
240     const DeltaQ &otherDeltaQ = (lhs > rhs) ? rhs : lhs;
241
242     std::vector<double> resultingCdf;
243     resultingCdf.reserve(highestDeltaQ.getBins());
244
245     for (size_t i = 0; i < highestDeltaQ.getBins(); i++) {
246         double result = op(highestDeltaQ.cdfAt(i), otherDeltaQ.cdfAt(i));
247         resultingCdf.push_back(result);
248     }
249
250     return {highestDeltaQ.getBinWidth(), resultingCdf, false};
251 }
252
253 DeltaQ operator*(double constant, const DeltaQ &deltaQ)
254 {
255     return deltaQ * constant;
256 }
257
258 DeltaQ operator+(const DeltaQ &lhs, const DeltaQ &rhs)
259 {
260     return applyBinaryOperation(lhs, rhs, std::plus<>());
261 }
```

```

257
258 DeltaQ operator-(const DeltaQ &lhs, const DeltaQ &rhs)
259 {
260     return applyBinaryOperation(lhs, rhs, std::minus<>());
261 }
262
263 DeltaQ operator*(const DeltaQ &lhs, const DeltaQ &rhs)
264 {
265     return applyBinaryOperation(lhs, rhs, std::multiplies<>());
266 }
267
268 std::string DeltaQ::toString() const
269 {
270     std::ostringstream oss;
271     oss << "<";
272
273     // Iterate through CDF values to construct the string
274     for (size_t i = 0; i < pdfValues.size(); ++i) {
275         const double bin = (i + 1) * binWidth;
276         oss << "(" << std::fixed << std::setprecision(7) << bin << ",
277         " << std::setprecision(7) << pdfValues[i] << ")";
278         if (i < pdfValues.size() - 1) {
279             oss << ", ";
280         }
281     }
282     oss << ">";
283     return oss.str();
284 }

```

C.4.3 DeltaQOperations.cpp

```

1  #include "DeltaQOperations.h"
2  #include "DeltaQ.h"
3  #include <fftw3.h>
4  #include <iostream>
5  #include <math.h>
6  #include <mutex>
7  #include <vector>
8  DeltaQ rebin(const DeltaQ &source, double targetBinWidth)
9  {
10     double originalBinWidth = source.getBinWidth();
11
12     if (std::abs(originalBinWidth - targetBinWidth) < 1e-9) {
13         return source; // Already same bin width
14     }
15
16     if (targetBinWidth < originalBinWidth) {
17         throw std::invalid_argument("Target bin width must be greater
18         than or equal to original.");
19     }
20
21     int factor = static_cast<int>(std::round(targetBinWidth /
22     originalBinWidth));
23     const auto &originalPdf = source.getPdfValues();

```

```

22
23     int newNumBins = static_cast<int>(std::ceil(originalPdf.size() /
static_cast<double>(factor)));
24     std::vector<double> newPdf(newNumBins, 0.0);
25
26     for (size_t i = 0; i < originalPdf.size(); ++i) {
27         newPdf[i / factor] += originalPdf[i];
28     }
29
30     return DeltaQ(targetBinWidth, newPdf, true);
31 }
32
33 DeltaQ convolve(const DeltaQ &lhs, const DeltaQ &rhs)
34 {
35     double commonBinWidth = std::max(lhs.getBinWidth(), rhs.
getBinWidth());
36
37     if (lhs == DeltaQ()) {
38         return rhs;
39     }
40     if (rhs == DeltaQ()) {
41         return lhs;
42     }
43
44     DeltaQ lhsRebinned = rebin(lhs, commonBinWidth);
45     DeltaQ rhsRebinned = rebin(rhs, commonBinWidth);
46
47     const auto &lhsPdf = lhsRebinned.getPdfValues();
48     const auto &rhsPdf = rhsRebinned.getPdfValues();
49
50     int resultSize = lhsPdf.size() + rhsPdf.size() - 1;
51     std::vector<double> resultPdf(resultSize, 0.0);
52
53     for (size_t i = 0; i < lhsPdf.size(); ++i) {
54         for (size_t j = 0; j < rhsPdf.size(); ++j) {
55             resultPdf[i + j] += lhsPdf[i] * rhsPdf[j];
56         }
57     }
58
59     return DeltaQ(commonBinWidth, resultPdf, true);
60 }
61
62 // Inspired by https://github.com/jeremyfix/FFTConvolution/blob/master
/Convolution/src/convolution_fftw.h
63 DeltaQ convolveFFT(const DeltaQ &lhs, const DeltaQ &rhs)
64 {
65     if (lhs == DeltaQ()) {
66         return rhs;
67     }
68     if (rhs == DeltaQ()) {
69         return lhs;
70     }
71
72     // Find a common bin width and rebin accordingly
73     double commonBinWidth = std::max(lhs.getBinWidth(), rhs.
getBinWidth());

```



```

74
75     DeltaQ lhsRebinned = rebin(lhs, commonBinWidth);
76     DeltaQ rhsRebinned = rebin(rhs, commonBinWidth);
77
78     const auto &lhsPdf = lhsRebinned.getPdfValues();
79     const auto &rhsPdf = rhsRebinned.getPdfValues();
80
81     // Find the power of 2 nearest to the convolution size
82     size_t lhsSize = lhsPdf.size();
83     size_t rhsSize = rhsPdf.size();
84     size_t convSize = lhsSize + rhsSize - 1;
85     size_t fftSize = 1;
86     while (fftSize < convSize)
87         fftSize <<= 1;
88
89     // Pad pdf with zeroes until end of PDF
90     std::vector<double> lhsPadded(fftSize, 0.0);
91     std::vector<double> rhsPadded(fftSize, 0.0);
92     std::copy(lhsPdf.begin(), lhsPdf.end(), lhsPadded.begin());
93     std::copy(rhsPdf.begin(), rhsPdf.end(), rhsPadded.begin());
94     static std::mutex fftw_mutex;
95     {
96         std::lock_guard<std::mutex> lock(fftw_mutex);
97         fftw_complex *lhsFreq = (fftw_complex *)fftw_malloc(sizeof(
fftw_complex) * (fftSize / 2 + 1));
98         fftw_complex *rhsFreq = (fftw_complex *)fftw_malloc(sizeof(
fftw_complex) * (fftSize / 2 + 1));
99         double *lhsTime = lhsPadded.data();
100        double *rhsTime = rhsPadded.data();
101
102        // Transform real input to complex output
103        fftw_plan planLhs = fftw_plan_dft_r2c_1d(fftSize, lhsTime,
lhsFreq, FFTW_ESTIMATE);
104        fftw_plan planRhs = fftw_plan_dft_r2c_1d(fftSize, rhsTime,
rhsFreq, FFTW_ESTIMATE);
105        fftw_execute(planLhs);
106        fftw_execute(planRhs);
107
108        // Do complex multiplication, r: ac - bd i : ad + bc
109        fftw_complex *resultFreq = (fftw_complex *)fftw_malloc(sizeof(
fftw_complex) * (fftSize / 2 + 1));
110        for (size_t i = 0; i < fftSize / 2 + 1; ++i) {
111            double a = lhsFreq[i][0], b = lhsFreq[i][1];
112            double c = rhsFreq[i][0], d = rhsFreq[i][1];
113            resultFreq[i][0] = a * c - b * d;
114            resultFreq[i][1] = a * d + b * c;
115        }
116
117        // Invert from complex plane to real plane
118        std::vector<double> resultTime(fftSize);
119        fftw_plan planInv = fftw_plan_dft_c2r_1d(fftSize, resultFreq,
resultTime.data(), FFTW_ESTIMATE);
120        fftw_execute(planInv);
121
122        for (auto &val : resultTime)
123            val /= fftSize;

```

```

124         resultTime.resize(convSize);
125
126         fftw_destroy_plan(planLhs);
127         fftw_destroy_plan(planRhs);
128         fftw_destroy_plan(planInv);
129         fftw_free(lhsFreq);
130         fftw_free(rhsFreq);
131         fftw_free(resultFreq);
132         return {lhsRebinned.getBinWidth(), resultTime, true};
133     }
134 }
135
136
137 DeltaQ convolveN(const std::vector<DeltaQ> &deltaQs)
138 {
139     if (deltaQs.empty())
140         return DeltaQ();
141
142     DeltaQ result = deltaQs[0];
143     for (size_t i = 1; i < deltaQs.size(); ++i) {
144         result = convolveFFT(result, deltaQs[i]);
145     }
146     return result;
147 }
148
149 DeltaQ probabilisticChoice(const std::vector<double> &probabilities,
150     const std::vector<DeltaQ> &deltaQs)
151 {
152     std::vector<DeltaQ> nonEmpty;
153     std::vector<double> effectiveProbs;
154     double commonBinWidth = 0.0;
155
156     for (size_t i = 0; i < deltaQs.size(); ++i) {
157         if (deltaQs[i] == DeltaQ()) {
158             continue;
159         }
160         nonEmpty.push_back(deltaQs[i]);
161         effectiveProbs.push_back(probabilities[i]);
162         commonBinWidth = std::max(commonBinWidth, deltaQs[i].
163             getBinWidth());
164     }
165
166     if (nonEmpty.empty()) {
167         return DeltaQ();
168     }
169
170     for (auto &dq : nonEmpty) {
171         dq = rebin(dq, commonBinWidth);
172     }
173
174     std::vector<DeltaQ> scaledDeltaQs;
175     for (size_t i = 0; i < nonEmpty.size(); ++i) {
176         scaledDeltaQs.push_back(nonEmpty[i] * effectiveProbs[i]);
177     }
178
179     DeltaQ result = scaledDeltaQs[0];

```

```
178     for (size_t i = 1; i < scaledDeltaQs.size(); ++i) {
179         result = result + scaledDeltaQs[i];
180     }
181
182     return result;
183 }
184
185 DeltaQ firstToFinish(const std::vector<DeltaQ> &deltaQs)
186 {
187     std::vector<DeltaQ> nonEmpty;
188     double commonBinWidth = 0.0;
189
190     for (const auto &dq : deltaQs) {
191         if (dq == DeltaQ()) {
192             continue;
193         }
194         nonEmpty.push_back(dq);
195         commonBinWidth = std::max(commonBinWidth, dq.getBinWidth());
196     }
197
198     if (nonEmpty.empty()) {
199         return DeltaQ();
200     }
201
202     for (auto &dq : nonEmpty) {
203         dq = rebin(dq, commonBinWidth);
204     }
205
206     const int largestSize = chooseLongestDeltaQSize(nonEmpty);
207     std::vector<double> resultingCdf;
208
209     for (int i = 0; i < largestSize; ++i) {
210         double sumAtI = 0;
211         double productAtI = 1;
212         for (const auto &dq : nonEmpty) {
213             const double cdfAtI = dq.cdfAt(i);
214             sumAtI += cdfAtI;
215             productAtI *= cdfAtI;
216         }
217         resultingCdf.push_back(sumAtI - productAtI);
218     }
219     return {commonBinWidth, resultingCdf, false};
220 }
221
222 DeltaQ allToFinish(const std::vector<DeltaQ> &deltaQs)
223 {
224     std::vector<DeltaQ> nonEmpty;
225     double commonBinWidth = 0.0;
226
227     for (const auto &dq : deltaQs) {
228         if (dq == DeltaQ()) {
229             continue;
230         }
231         nonEmpty.push_back(dq);
232         commonBinWidth = std::max(commonBinWidth, dq.getBinWidth());
233     }
```

```

234
235     if (nonEmpty.empty()) {
236         return DeltaQ();
237     }
238
239     for (auto &dq : nonEmpty) {
240         dq = rebin(dq, commonBinWidth);
241     }
242
243     DeltaQ result = nonEmpty[0];
244     for (size_t i = 1; i < nonEmpty.size(); ++i) {
245         result = result * nonEmpty[i];
246     }
247
248     return result;
249 }
250
251 int chooseLongestDeltaQSize(const std::vector<DeltaQ> &deltaQs)
252 {
253     int highestSize = 0;
254     for (const DeltaQ &deltaQ : deltaQs) {
255         if (deltaQ.getBins() > highestSize) {
256             highestSize = deltaQ.getBins();
257         }
258     }
259     return highestSize;
260 }

```

C.4.4 Snapshot.cpp

```

1  #include "Snapshot.h"
2
3  void Snapshot::addObservedDeltaQ(std::uint64_t time, const DeltaQ &
4      deltaQ, const std::vector<Bound> &bounds)
5  {
6      DeltaQRepr repr = {time, deltaQ, bounds};
7      observedDeltaQs[time] = repr;
8  }
9
10 void Snapshot::removeOldestObservedDeltaQ()
11 {
12     if (getObservedSize() == 0)
13         return;
14     observedDeltaQs.erase(observedDeltaQs.begin());
15 }
16
17 void Snapshot::addCalculatedDeltaQ(std::uint64_t time, const DeltaQ &
18     deltaQ, const std::vector<Bound> &bounds)
19 {
20     DeltaQRepr repr = {time, deltaQ, bounds};
21     calculatedDeltaQs[time] = repr;
22 }
23
24 DeltaQ Snapshot::getOldestCalculatedDeltaQ() const
25 {

```

```

24     return calculatedDeltaQs.begin()->second.deltaQ;
25 }
26
27 DeltaQ Snapshot::getOldestObservedDeltaQ() const
28 {
29     return observedDeltaQs.begin()->second.deltaQ;
30 }
31
32 void Snapshot::removeOldestCalculatedDeltaQ()
33 {
34     if (getCalculatedSize() == 0)
35         return;
36
37     calculatedDeltaQs.erase(calculatedDeltaQs.begin());
38 }
39
40 void Snapshot::resizeTo(size_t newSize)
41 {
42     int observedSize = getObservedSize();
43     int calculatedSize = getCalculatedSize();
44
45     // Don't shrink if not needed
46     if (observedSize <= newSize && calculatedSize <= newSize)
47         return;
48
49     if (observedSize > newSize) {
50         int toObserved = observedSize - newSize;
51         auto endIt = std::next(observedDeltaQs.begin(), toObserved);
52         observedDeltaQs.erase(observedDeltaQs.begin(), endIt);
53     }
54
55     if (calculatedSize > newSize) {
56         int toCalculated = calculatedSize - newSize;
57         auto endItC = std::next(calculatedDeltaQs.begin(),
toCalculated);
58         calculatedDeltaQs.erase(calculatedDeltaQs.begin(), endItC);
59     }
60
61     if (QTAs.size() > newSize) {
62         int toSize = QTAs.size() - newSize;
63         auto endIt = std::next(QTAs.begin(), toSize);
64         QTAs.erase(QTAs.begin(), endIt);
65     }
66 }
67 void Snapshot::addQTA(uint64_t time, const QTA &qta)
68 {
69     QTAs[time] = qta;
70 }
71
72 std::optional<DeltaQRepr> Snapshot::getObservedDeltaQAtTime(std::
uint64_t time)
73 {
74     auto it = observedDeltaQs.find(time);
75     if (it != observedDeltaQs.end()) {
76         return it->second;
77     }

```

```
78     return std::nullopt;
79 }
80
81 std::optional<DeltaQRepr> Snapshot::getCalculatedDeltaQAtTime(std::
    uint64_t time)
82 {
83     auto it = calculatedDeltaQs.find(time);
84     if (it != calculatedDeltaQs.end()) {
85         return it->second;
86     }
87     return std::nullopt;
88 }
89
90 std::size_t Snapshot::getObservedSize() const
91 {
92     return observedDeltaQs.size();
93 }
94
95 std::size_t Snapshot::getCalculatedSize() const
96 {
97     return calculatedDeltaQs.size();
98 }
99
100 std::vector<DeltaQRepr> Snapshot::getObservedDeltaQs() const &
101 {
102     auto obs = std::vector<DeltaQRepr>();
103     for (const auto &s : observedDeltaQs)
104         obs.emplace_back(s.second);
105     return obs;
106 }
107
108 std::vector<DeltaQRepr> Snapshot::getCalculatedDeltaQs() const &
109 {
110     auto calc = std::vector<DeltaQRepr>();
111     for (const auto &s : calculatedDeltaQs)
112         calc.emplace_back(s.second);
113     return calc;
114 }
115
116 std::vector<QTA> Snapshot::getQTAs() const &
117 {
118     auto QTAsVec = std::vector<QTA>();
119     for (const auto &q : QTAs) {
120         QTAsVec.emplace_back(q.second);
121     }
122     return QTAsVec;
123 }
124
125 void Snapshot::setName(const std::string &name)
126 {
127     observableName = name;
128 }
129
130 std::string Snapshot::getName() &
131 {
132     return observableName;
```

133 }

C.4.5 TriggerManager.cpp

```

1  #include "TriggerManager.h"
2
3  #include <algorithm>
4
5  TriggerManager::Trigger::Trigger(TriggerType t, TriggerDefs::Condition
   c, TriggerDefs::Action a, bool e)
6      : type(t)
7      , condition(std::move(c))
8      , action(std::move(a))
9      , enabled(e)
10 {
11 }
12
13 void TriggerManager::addTrigger(TriggerType type, TriggerDefs::
   Condition condition, TriggerDefs::Action action, bool enabled)
14 {
15     triggers_.emplace_back(type, std::move(condition), std::move(
   action), enabled);
16 }
17
18 std::vector<TriggerManager::Trigger> TriggerManager::getTriggersByType
   (TriggerType type)
19 {
20     std::vector<Trigger> result;
21     for (auto &trigger : triggers_) {
22         if (trigger.type == type) {
23             result.push_back(trigger);
24         }
25     }
26     return result;
27 }
28
29 void TriggerManager::evaluate(const DeltaQ &dq, const QTA &qta, std:::
   uint64_t time) const
30 {
31     for (const auto &trigger : triggers_) {
32         if (trigger.enabled && trigger.condition(dq, qta)) {
33             trigger.action(dq, qta, time);
34         }
35     }
36 }
37
38 void TriggerManager::removeTriggersByType(TriggerType type)
39 {
40     triggers_.erase(std::remove_if(triggers_.begin(), triggers_.end(),
   [type](const auto &trigger) { return trigger.type == type; }),
   triggers_.end());
41 }
42
43 void TriggerManager::clearAllTriggers()
44 {

```

```

45     triggers_.clear();
46 }
47
48 void TriggerManager::setTriggersEnabled(TriggerType type, bool enabled
49 )
50 {
51     for (auto &trigger : triggers_) {
52         if (trigger.type == type) {
53             trigger.enabled = enabled;
54         }
55     }
56 std::vector<TriggerManager::Trigger> TriggerManager::getAllTriggers()
57 const
58 {
59     std::vector<Trigger> result;
60     for (const auto &trigger : triggers_) {
61         result.push_back(trigger);
62     }
63     return result;
64 }

```

C.4.6 Triggers.cpp

```

1  #include "Triggers.h"
2
3  namespace TriggerDefs
4  {
5      namespace Conditions
6      {
7          Condition SampleLimit(int maxSamples)
8          {
9              return [maxSamples](const DeltaQ &dq, const QTA &) { return dq
10 .getTotalSamples() > maxSamples; };
11          }
12
13          Condition QTABounds()
14          {
15              return [](const DeltaQ &dq, const QTA &qta) {
16                  const QTA &dqQta = dq.getQTA();
17                  return dqQta.perc_25 > qta.perc_25 || dqQta.perc_50 > qta.
18 perc_50 || dqQta.perc_75 > qta.perc_75 || dqQta.cdfMax < qta.
19 cdfMax;
20              };
21          }
22
23          Condition FailureRate(double threshold)
24          {
25              return [threshold](const DeltaQ &dq, const QTA &) { return dq.
26 getQTA().cdfMax < threshold; };
27          }
28      }
29  }
30
31  namespace Actions
32  {

```



```

28     Action LogToConsole(const std::string &message)
29     {
30         return [message](const DeltaQ &, const QTA &, std::uint64_t) {
31             std::cout << "TRIGGER: " << message << "\n"; };
32     }
33
34     Action notify()
35     {
36         return [] (const DeltaQ &dq, const QTA &qta, std::uint64_t) {
37             };
38     }
39
40     Action SaveSnapshot(const std::string &filename)
41     {
42         return [] (const DeltaQ &dq, const QTA &, std::uint64_t) { };
43     }

```

C.5 parser

C.5.1 SystemBuilder.cpp

```

1  #include "SystemBuilder.h"
2  #include <memory>
3  #include <stdexcept>
4
5  #include <locale>
6  System SystemBuilderVisitor::getSystem() const
7  {
8      return system;
9  }
10
11 void SystemBuilderVisitor::checkForCycles() const
12 {
13     std::set<std::string> visited;
14     std::set<std::string> recursionStack;
15
16     for (const auto &[node, _] : dependencies) {
17         if (hasCycle(node, visited, recursionStack)) {
18             throw std::invalid_argument("Cycle detected in system
19             definition involving: " + node);
20         }
21     }
22 }
23
24 bool SystemBuilderVisitor::hasCycle(const std::string &node, std::set<
25     std::string> &visited, std::set<std::string> &recursionStack)
26     const
27 {
28     if (recursionStack.find(node) != recursionStack.end()) {
29         return true;
30     }
31     if (visited.find(node) != visited.end()) {

```

```
29         return false;
30     }
31
32     visited.insert(node);
33     recursionStack.insert(node);
34
35     if (dependencies.find(node) != dependencies.end()) {
36         for (const auto &neighbor : dependencies.at(node)) {
37             if (hasCycle(neighbor, visited, recursionStack)) {
38                 return true;
39             }
40         }
41     }
42
43     recursionStack.erase(node);
44     return false;
45 }
46
47 std::any SystemBuilderVisitor::visitStart(parser::DQGrammarParser::
48     StartContext *context)
49 {
50     for (const auto definition : context->definition()) {
51         visitDefinition(definition);
52     }
53
54     if (context->system()) {
55         visitSystem(context->system());
56     }
57
58     system.setOutcomes(outcomes);
59     system.setProbes(probes);
60     system.setOperators(operators);
61     /*
62     for (const auto &[name, link] : definitionLinks) {
63         std::cout << name << " [ ";
64         for (auto &name2 : link) {
65             std::cout << name2 << " ";
66         }
67         std::cout << "]\n";
68     }
69
70     for (const auto &[name, op] : operatorLinks) {
71         std::cout << name << " [ ";
72         for (auto link : op) {
73             std::cout << " [ ";
74             for (auto lill : link) {
75                 std::cout << lill << " ";
76             }
77             std::cout << "]\n";
78         }
79         std::cout << "]\n";
80     }
81     */
82     checkForCycles();
83     return nullptr;
84 }
```

```

84
85 std::any SystemBuilderVisitor::visitDefinition(parser::DQGrammarParser
    ::DefinitionContext *context)
86 {
87     std::string probeName = context->IDENTIFIER()->getText();
88     if (std::find(definedProbes.begin(), definedProbes.end(),
        probeName) != definedProbes.end()) {
89         throw std::invalid_argument("Probe has already been defined");
90     }
91     if (allNames.find(probeName) != allNames.end()) {
92         throw std::invalid_argument("Duplicate name detected: " +
        probeName);
93     }
94     allNames.insert(probeName);
95     currentlyBuildingProbe = probeName;
96
97     const auto chainComponents = std::any_cast<std::vector<std::
        shared_ptr<Observable>>>(visitComponent_chain(context->
        component_chain()));
98     std::vector<std::shared_ptr<Observable>> probeCausalLinks;
99     std::vector<std::string> links;
100    for (auto &comp : chainComponents) {
101        probeCausalLinks.push_back(comp);
102        links.push_back(comp->getName());
103    }
104
105    const auto probe = std::make_shared<Probe>(probeName,
        probeCausalLinks);
106
107    probes[probeName] = probe;
108    definedProbes.push_back(probeName);
109    definitionLinks[probeName] = links;
110    currentlyBuildingProbe = "";
111    return nullptr;
112 }
113
114 std::any SystemBuilderVisitor::visitSystem(parser::DQGrammarParser::
    SystemContext *context)
115 {
116     const auto chainComponents = std::any_cast<std::vector<std::
        shared_ptr<Observable>>>(visitComponent_chain(context->
        component_chain()));
117
118     std::vector<std::string> links;
119     for (auto &comp : chainComponents) {
120         links.push_back(comp->getName());
121     }
122
123     systemLinks = links;
124
125     return nullptr;
126 }
127
128 std::any SystemBuilderVisitor::visitComponent(parser::DQGrammarParser
    ::ComponentContext *context)
129 {

```

```

130     if (context->behaviorComponent()) {
131         return visitBehaviorComponent(context->behaviorComponent());
132     } else if (context->probeComponent()) {
133         return visitProbeComponent(context->probeComponent());
134     } else if (context->outcome()) {
135         return visitOutcome(context->outcome());
136     }
137     return nullptr;
138 }
139
140 std::any SystemBuilderVisitor::visitBehaviorComponent(parser::
    DQGrammarParser::BehaviorComponentContext *context)
141 {
142     const std::string typeStr = context->BEHAVIOR_TYPE()->getText();
143     std::string name = context->IDENTIFIER()->getText();
144
145     if (operators.find(name) != operators.end()) {
146         return std::dynamic_pointer_cast<Observable>(operators[name]);
147     }
148     if (allNames.find(name) != allNames.end()) {
149         throw std::invalid_argument("Duplicate name detected: " + name
150     );
151     }
152
153     allNames.insert(name);
154     OperatorType type;
155     if (typeStr == "a")
156         type = OperatorType::ATF;
157     else if (typeStr == "f")
158         type = OperatorType::FTF;
159     else if (typeStr == "p")
160         type = OperatorType::PRB;
161     else
162         throw std::invalid_argument("Unknown operator type: " +
163     typeStr);
164
165     const auto op = std::make_shared<Operator>(name, type);
166     if (type == OperatorType::PRB && context->probability_list()) {
167         const auto probabilities = std::any_cast<std::vector<double>>>(
168     visitProbability_list(context->probability_list()));
169         op->setProbabilities(probabilities);
170     } else if (type == OperatorType::PRB && !context->probability_list
171     ()) {
172         throw std::invalid_argument("A probabilistic operator must
173     have probabilities");
174     } else if (type != OperatorType::PRB && context->probability_list
175     ()) {
176         throw std::invalid_argument("A non probabilistic operator
177     cannot have probabilities");
178     }
179
180     std::vector<std::vector<std::shared_ptr<Observable>>>
181     operatorPtrLinks;
182
183     if (context->component_list()) {

```

```

176         auto childrenChains = std::any_cast<std::vector<std::vector<
std::shared_ptr<Observable>>>>(visitComponent_list(context->
component_list()));
177
178         for (auto &chain : childrenChains) {
179             if (!chain.empty()) {
180                 operatorPtrLinks.push_back(chain);
181             }
182         }
183     }
184
185     op->setCausalLinks(operatorPtrLinks);
186
187     if (context->component_list()) {
188         auto childrenChains = std::any_cast<std::vector<std::vector<
std::shared_ptr<Observable>>>>(visitComponent_list(context->
component_list()));
189
190         std::vector<std::vector<std::string>> childrenLinks;
191
192         for (auto &chain : childrenChains) {
193             if (!chain.empty()) {
194                 std::vector<std::string> chainNames;
195                 for (auto &comp : chain) {
196                     chainNames.push_back(comp->getName());
197                 }
198                 childrenLinks.push_back(chainNames);
199             }
200         }
201
202         operatorLinks[name] = childrenLinks;
203     }
204
205     operators[name] = op;
206     return std::dynamic_pointer_cast<Observable>(op);
207 }
208 std::any SystemBuilderVisitor::visitProbeComponent(parser::
DQGrammarParser::ProbeComponentContext *context)
209 {
210     std::string name = context->IDENTIFIER()->getText();
211
212     if (!currentlyBuildingProbe.empty()) {
213         dependencies[currentlyBuildingProbe].push_back(name);
214     }
215
216     if (probes.find(name) != probes.end()) {
217         return std::dynamic_pointer_cast<Observable>(probes[name]);
218     }
219
220     // Create a stub probe (may be fleshed out later)
221     auto probe = std::make_shared<Probe>(name);
222     probes[name] = probe;
223     return std::dynamic_pointer_cast<Observable>(probe);
224 }
225

```

```

226 std::any SystemBuilderVisitor::visitProbability_list(parser::
    DQGrammarParser::Probability_listContext *context)
227 {
228     std::locale::global(std::locale("C"));
229
230     std::vector<double> probabilities;
231     for (auto num : context->NUMBER()) {
232         probabilities.push_back(std::stod(num->getText()));
233     }
234     return probabilities;
235 }
236
237 std::any SystemBuilderVisitor::visitComponent_list(parser::
    DQGrammarParser::Component_listContext *context)
238 {
239     std::vector<std::vector<std::shared_ptr<Observable>>>
    componentsChains;
240
241     for (auto chainCtx : context->component_chain()) {
242         auto chain = std::any_cast<std::vector<std::shared_ptr<
    Observable>>>(visitComponent_chain(chainCtx));
243         componentsChains.push_back(chain);
244     }
245
246     return componentsChains;
247 }
248
249 std::any SystemBuilderVisitor::visitComponent_chain(parser::
    DQGrammarParser::Component_chainContext *context)
250 {
251     std::vector<std::shared_ptr<Observable>> components;
252
253     for (auto compCtx : context->component()) {
254         auto component = std::any_cast<std::shared_ptr<Observable>>(
    visitComponent(compCtx));
255         components.push_back(component);
256     }
257     return components;
258 }
259
260 std::any SystemBuilderVisitor::visitOutcome(parser::DQGrammarParser::
    OutcomeContext *context)
261 {
262     std::string name = context->IDENTIFIER()->getText();
263
264     if (outcomes.find(name) != outcomes.end()) {
265         return std::dynamic_pointer_cast<Observable>(outcomes[name]);
266     }
267     if (allNames.find(name) != allNames.end()) {
268         throw std::invalid_argument("Duplicate name detected: " + name
    );
269     }
270     allNames.insert(name);
271
272     auto outcome = std::make_shared<Outcome>(name);
273     outcomes[name] = outcome;

```

```

274     return std::dynamic_pointer_cast<Observable>(outcome);
275 }

```

C.5.2 SystemParserInterface.cpp

```

1  #include "SystemParserInterface.h"
2  #include "SystemErrorListener.h"
3  #include <exception>
4  #include <fstream>
5  #include <iostream>
6  #include <sstream>
7  #include <stdexcept>
8
9  /**
10 * @brief Parses a system definition from a file.
11 * @param filename Path to the file containing system definition.
12 * @return Optional containing the parsed System if successful,
13 *         nullptr on error.
14 * @throws std::invalid_argument if parsing fails.
15 */
16 std::optional<System> SystemParserInterface::parseFile(const std::
17 string &filename)
18 {
19     std::ifstream file(filename);
20     if (!file) {
21         std::cerr << "Error: Could not open file: " << filename << std
22 ::endl;
23         return std::nullptr;
24     }
25
26     // Read entire file content
27     std::stringstream buffer;
28     buffer << file.rdbuf();
29     std::string content = buffer.str();
30
31     antlr4::ANTLRInputStream input(content);
32     try {
33         return parseInternal(input);
34     } catch (std::exception &e) {
35         throw std::invalid_argument(e.what());
36     }
37 }
38
39 /**
40 * @brief Parses a system definition from a string.
41 * @param inputStr String containing system definition.
42 * @return Optional containing the parsed System if successful,
43 *         nullptr on error.
44 * @throws std::invalid_argument if parsing fails.
45 */
46 std::optional<System> SystemParserInterface::parseString(const std::
47 string &inputStr)
48 {
49     antlr4::ANTLRInputStream input(inputStr);
50     try {

```

```

46         return parseInternal(input);
47     } catch (std::exception &e) {
48         throw std::invalid_argument(e.what());
49     }
50 }
51
52 /**
53  * @brief Internal parsing implementation using ANTLR.
54  * @param input ANTLR input stream containing system definition.
55  * @return Optional containing the parsed System if successful,
56  *         nullptr on error.
57  * @throws std::invalid_argument if parsing fails.
58  */
59 std::optional<System> SystemParserInterface::parseInternal(antlr4::
60     ANTLRInputStream &input)
61 {
62     // Initialize lexer and parser
63     parser::DQGrammarLexer lexer(&input);
64     antlr4::CommonTokenStream tokens(&lexer);
65     parser::DQGrammarParser parser(&tokens);
66
67     // Configure error handling
68     SystemErrorListener errorListener;
69     lexer.removeErrorListeners();
70     parser.removeErrorListeners();
71     lexer.addErrorListener(&errorListener);
72     parser.addErrorListener(&errorListener);
73
74     try {
75         // Parse and build system
76         auto tree = parser.start();
77         SystemBuilderVisitor visitor;
78         visitor.visitStart(tree);
79         return visitor.getSystem();
80     } catch (std::exception &e) {
81         throw std::invalid_argument(e.what());
82     }
83 }

```

C.6 server

C.6.1 Server.cpp

```

1  #include "Server.h"
2  #include "../Application.h"
3  #include <arpa/inet.h>
4  #include <cstdint>
5  #include <cstring>
6  #include <fcntl.h>
7  #include <iostream>
8  #include <regex>
9  #include <signal.h>
10 #include <sys/socket.h>
11 #include <unistd.h>

```



```

12 #define TIMEOUT "to"
13 #define EXEC_OK "ok"
14 #define FAIL "fa"
15
16 /**
17  * @brief Constructs the Server and registers system observer.
18  * @param port The TCP port to listen on.
19  */
20 Server::Server(int port)
21     : port(port)
22     , server_fd(0)
23     , new_socket(0)
24     , server_started(false)
25 {
26     Application::getInstance().addObserver([this]() { this->
updateSystem(); });
27
28     // Start worker thread (this can run independently)
29     workerThread = std::thread([this]() {
30         while (!shutdownWorker) {
31             std::unique_lock lock(queueMutex);
32             queueCond.wait(lock, [this] { return !sampleQueue.empty()
|| shutdownWorker; });
33
34             while (!sampleQueue.empty()) {
35                 auto [name, sample] = sampleQueue.front();
36                 sampleQueue.pop();
37                 lock.unlock();
38
39                 if (system) {
40                     system->addSample(name, sample);
41                 }
42
43                 lock.lock();
44             }
45         }
46     });
47 }
48
49 /**
50  * @brief Destructor cleans up sockets and joins threads.
51  */
52 Server::~Server()
53 {
54     std::lock_guard<std::mutex> lock(erlangMutex);
55     if (erlang_socket > 0) {
56         close(erlang_socket);
57         erlang_socket = -1;
58     }
59     close(new_socket);
60     close(server_fd);
61     if (serverThread.joinable())
62         serverThread.join();
63 }
64 /**
65  * @brief Updates the system reference from Application.

```

```
66  */
67 void Server::updateSystem()
68 {
69     system = Application::getInstance().getSystem();
70 }
71
72 /**
73  * @brief Main server loop handling client connections.
74  */
75 void Server::run()
76 {
77     server_fd = socket(AF_INET, SOCK_STREAM, 0);
78     if (server_fd == 0) {
79         perror("Socket failed");
80         return;
81     }
82
83     // Configure socket options
84     int opt = 1;
85     setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))
86 ;
87     setsockopt(server_fd, SOL_SOCKET, SO_REUSEPORT, &opt, sizeof(opt))
88 ;
89
90     // Set large buffer sizes
91     int bufSize = 1 << 20; // 1MB
92     setsockopt(server_fd, SOL_SOCKET, SO_RCVBUF, &bufSize, sizeof(
93 bufSize));
94     setsockopt(server_fd, SOL_SOCKET, SO_SNDBUF, &bufSize, sizeof(
95 bufSize));
96
97     // Bind socket to specified IP
98     address.sin_family = AF_INET;
99     address.sin_port = htons(port);
100
101     // Parse IP address
102     if (server_ip == "0.0.0.0" || server_ip.empty()) {
103         address.sin_addr.s_addr = INADDR_ANY;
104     } else {
105         if (inet_pton(AF_INET, server_ip.c_str(), &address.sin_addr)
106 <= 0) {
107             std::cerr << "Invalid IP address: " << server_ip << std::
108 endl;
109             close(server_fd);
110             return;
111         }
112     }
113
114     if (bind(server_fd, (struct sockaddr *)&address, sizeof(address))
115 < 0) {
116         perror("Bind failed");
117         close(server_fd);
118         return;
119     }
120
121     // Set non-blocking mode
```

```

115     fcntl(server_fd, F_SETFL, O_NONBLOCK);
116
117     if (listen(server_fd, SOMAXCONN) < 0) {
118         perror("Listen failed");
119         close(server_fd);
120         return;
121     }
122
123     std::cout << "Server running on " << server_ip << ":" << port <<
std::endl;
124     running = true;
125     server_started = true;
126
127     while (running) {
128         int addrlen = sizeof(address);
129         client_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t *)&addrlen);
130
131         if (client_socket < 0) {
132             if (errno == EWOULDBLOCK || errno == EAGAIN) {
133                 std::this_thread::sleep_for(std::chrono::milliseconds
(100));
134                 cleanupThreads();
135                 continue;
136             }
137             if (!running)
138                 break; // Server was stopped
139             perror("Accept failed");
140             continue;
141         }
142
143         std::lock_guard<std::mutex> lock(clientsMutex);
144         clientThreads.emplace_back(&Server::handleClient, this,
client_socket);
145     }
146
147     // Cleanup
148     close(server_fd);
149     server_fd = 0;
150     cleanupThreads();
151     server_started = false;
152 }
153
154 /**
155  * @brief Starts the server on specified IP and port.
156  * @param ip The IP address to bind to (default: "0.0.0.0" for all
interfaces)
157  * @param port The port to listen on
158  * @return true if server started successfully
159  */
160 bool Server::startServer(const std::string &ip, int port)
161 {
162     if (server_started) {
163         std::cerr << "Server already running. Stop it first." << std:::
endl;
164         return false;

```

```

165     }
166
167     this->server_ip = ip;
168     this->port = port;
169
170     serverThread = std::thread(&Server::run, this);
171
172     // Wait a bit to see if server started successfully
173     std::this_thread::sleep_for(std::chrono::milliseconds(100));
174     return server_started;
175 }
176
177 /**
178  * @brief Stops the server and closes all sockets.
179  */
180 void Server::stopServer()
181 {
182     if (!server_started) {
183         std::cout << "Server not running." << std::endl;
184         return;
185     }
186
187     running = false;
188     server_started = false;
189
190     // Close server socket to break accept loop
191     if (server_fd > 0) {
192         close(server_fd);
193         server_fd = 0;
194     }
195
196     // Join server thread
197     if (serverThread.joinable()) {
198         serverThread.join();
199     }
200
201     // Cleanup client threads
202     cleanupThreads();
203
204     std::cout << "Server stopped." << std::endl;
205 }
206
207 /**
208  * @brief Sets the Erlang endpoint for connections.
209  * @param ip Erlang server IP address
210  * @param port Erlang server port
211  * @return true if endpoint set successfully
212  */
213 bool Server::setErlangEndpoint(const std::string &ip, int port)
214 {
215     std::lock_guard<std::mutex> lock(erlangMutex);
216
217     // Close existing connection if any
218     if (erlang_socket > 0) {
219         close(erlang_socket);
220         erlang_socket = -1;

```

```

221     }
222
223     erlang_ip = ip;
224     erlang_port = port;
225
226     std::cout << "Erlang endpoint set to " << ip << ":" << port << std
227     ::endl;
228     return true;
229 }
230 /**
231  * @brief Connects to the Erlang process.
232  * @return true if connection succeeded.
233  */
234 bool Server::connectToErlang()
235 {
236     std::lock_guard<std::mutex> lock(erlangMutex);
237
238     if (erlang_socket > 0)
239         return true; // Already connected
240
241     erlang_socket = socket(AF_INET, SOCK_STREAM, 0);
242     if (erlang_socket < 0) {
243         perror("Erlang socket creation failed");
244         return false;
245     }
246
247     sockaddr_in erlang_addr {};
248     erlang_addr.sin_family = AF_INET;
249     erlang_addr.sin_port = htons(erlang_port);
250
251     if (inet_pton(AF_INET, erlang_ip.c_str(), &erlang_addr.sin_addr)
252     <= 0) {
253         std::cerr << "Invalid Erlang IP: " << erlang_ip << std::endl;
254         close(erlang_socket);
255         erlang_socket = -1;
256         return false;
257     }
258
259     if (connect(erlang_socket, (struct sockaddr *)&erlang_addr, sizeof
260     (erlang_addr)) < 0) {
261         perror("Failed to connect to Erlang");
262         close(erlang_socket);
263         erlang_socket = -1;
264         return false;
265     }
266
267     std::cout << "Connected to Erlang on " << erlang_ip << ":" <<
268     erlang_port << std::endl;
269     return true;
270 }
271 /**
272  * @brief Sends a command to the Erlang process.
273  * @param command The command string to send.
274  */

```

```

273 void Server::sendToErlang(const std::string &command)
274 {
275     signal(SIGPIPE, SIG_IGN); // Ignore SIGPIPE to prevent crashes on
                                disconnect
276
277     if (!connectToErlang()) {
278         std::cerr << "Unable to send to Erlang: not connected.\n";
279         return;
280     }
281
282     std::lock_guard<std::mutex> lock(erlangMutex);
283
284     std::string msgWithNewline = command + "\n";
285     ssize_t sent = send(erlang_socket, msgWithNewline.c_str(),
msgWithNewline.size(), 0);
286
287     if (sent == -1) {
288         perror("send failed");
289
290         if (errno == EPIPE) {
291             std::cerr << "Broken pipe: Erlang side likely disconnected
." << std::endl;
292             close(erlang_socket);
293             erlang_socket = -1;
294         }
295
296         return;
297     }
298     std::cout << "Sent to Erlang: " << command << std::endl;
299 }
300
301 /**
302  * @brief Handles communication with a client.
303  * @param clientSocket The client socket file descriptor.
304  */
305 void Server::handleClient(int clientSocket)
306 {
307     std::string buffer;
308     char tempBuf[4096];
309
310     while (running) {
311         int valread = read(clientSocket, tempBuf, sizeof(tempBuf));
312         if (valread <= 0) {
313             if (valread == 0 || errno == ECONNRESET)
314                 break;
315             if (errno == EWOULDBLOCK || errno == EAGAIN) {
316                 std::this_thread::sleep_for(std::chrono::milliseconds
(10));
317                 continue;
318             }
319             perror("Read failed");
320             break;
321         }
322
323         buffer.append(tempBuf, valread);
324

```

```

325     size_t pos;
326     size_t offset = 0;
327     while ((pos = buffer.find('\n', offset)) != std::string::npos)
328     {
329         std::string_view message(buffer.data() + offset, pos -
offset);
330         parseErlangMessage(message.data(), message.size());
331         offset = pos + 1;
332     }
333     buffer.erase(0, offset);
334 }
335
336 close(clientSocket);
337 }
338
339 /**
340  * @brief Cleans up finished client threads.
341  */
342 void Server::cleanupThreads()
343 {
344     std::lock_guard<std::mutex> lock(clientsMutex);
345     auto it = clientThreads.begin();
346     while (it != clientThreads.end()) {
347         if (it->joinable()) {
348             it->join();
349             it = clientThreads.erase(it);
350         } else {
351             ++it;
352         }
353     }
354 }
355
356 /**
357  * @brief Stops the server and worker threads.
358  */
359 void Server::stop()
360 {
361     running = false;
362     {
363         std::lock_guard lock(queueMutex);
364         shutdownWorker = true;
365     }
366     queueCond.notify_all();
367     if (workerThread.joinable())
368         workerThread.join();
369 }
370
371 /**
372  * @brief Parses messages from Erlang and adds samples to queue.
373  * @param buffer The message buffer.
374  * @param len Length of the message.
375  */
376 void Server::parseErlangMessage(const char *buffer, int len)
377 {
378     if (buffer == nullptr || len <= 0 || len >= 1024) {
379         std::cerr << "error" << std::endl;
380     }
381 }

```

```

379         return;
380     }
381
382     std::string message(buffer, len);
383
384     // Parse message components
385     size_t nPos = message.find("n:");
386     size_t bPos = message.find(";b:");
387     size_t ePos = message.find(";e:");
388     size_t sPos = message.find(";s:");
389
390     if (nPos != 0 || bPos == std::string::npos || ePos == std::string::npos || sPos == std::string::npos) {
391         std::cerr << "Failed to parse message: " << message << std::endl;
392         return;
393     }
394
395     // Extract message fields
396     std::string name = message.substr(2, bPos - 2);
397     std::string bStr = message.substr(bPos + 3, ePos - (bPos + 3));
398     std::string eStr = message.substr(ePos + 3, sPos - (ePos + 3));
399     std::string statusStr = message.substr(sPos + 3);
400
401     // Convert to sample data
402     uint64_t startTime = std::stoull(bStr);
403     uint64_t endTime = std::stoull(eStr);
404     Sample sample;
405     Status status = Status::SUCCESS;
406
407     if (statusStr == TIMEOUT || statusStr == FAIL) {
408         status = (statusStr == TIMEOUT) ? Status::TIMEDOUT : Status::
FAILED;
409     } else if (statusStr != EXEC_OK) {
410         std::cerr << "Unknown status: " << statusStr << std::endl;
411         return;
412     }
413
414     double long elapsed = (endTime - startTime) / 1'000'000'000.0L;
415     sample = {startTime, endTime, elapsed, status};
416
417     // Add to processing queue
418     {
419         std::lock_guard lock(queueMutex);
420         sampleQueue.emplace(name, sample);
421     }
422     queueCond.notify_one();
423 }

```


Appendix D

C++ Header Files

D.1 Root

D.1.1 Application.h

```
1
2 #pragma once
3 #include "diagram/Observable.h"
4 #include "diagram/System.h"
5 #include "server/Server.h"
6 #include <functional>
7 #include <iostream>
8 #include <memory>
9 #include <mutex>
10 #include <vector>
11
12 struct SystemDiff {
13     std::vector<std::string> addedProbes;
14     std::vector<std::string> removedProbes;
15     std::vector<std::string> changedProbes;
16
17     std::vector<std::string> addedOutcomes;
18     std::vector<std::string> removedOutcomes;
19     std::vector<std::string> changedOutcomes;
20
21     std::vector<std::string> addedOperators;
22     std::vector<std::string> removedOperators;
23     std::vector<std::string> changedOperators;
24 };
25
26 class Application
27 {
28
29     std::shared_ptr<System> system = nullptr;
30     std::vector<std::function<void()>> observers; // List of functions
31     // to notify
32     Server *server = nullptr;
33     bool componentsDiffer(const std::shared_ptr<Observable> &a, const
34     std::shared_ptr<Observable> &b);
```

```
33
34     SystemDiff diffWith(System &newSystem);
35
36     Application();
37     void notifyObservers();
38
39 public:
40     static Application &getInstance();
41     void setServer(Server *);
42     void setSystem(System newSystem);
43     std::shared_ptr<System> getSystem();
44     void addObserver(std::function<void()> callback);
45     void sendDelayChange(std::string &, double);
46
47     bool startCppServer(const std::string &&, int);
48     void stopCppServer();
49
50     void setErlangEndpoint(const std::string &&, int);
51     void setStubRunning(bool running);
52 };
```

D.2 dashboard

D.2.1 ColorRegistry.h

```
1 #pragma once
2 #include <QColor>
3 #include <string>
4 #include <unordered_map>
5
6 class ColorRegistry
7 {
8 public:
9     static QColor getColorFor(const std::string &name);
10
11 private:
12     static QColor generateDistinctColor(int index);
13     static std::unordered_map<std::string, QColor> colorMap;
14 };
```

D.2.2 CustomLegendEntry.h

```
1
2
3 #pragma once
4
5 #include <QLabel>
6 #include <QWidget>
7 #include <qboxlayout.h>
8
9 class CustomLegendEntry : public QWidget
10 {
11     Q_OBJECT
```

```
12     QHBoxLayout *layout;
13     QLabel *colorBox;
14     QLabel *nameLabel;
15
16 public:
17     CustomLegendEntry(const QString &name, const QColor &color,
18         QWidget *parent = nullptr);
19 };
```

D.2.3 CustomLegendPanel.h

```
1 #pragma once
2
3 #include <QVBoxLayout>
4 #include <QWidget>
5 #include <map>
6 #include <qscrollarea.h>
7
8 /**
9  * @class CustomLegendPanel
10  * @brief A scrollable widget that displays a legend for a plot.
11  */
12 class CustomLegendPanel : public QWidget
13 {
14     Q_OBJECT
15
16 public:
17     explicit CustomLegendPanel(QWidget *parent = nullptr);
18
19     /**
20      * @brief Adds a new entry to the legend.
21      * @param name The display name for the entry.
22      * @param color The color for the entry.
23      */
24     void addEntry(const QString &name, const QColor &color);
25
26     /**
27      * @brief Removes an entry from the legend by name.
28      * @param name The name of the entry to remove.
29      */
30     void removeEntry(const QString &name);
31
32     /**
33      * @brief Clears all entries from the legend.
34      */
35     void clear();
36
37 private:
38     std::map<QString, QWidget *> legendEntries; ///< Map of legend
39     QVBoxLayout *mainLayout;                  ///< Main layout of
40     QVBoxLayout *legendLayout;                 ///< Layout containing
41     the legend entries
```

```
41     QScrollArea *scrollArea;                ///< Scroll area for
        the legend content
42     QWidget *scrollContent;                ///< Widget containing
        the scrollable content
43 };
```

D.2.4 DQPlotController.h

```
1
2 #ifndef DQPLOTCONTROLLER_H
3 #define DQPLOTCONTROLLER_H
4
5 #include <qlineseries.h>
6 #pragma once
7
8 // Project includes
9 #include "../diagram/System.h"
10 #include "DeltaQPlot.h"
11
12 // Qt includes
13 #include <QString>
14 #include <QtCharts/QLineSeries>
15
16 // C++ includes
17 #include <map>
18 #include <memory>
19 #include <string>
20 #include <vector>
21
22 // All series pertaining to an outcome
23 struct OutcomeSeries {
24     QLineSeries *outcomeS;
25     QLineSeries *lowerBoundS;
26     QLineSeries *upperBoundS;
27     QLineSeries *meanS;
28     QLineSeries *qtaS;
29 };
30
31 // All series pertaining to a probe
32 struct ExpressionSeries {
33     QLineSeries *obsS;
34     QLineSeries *obsLowerBoundS;
35     QLineSeries *obsUpperBoundS;
36     QLineSeries *obsMeanS;
37     QLineSeries *calcS;
38     QLineSeries *calcLowerBoundS;
39     QLineSeries *calcUpperBoundS;
40     QLineSeries *calcMeanS;
41     QLineSeries *qtaS;
42 };
43
44 class DeltaQPlot;
45
46 /**
47  * @class DQPlotController
```

```
48  * @brief Controls the data management and update logic of DeltaQPlot.
49  */
50  class DQPlotController
51  {
52  public:
53      /**
54       * @brief Constructor with associated plot and selected components
55       */
56      DQPlotController(DeltaQPlot *plot, const std::vector<std::string>
&selectedItems);
57
58      /**
59       * @brief Destructor.
60       */
61      ~DQPlotController();
62
63      /**
64       * @brief Checks if a component is already being plotted.
65       */
66      bool containsComponent(std::string name);
67
68      /**
69       * @brief Updates the plot according to a new list of selected
components.
70       */
71      void editPlot(const std::vector<std::string> &selectedItems);
72
73      /**
74       * @brief Adds a new component (probe or outcome) to the plot.
75       */
76      void addComponent(const std::string &name, bool isOutcome);
77
78      QLineSeries *createAndAddLineSeries(const std::string &legendName)
;
79
80      void addOutcomeSeries(const std::string &name);
81
82      void removeOutcomeSeries(const std::string &name);
83
84      void addExpressionSeries(const std::string &name, bool isProbe);
85
86      void removeExpressionSeries(const std::string &name, bool isProbe)
;
87      /**
88       * @brief Returns a list of names of all currently plotted
components.
89       */
90      std::vector<std::string> getComponents();
91      /**
92       * @brief Removes a plotted component by name (const reference).
93       */
94      void removeComponent(const std::string &name);
95
96      /**
```

```

97     * @brief Updates the data of all series based on provided time
    range and bin width.
98     */
99     void update(uint64_t timeLowerBound, uint64_t timeUpperBound);
100
101     void setTitle();
102
103     bool isEmptyAfterReset();
104
105 private:
106     double updateOutcome(OutcomeSeries &, const std::shared_ptr<
    Outcome> &, uint64_t, uint64_t);
107
108     double updateProbe(ExpressionSeries &, std::shared_ptr<Probe> &,
    uint64_t, uint64_t);
109
110     double updateOperator(ExpressionSeries &, std::shared_ptr<Operator
    > &, uint64_t, uint64_t);
111
112     void updateExpression(ExpressionSeries &, DeltaQRepr &&,
    DeltaQRepr &&, QTA &&, double maxDelay);
113
114     DeltaQPlot *plot;
115
116     std::mutex updateMutex;
117     std::mutex resetMutex;
118
119     std::map<std::string, std::pair<OutcomeSeries, std::shared_ptr<
    Outcome>>> outcomes;
120     std::map<std::string, std::pair<ExpressionSeries, std::shared_ptr<
    Probe>>> probes;
121     std::map<std::string, std::pair<ExpressionSeries, std::shared_ptr<
    Operator>>> operators;
122 };
123
124 #endif // DQPLOTCONTROLLER_H

```

D.2.5 DQPlotList.h

```

1 #pragma once
2
3 #ifndef DQ_PLOT_LIST_H
4 #define DQ_PLOT_LIST_H
5
6 #include "DQPlotController.h"
7 #include <QCheckBox>
8 #include <QListWidget>
9 #include <QPushButton>
10 #include <QVBoxLayout>
11 #include <QWidget>
12
13 class DQPlotController;
14
15 /**
16  * @class DQPlotList

```

```
17  * @brief A widget that displays and manages lists of available and
18  *       selected plot components for a selected plot.
19  */
20  class DQPlotList : public QWidget
21  {
22      Q_OBJECT
23
24  public:
25      explicit DQPlotList(DQPlotController *controller, QWidget *parent
26                          = nullptr);
27
28      /**
29       * @brief Resets the widget's state.
30       */
31      void reset();
32
33      /**
34       * @brief Checks if the widget is empty after reset.
35       * @return true if no components are selected, false otherwise.
36       */
37      bool isEmptyAfterReset();
38
39      /**
40       * @brief Updates both the available and selected lists.
41       */
42      void updateLists();
43
44      /**
45       * @brief Default destructor.
46       */
47      ~DQPlotList() = default;
48
49  private Q_SLOTS:
50      /**
51       * @brief Handles confirmation of selected components to add.
52       */
53      void onConfirmSelection();
54
55      /**
56       * @brief Handles removal of selected components.
57       */
58      void onRemoveSelection();
59
60  private:
61      /**
62       * @brief Adds an item to either the available or selected list.
63       * @param name The name of the component.
64       * @param isSelected Whether the item should be in the selected
65       * list.
66       * @param category The category for the item (used for grouping).
67       */
68      void addItemToList(const std::string &name, bool isSelected, const
69                        QString &category);
```

```

68     DQPlotController *controller; ///< The controller managing plot
        components.
69
70     QListWidget *selectedList;      ///< List widget showing currently
        selected components.
71     QListWidget *availableList;    ///< List widget showing available
        components.
72
73     QPushButton *addButton;        ///< Button to add selected
        components.
74     QPushButton *removeButton;    ///< Button to remove selected
        components.
75 };
76
77 #endif // DQ_PLOT_LIST_H

```

D.2.6 DelaySettingsWidget.h

```

1  #pragma once
2
3  #include <QComboBox>
4  #include <QHBoxLayout>
5  #include <QLabel>
6  #include <QPushButton>
7  #include <QSlider>
8  #include <QSpinBox>
9  #include <QVBoxLayout>
10 #include <QWidget>
11 #include <cmath>
12
13 /**
14  * @brief A widget for configuring delay parameters for a selected
        observable.
15  */
16 class DelaySettingsWidget : public QWidget
17 {
18     Q_OBJECT
19
20 public:
21     explicit DelaySettingsWidget(QWidget *parent = nullptr);
22
23     /**
24      * @brief Populates the observable combo box using the system's
        observable list.
25      */
26     void populateComboBox();
27
28     /**
29      * @brief Computes the maximum delay in milliseconds based on the
        slider and spinbox values.
30      * @return Maximum delay in milliseconds.
31      */
32     double getMaxDelayMs() const;
33
34 Q_SIGNALS:

```



```

35     /**
36      * @brief Signal emitted when delay parameters have been changed
    and saved.
37      */
38     void delayParametersChanged();
39
40 private Q_SLOTS:
41     /**
42      * @brief Updates the label showing the current maximum delay
    based on UI values.
43      */
44     void updateMaxDelay();
45
46     /**
47      * @brief Handles saving the currently selected delay parameters
    to the system.
48      */
49     void onSaveDelayClicked();
50
51     /**
52      * @brief Loads the saved settings for the currently selected
    observable.
53      */
54     void loadObservableSettings();
55
56 private:
57     QVBoxLayout *mainLayout;           ///< Main layout container.
58     QLabel *settingsLabel;             ///< Label describing the
    purpose of the widget.
59     QLabel *maxDelayLabel;             ///< Label showing the computed
    max delay.
60
61     QHBoxLayout *settingsLayout;        ///< Layout for parameter
    controls.
62     QComboBox *observableComboBox;     ///< Combo box for selecting
    observables.
63     QSlider *delaySlider;               ///< Slider to set delay
    exponent.
64     QSpinBox *binSpinBox;               ///< Spin box to set number of
    bins.
65
66     QPushButton *saveDelayButton;      ///< Button to save the delay
    configuration.
67 };

```

D.2.7 DeltaQPlot.h

```

1
2 #ifndef DELTAQPLOT_H
3 #define DELTAQPLOT_H
4
5 #include "CustomLegendPanel.h"
6 #include <qboxlayout.h>
7 #pragma once
8

```

```
9 // Qt includes
10 #include <QChartView>
11 #include <QLineSeries>
12 #include <QToolButton>
13 #include <QValueAxis>
14 // C++ includes
15 #include <string>
16 #include <vector>
17
18 class DQPlotController;
19 class DQPlotList;
20
21 /**
22  * @class DeltaQPlot
23  * @brief A class representing a DeltaQ chart view that allows
24  * visualization of probes over time.
25  */
26 class DeltaQPlot : public QWidget
27 {
28     Q_OBJECT
29 public:
30     /**
31      * @brief Constructs a DeltaQPlot with selected components.
32      * @param selectedItems List of selected component names.
33      * @param parent Parent widget.
34      */
35     explicit DeltaQPlot(const std::vector<std::string> &selectedItems,
36         QWidget *parent = nullptr);
37
38     /**
39      * @brief Destructor.
40      */
41     ~DeltaQPlot();
42
43     /**
44      * @brief Adds a QLineSeries to the chart.
45      * @param series Pointer to the line series.
46      * @param name Name of the series.
47      */
48     void addSeries(QLineSeries *series, const std::string &name);
49
50     /**
51      * @brief Updates the plot data using provided time range and bin
52      * width.
53      */
54     void update(uint64_t timeLowerBound, uint64_t timeUpperBound);
55
56     /**
57      * @brief Removes a series from the chart.
58      * @param series Series to remove.
59      */
60     void removeSeries(QAbstractSeries *series);
61
62     /**
63      * @brief Updates plot with new set of selected components.
```

```
62     */
63     void editPlot(const std::vector<std::string> &selectedItems);
64
65     /**
66      * @brief Gets list of currently plotted component names.
67      */
68     std::vector<std::string> getComponents();
69
70     bool isEmptyAfterReset();
71
72     /**
73      * @brief Updates an existing series with new data points.
74      */
75     void updateSeries(QLineSeries *series, const QVector<QPointF> &
76 data);
77
78     void updateXRange(double xRange);
79     /**
80      * @brief Returns the associated plot list.
81      */
82     DQPlotList *getPlotList();
83
84     void setTitle(QString &&);
85 protected:
86     /**
87      * @brief Handles mouse press events on the chart.
88      */
89     void mousePressEvent(QMouseEvent *event) override;
90
91     Q_SIGNALS:
92     /**
93      * @brief Emitted whenout
94      * QHBox this plot is selected by the user.
95      */
96     void plotSelected(DeltaQPlot *plot);
97
98     private:
99     QHBoxLayout *layout;
100     QToolButton *toggleButton;
101     QChartView *chartView;
102     QChart *chart;
103
104     QValueAxis *axisX;
105     QValueAxis *axisY;
106
107     QLineSeries *operationSeries;
108     DQPlotController *controller;
109     DQPlotList *plotList;
110
111     CustomLegendPanel *legendPanel;
112 };
113
114 #endif // DELTAQPLOT_H
```

D.2.8 MainWindow.h

```

1  #pragma once
2  #include "../diagram/System.h"
3  #include "DeltaQPlot.h"
4  #include "NewPlotList.h"
5  #include "ObservableSettings.h"
6  #include "Sidebar.h"
7  #include "StubControlWidget.h"
8  #include "TriggersTab.h"
9
10 #include <QHBoxLayout>
11 #include <QListWidget>
12 #include <QMainWindow>
13 #include <QPushButton>
14 #include <QTimer>
15 #include <QVBoxLayout>
16 #include <qboxlayout.h>
17 #include <qwidget.h>
18
19 /**
20  * @class MainWindow
21  * @brief The main application window containing plots and control
22  * panels.
23  */
24 class MainWindow : public QMainWindow
25 {
26     Q_OBJECT
27
28     QHBoxLayout *mainLayout;          ///< Main horizontal layout
29
30     QScrollArea *scrollArea;          ///< Scroll area for plots
31     QGridLayout *plotLayout;          ///< Grid layout for plot
32     arrangement
33     QWidget *plotContainer;           ///< Container widget for plots
34
35     QWidget *centralWidget;           ///< Central widget for main
36     layout
37
38     QThread *timerThread;             ///< Thread for update timer
39     QTimer *updateTimer;             ///< Timer for periodic updates
40
41     QWidget *sideContainer;           ///< Container for side panels
42     QVBoxLayout *sideLayout;          ///< Layout for side panels
43     QTabWidget *sideTabWidget;        ///< Tab widget for side panels
44     TriggersTab *triggersTab;         ///< Triggers configuration
45     panel
46     Sidebar *sidebar;                 ///< Main sidebar control panel
47     ObservableSettings *observableSettings; ///< Observable settings
48     panel
49
50     StubControlWidget *stubWidget;    ///< Stub control widget (
51     placeholder)
52     QPushButton *addPlotButton;      ///< Button to add new plots

```

```

48     QMap<DeltaQPlot *, QWidget *> plotContainers; ///< Map of plots to
        their containers
49     uint64_t timeLowerBound;                ///< Lower time bound for data
        updates
50
51     std::mutex plotDelMutex;                ///< Mutex for plot deletion
        safety
52     std::mutex updateMutex;                ///< Mutex for update operations
53
54     int pollingRate {200};                ///< Current polling rate in
        milliseconds
55
56 public:
57     MainWindow(QWidget *parent = nullptr);
58
59     ~MainWindow();
60
61     /**
62      * @brief Resets the window state, cleaning up empty plots.
63      */
64     void reset();
65
66 private Q_SLOTS:
67     /**
68      * @brief Updates all plots with new data.
69      */
70     void updatePlots();
71
72     /**
73      * @brief Handles adding new plots from sidebar selection.
74      */
75     void onAddPlotClicked();
76
77     /**
78      * @brief Removes a specific plot.
79      * @param plot The plot to remove.
80      */
81     void onRemovePlot(DeltaQPlot *plot);
82
83     /**
84      * @brief Handles plot selection changes.
85      * @param plot The newly selected plot.
86      */
87     void onPlotSelected(DeltaQPlot *plot);
88
89 protected:
90     /**
91      * @brief Handles context menu events for plot management.
92      * @param event The context menu event.
93      */
94     void contextMenuEvent(QContextMenuEvent *event) override;
95
96     /**
97      * @brief Handles window resize events to adjust plot sizes.
98      * @param event The resize event.
99      */

```

```
100     void resizeEvent(QResizeEvent *event) override;
101 };
```

D.2.9 NewPlotList.h

```
1  #pragma once
2
3  #ifndef NEW_PLOT_LIST_H
4  #define NEW_PLOT_LIST_H
5
6  #include "../diagram/System.h"
7  #include <QCheckBox>
8  #include <QListWidget>
9  #include <qlistwidget.h>
10 #include <qwidget.h>
11
12 /**
13  * @brief A list widget for selecting observables to create a new plot
14  *
15  */
16 class NewPlotList : public QListWidget
17 {
18     Q_OBJECT
19
20 public:
21     explicit NewPlotList(QWidget *parent = nullptr);
22
23     /**
24      * @brief Gets the names of all currently selected observables.
25      * @return A vector of strings representing selected observables.
26      */
27     std::vector<std::string> getSelectedItems();
28
29     /**
30      * @brief Deselects all currently selected items in the list.
31      */
32     void deselectAll();
33
34     /**
35      * @brief Clears the list and repopulates it with updated
36      * observables from the system.
37      */
38     void reset();
39 private:
40     /**
41      * @brief Adds observable items to the list from the current
42      * system.
43      */
44     void addItem();
45 };
46 #endif // NEW_PLOT_LIST_H
```

D.2.10 ObservableSettings.h

```
1 #ifndef OBS_SETTINGS_H
2 #define OBS_SETTINGS_H
3
4 #include <qlabel.h>
5 #pragma once
6 #include <QWidget>
7
8 #include "DelaySettingsWidget.h"
9 #include "QTAInputWidget.h"
10
11 class ObservableSettings : public QWidget
12 {
13     Q_OBJECT
14
15     QVBoxLayout *layout;
16
17     QLabel *delayLabel;
18     DelaySettingsWidget *delaySettingsWidget;
19
20     QLabel *qtaLabel;
21     QTAInputWidget *qtaInputWidget;
22
23 public:
24     explicit ObservableSettings(QWidget *parent = nullptr);
25 };
26
27 #endif
```

D.2.11 PollingRateWidget.h

```
1 #pragma once
2
3 #include <QLabel>
4 #include <QPushButton>
5 #include <QSlider>
6 #include <QVBoxLayout>
7 #include <QWidget>
8
9 /**
10  * @class PollingRateWidget
11  * @brief A widget for selecting and saving polling rate intervals.
12  *
13  * Provides a slider interface to select from predefined polling rates
14  * (in milliseconds) and emits the selected rate when saved.
15  */
16 class PollingRateWidget : public QWidget
17 {
18     Q_OBJECT
19
20 public:
21     explicit PollingRateWidget(QWidget *parent = nullptr);
22
23 private Q_SLOTS:
```

```

24     /**
25      * @brief Handles slider value changes to update the displayed
rate.
26      * @param value The current slider index (0-based).
27      */
28     void onSliderValueChanged(int value);
29
30     /**
31      * @brief Handles save button clicks to emit the selected rate.
32      */
33     void onSaveClicked();
34
35 Q_SIGNALS:
36     /**
37      * @brief Emitted when a new polling rate is saved.
38      * @param milliseconds The selected polling rate in milliseconds.
39      */
40     void onPollingRateChanged(int milliseconds);
41
42 private:
43     QSlider *slider;           ///< Slider for selecting polling rate
44     QLabel *valueLabel;       ///< Displays the current polling rate
45     QPushButton *saveButton;  ///< Button to save the selected rate
46     QVector<int> pollingRates; ///< Available polling rate options (
ms)
47 };

```

D.2.12 QTAInputWidget.h

```

1  #ifndef QTAINPUTWIDGET_H
2  #define QTAINPUTWIDGET_H
3
4  #include <QWidget>
5  #include <QLineEdit>
6  #include <QComboBox>
7  #include <QFormLayout>
8  #include <QLabel>
9  #include <QPushButton>
10
11 /**
12  * @class QTAInputWidget
13  * @brief A Qt widget for configuring QTAs for observables.
14  */
15 class QTAInputWidget : public QWidget
16 {
17     Q_OBJECT
18
19 public:
20     explicit QTAInputWidget(QWidget *parent = nullptr);
21
22     /**
23      * @brief Gets the 25th percentile value (in seconds).
24      * @return The value entered in the 25th percentile field.
25      */
26     double getPerc25() const;

```



```

27
28     /**
29     * @brief Gets the 50th percentile (median) value (in seconds).
30     * @return The value entered in the 50th percentile field.
31     */
32     double getPerc50() const;
33
34     /**
35     * @brief Gets the 75th percentile value (in seconds).
36     * @return The value entered in the 75th percentile field.
37     */
38     double getPerc75() const;
39
40     /**
41     * @brief Gets the maximum allowed CDF value (0 to 1).
42     * @return The value entered in the CDF max field.
43     */
44     double getCdfMax() const;
45
46     /**
47     * @brief Gets the currently selected observable name.
48     * @return The name of the selected observable.
49     */
50     QString getSelectedObservable() const;
51
52 public Q_SLOTS:
53     /**
54     * @brief Populates the observable dropdown with available
55     * observables.
56     * @note Called automatically when the system updates.
57     */
58     void populateComboBox();
59
60     /**
61     * @brief Loads QTA settings for the selected observable into the
62     * UI fields.
63     * @note Triggered when the dropdown selection changes.
64     */
65     void loadObservableSettings();
66
67     /**
68     * @brief Saves the current QTA settings to the system.
69     * @note Called when the "Save" button is clicked.
70     * @throws std::exception if validation fails (e.g., invalid CDF
71     * value).
72     */
73     void onSaveButtonClicked();
74
75 private:
76     QComboBox *observableComboBox;    ///< Dropdown to select an
77     observable (probe/outcome).
78     QLineEdit *perc25Edit;            ///< Input field for the 25th
79     percentile (seconds).
80     QLineEdit *perc50Edit;            ///< Input field for the 50th
81     percentile (seconds).

```

```

76     QLineEdit *perc75Edit;           ///< Input field for the 75th
percentile (seconds).
77     QLineEdit *cdfMaxEdit;           ///< Input field for the max CDF
value (0-1).
78     QPushButton *saveButton;         ///< Button to save QTA settings.
79     QLabel *qtaLabel;                ///< Label describing the widget's
purpose.
80 };
81
82 #endif // QTAINPUTWIDGET_H

```

D.2.13 Sidebar.h

```

1  #ifndef SIDEBAR_H
2  #define SIDEBAR_H
3
4  #include "DQPlotList.h"
5  #include "NewPlotList.h"
6  #include "PollingRateWidget.h"
7  #include "SystemCreationWidget.h"
8  #include <QComboBox>
9  #include <QLabel>
10 #include <QPushButton>
11 #include <QSpinBox>
12 #include <QSplitter>
13 #include <QTextEdit>
14 #include <QVBoxLayout>
15 #include <QWidget>
16 #include <qboxlayout.h>
17
18 /**
19  * @class Sidebar
20  * @brief Main sidebar widget containing plot management controls and
system configuration.
21  *
22  * The Sidebar provides UI components for:
23  * - Creating new plots
24  * - Managing existing plots
25  * - System configuration
26  * - Polling rate adjustment
27  */
28 class Sidebar : public QWidget
29 {
30     Q_OBJECT
31
32     QVBoxLayout *newPlotListLayout;           ///< Layout for new plot
selection components
33     QWidget *newPlotListWidget;               ///< Container widget for new
plot controls
34     QLabel *newPlotLabel;                     ///< Label for new plot
section
35     NewPlotList *newPlotList;                 ///< List widget for
selecting probes for new plots
36     QPushButton *addNewPlotButton;           ///< Button to create new
plot

```

```

37
38     QWidget *currentPlotWidget;           ///< Container widget for
current plot controls
39     QVBoxLayout *currentPlotLayout;       ///< Layout for current plot
components
40     QLabel *currentPlotLabel;             ///< Label for current plot
section
41     DQPlotList *currentPlotList = nullptr; ///< List widget for
managing current plot's probes
42
43     QSplitter *mainSplitter;              ///< Main splitter organizing
sections vertically
44     QVBoxLayout *layout;                  ///< Main layout of the
sidebar
45
46     SystemCreationWidget *systemCreationWidget; ///< Widget for system
creation/configuration
47     PollingRateWidget *pollingRateWidget; ///< Widget for adjusting
polling rate
48
49 Q_SIGNALS:
50     /**
51      * @brief Emitted when the "Add plot" button is clicked.
52      */
53     void addPlotClicked();
54
55     /**
56      * @brief Emitted when polling rate is changed.
57      * @param milliseconds The new polling rate in milliseconds.
58      */
59     void onPollingRateChanged(int milliseconds);
60
61 private Q_SLOTS:
62     /**
63      * @brief Handles "Add plot" button click event.
64      */
65     void onAddPlotClicked();
66
67     /**
68      * @brief Handles polling rate change events.
69      * @param ms The new polling rate in milliseconds.
70      */
71     void handlePollingRateChanged(int ms);
72
73 public:
74     /**
75      * @brief Constructs a Sidebar widget.
76      * @param parent The parent widget (optional).
77      */
78     explicit Sidebar(QWidget *parent = nullptr);
79
80     /**
81      * @brief Sets the current plot list widget.
82      * @param currentPlotList The DQPlotList widget to display.
83      */
84     void setCurrentPlotList(DQPlotList *currentPlotList);

```

```
85
86     /**
87      * @brief Hides the current plot management section.
88      */
89     void hideCurrentPlot();
90
91     /**
92      * @brief Gets the new plot list widget.
93      * @return Pointer to the NewPlotList widget.
94      */
95     NewPlotList *getPlotList() const
96     {
97         return newPlotList;
98     }
99
100    /**
101     * @brief Clears new plot selection after plot creation.
102     */
103    void clearOnAdd();
104};
105
106#endif
```

D.2.14 SnapshotViewerWindow.h

```
1 #pragma once
2
3 #include <QChartView>
4 #include <QComboBox>
5 #include <QLabel>
6 #include <QSlider>
7 #include <QWidget>
8
9 #include "../maths/Snapshot.h"
10 #include <map>
11
12 /**
13  * @brief A QWidget-based window for visualizing snapshots from fired
14  * triggers.
15  */
16 class SnapshotViewerWindow : public QWidget
17 {
18     Q_OBJECT
19 public:
20     explicit SnapshotViewerWindow(std::vector<Snapshot> &snapshotList,
21                                   QWidget *parent = nullptr);
22
23     /**
24      * @brief Sets the snapshots to display in the viewer.
25      * @param snapshotList A vector of Snapshots.
26      */
27     void setSnapshots(std::vector<Snapshot> &snapshotList);
28 private:
29     Q_SLOTS:
```

```

29     /**
30      * @brief Slot triggered when the observable selection changes.
31      * @param name Name of the newly selected observable.
32      */
33     void onObservableChanged(const QString &name);
34
35     /**
36      * @brief Slot triggered when the time slider is moved.
37      * @param value Index of the snapshot in the selected observable.
38      */
39     void onTimeSliderChanged(int value);
40
41 private:
42     /**
43      * @brief Updates the chart view based on the current observable
44      * and time index.
45      */
46     void updatePlot();
47
48     QChartView *chartView;           ///< Chart view for
49     plotting the snapshot data.
50     QComboBox *observableSelector;    ///< Dropdown for
51     selecting an observable.
52     QSlider *timeSlider;              ///< Slider for
53     selecting time index.
54     QLabel *timeLabel;               ///< Label showing the
55     current time.
56
57     std::map<std::string, Snapshot> snapshots; ///< Map of observable
58     name to snapshot.
59     std::string currentObservable;      ///< Currently selected
60     observable.
61 };

```

D.2.15 StubControlWidget.h

```

1  #pragma once
2  #include "src/Application.h"
3  #include <QApplication>
4  #include <QGridLayout>
5  #include <QGroupBox>
6  #include <QHBoxLayout>
7  #include <QLabel>
8  #include <QLineEdit>
9  #include <QPushButton>
10 #include <QVBoxLayout>
11 #include <QWidget>
12
13 class StubControlWidget : public QWidget
14 {
15     Q_OBJECT
16 public:
17     StubControlWidget(QWidget *parent = nullptr);
18
19 private:

```

```

20 // Erlang controls
21 QPushButton *startErlangButton;
22 QPushButton *stopErlangButton;
23
24 // Server controls
25 QPushButton *startServerButton;
26 QPushButton *stopServerButton;
27 QLineEdit *serverIpEdit;
28 QLineEdit *serverPortEdit;
29
30 // Erlang receiver settings
31 QLineEdit *erlangReceiverIpEdit;
32 QLineEdit *erlangReceiverPortEdit;
33 QPushButton *setErlangEndpointButton;
34
35 QVBoxLayout *mainLayout;
36
37 private Q_SLOTS:
38 // Erlang slots
39 void onStartErlangClicked();
40 void onStopErlangClicked();
41
42 // Server slots
43 void onStartServerClicked();
44 void onStopServerClicked();
45
46 // Erlang endpoint slot
47 void onSetErlangEndpointClicked();
48 };

```

D.2.16 SystemCreationWidget.h

```

1 #ifndef SYSTEMCREATIONWIDGET_H
2 #define SYSTEMCREATIONWIDGET_H
3
4 #include <QHBoxLayout>
5 #include <QLabel>
6 #include <QPushButton>
7 #include <QTextEdit>
8 #include <QVBoxLayout>
9 #include <QWidget>
10
11 /**
12  * @class SystemCreationWidget
13  * @brief A QWidget that allows users to create, edit, load, and save
14  *        system definitions.
15  */
16 class SystemCreationWidget : public QWidget
17 {
18     Q_OBJECT
19 public:
20     explicit SystemCreationWidget(QWidget *parent = nullptr);
21
22     /**

```

```
23     * @brief Retrieves the current system text from the editor.
24     * @return The system text as a std::string.
25     */
26     std::string getSystemText() const;
27
28     /**
29     * @brief Sets the system text in the editor.
30     * @param text The new system definition text.
31     */
32     void setSystemText(const std::string &text);
33
34     Q_SIGNALS:
35     /**
36     * @brief Emitted when the system is successfully updated.
37     */
38     void systemUpdated();
39
40     /**
41     * @brief Emitted when the system is successfully saved.
42     */
43     void systemSaved();
44
45     /**
46     * @brief Emitted when a system is successfully loaded.
47     */
48     void systemLoaded();
49
50     private Q_SLOTS:
51     /**
52     * @brief Parses the text and updates the system instance.
53     */
54     void onUpdateSystem();
55
56     /**
57     * @brief Saves the current system text to a file.
58     */
59     void saveSystemTo();
60
61     /**
62     * @brief Loads a system from a file and updates the editor.
63     */
64     void loadSystem();
65
66     private:
67     QTextEdit *systemTextEdit;          ///< Editor widget for system
68     text.
69     QPushButton *updateSystemButton;    ///< Button to update system.
70     QPushButton *saveSystemButton;      ///< Button to save system.
71     QPushButton *loadSystemButton;      ///< Button to load system.
72     QLabel *systemLabel;                ///< Label describing the
73     editor.
74
75     QVBoxLayout *mainLayout;            ///< Layout for the main
76     components.
77     QHBoxLayout *buttonLayout;           ///< Layout for the buttons.
78 };
```

```
76
77 #endif // SYSTEMCREATIONWIDGET_H
```

D.2.17 TriggersTab.h

```
1 #pragma once
2
3 #include <QCheckBox>
4 #include <QComboBox>
5 #include <QFormLayout>
6 #include <QListWidget>
7 #include <QMap>
8 #include <QSpinBox>
9 #include <QVBoxLayout>
10 #include <QWidget>
11
12 #include "../Application.h"
13 #include "src/maths/TriggerManager.h"
14
15 /**
16  * @class TriggersTab
17  * @brief Widget for managing and monitoring trigger conditions on
18  *        observables.
19  *
20  * Provides UI for:
21  * - Setting up trigger conditions (sample limits, QTA violations)
22  * - Displaying triggered events
23  * - Viewing snapshots of triggered states
24  */
25 class TriggersTab : public QWidget
26 {
27     Q_OBJECT
28 public:
29     explicit TriggersTab(QWidget *parent = nullptr);
30
31     ~TriggersTab();
32
33     /**
34      * @brief Adds a triggered message to the display list.
35      * @param msg The message to display.
36      */
37     void addTriggeredMessage(const QString &msg);
38
39 private Q_SLOTS:
40     /**
41      * @brief Handles observable selection changes.
42      * @param name The newly selected observable name.
43      */
44     void onObservableChanged(const QString &name);
45
46     /**
47      * @brief Handles trigger condition changes.
48      */
49     void onTriggerChanged();
```



```

50
51     /**
52      * @brief Handles triggered item clicks to show snapshots.
53      * @param item The clicked list item.
54      */
55     void onTriggeredItemClicked(QListWidgetItem *item);
56
57 private:
58     /**
59      * @brief Captures snapshots when triggers are activated.
60      * @param time The timestamp of the trigger event.
61      * @param name The name of the observable that triggered.
62      */
63     void captureSnapshots(std::uint64_t time, const std::string &name)
64     ;
65
66     /**
67      * @brief Populates the observable dropdown list.
68      */
69     void populateObservables();
70
71     /**
72      * @brief Updates checkbox states based on current triggers.
73      */
74     void updateCheckboxStates();
75
76     QVBoxLayout *mainLayout;           ///< Main vertical layout
77     QFormLayout *formLayout;           ///< Form layout for controls
78     QComboBox *observableComboBox;     ///< Dropdown for observable
79     selection
80
81     QWidget *sampleLimitWidget;        ///< Container for sample
82     limit controls
83     QHBoxLayout *sampleLimitLayout;    ///< Layout for sample limit
84     controls
85     QCheckBox *sampleLimitCheckBox;     ///< Checkbox to enable
86     sample limit trigger
87     QSpinBox *sampleLimitSpinBox;      ///< Spinbox for sample limit
88     threshold
89
90     QCheckBox *qtaBoundsCheckBox;       ///< Checkbox for QTA bounds
91     violation trigger
92
93     QListWidget *triggeredList;         ///< List widget for
94     triggered events
95
96     /**
97      * @brief Gets the currently selected observable.
98      * @return Shared pointer to the current observable.
99      * @throws std::runtime_error if system or observable doesn't
100     exist.
101     */
102     std::shared_ptr<Observable> getCurrentObservable();
103
104     static constexpr int sampleLimitThreshold = 500;    ///< Default
105     sample limit threshold

```

```
96     static constexpr double failureRateThreshold = 0.95;    ///< Failure
97     rate threshold constant
};
```

D.3 diagram

D.3.1 Observable.h

```
1  #pragma once
2
3  #include "../maths/Snapshot.h"
4  #include "Sample.h"
5  #include "src/math/TriggerManager.h"
6  #include <deque>
7  #include <math.h>
8  #include <mutex>
9
10 #define DELTA_T_BASE 0.001
11 #define MAX_DQ 30
12 class Observable
13 {
14 protected:
15     std::string name;
16     std::deque<Sample> samples;
17     mutable bool sorted;
18
19     std::deque<DeltaQ> confidenceIntervalHistory;
20     double maxDelay {0.05};
21     int deltaTExp {0}; // Exponent for dynamic binning
22     int nBins {50}; // Number of bins
23
24     TriggerManager triggerManager;
25
26     ConfidenceInterval observedInterval;
27     QTA qta;
28
29     Snapshot observableSnapshot;
30
31     std::mutex observedMutex;
32     std::mutex samplesMutex;
33     std::mutex paramMutex;
34
35     bool recording = false;
36
37 private:
38     void updateSnapshot(uint64_t timeLowerBound, DeltaQ &deltaQ);
39
40 public:
41     /**
42      * @brief Constructor an observable with its name
43      */
44     Observable(const std::string &name);
45
46     virtual ~Observable() { };
```

```
47     /**
48      * Add a sample (outcome instance) to an observable
49      */
50     void addSample(const Sample &sample);
51     /**
52      * @brief Get all sample with endTime timeLowerBound -
timeUpperBound
53      * @return The sample in range timeLowerBound - timeUpperBound
54      */
55     std::vector<Sample> getSamplesInRange(std::uint64_t timeLowerBound
, std::uint64_t timeUpperBound);
56     /**
57      * @brief Get observed DeltaQ in range timeLowerBound -
timeUpperBound from snapshot, if it has not been calculated,
calculate it
58      * @return DeltaQ
59      */
60     DeltaQ getObservedDeltaQ(uint64_t, uint64_t);
61     /**
62      * @brief Calculate the observed DeltaQ in range timeLowerBound -
timeUpperBound, add it to snapshot and ConfidenceInterval
63      * @return The calculated DeltaQ
64      */
65     DeltaQ calculateObservedDeltaQ(uint64_t, uint64_t);
66
67     /**
68      * @brief Get DeltaQ representation for graphical plotting
69      */
70     DeltaQRepr getObservedDeltaQRepr(uint64_t, uint64_t);
71
72     /**
73      * @brief Set new parameters for a DeltaQ
74      * @return new dMax
75      */
76     double setNewParameters(int newExp, int newNBins);
77
78     double getBinWidth() const
79     {
80         return DELTA_T_BASE * std::pow(2, deltaTExp);
81     }
82
83     int getNBins() const
84     {
85         return nBins;
86     }
87
88     double getMaxDelay() const
89     {
90         return maxDelay;
91     }
92
93     QTA getQTA() const
94     {
95         return qta;
96     }
97
```

```

98     int getDeltaTExp() const
99     {
100         return deltaTExp;
101     }
102
103     const TriggerManager &getTriggerManager() const
104     {
105         return triggerManager;
106     }
107
108     /**
109     * @brief Set recoding snapshot
110     * @param bool is recording
111     */
112     void setRecording(bool);
113
114     /**
115     * @brief Set QTA for an observable
116     * @param qta new QTA
117     */
118     void setQTA(const QTA &);
119     /**
120     * @brief add a trigger to an observable
121     * @param type the observable type
122     * @param condition the condition to evaluate
123     * @param action action to perform on trigger fired
124     * @param enabled
125     * @param sampleLimit sampleLimit sample limit for sample limit
126     trigger
127     */
128     void addTrigger(TriggerType type, TriggerDefs::Condition condition
129     , TriggerDefs::Action action, bool enabled, std::optional<int>
130     sampleLimit);
131     /**
132     * @brief Remove observable trigger
133     */
134     void removeTrigger(TriggerType type);
135
136     /**
137     * @brief Get snapshot of observable
138     */
139     Snapshot getSnapshot();
140     /**
141     * @brief Get observable name
142     * @return its name
143     */
144     [[nodiscard]] std::string getName() const &;
145 };

```

D.3.2 Operator.h

```

1 #pragma once
2
3 #include "../maths/DeltaQ.h"
4 #include "Observable.h"

```

```

5 #include "OperatorType.h"
6
7 /**
8  * @class Operator This class represents an operator according to the
9  * DeltaQSD paradigm
10 */
11 class Operator : public Observable
12 {
13     OperatorType type;
14
15     std::vector<double> probabilities; ///< The probabilities of the
16     components inside the operator, only available for probabilistic
17     operator
18
19     std::vector<std::vector<std::shared_ptr<Observable>>> causalLinks;
20     ///< The causal links for each children
21
22     ConfidenceInterval calculatedInterval;
23
24     std::mutex calcMutex;
25
26     std::deque<DeltaQ> calculatedDeltaQHistory; ///< History for
27     confidence intervals
28
29 public:
30     Operator(const std::string &name, OperatorType);
31
32     ~Operator();
33
34     /**
35      * @brief calculate a Calculated deltaQ with bounds timeLowerBound
36      * , timeUpperBound
37      * @param timeLowerBound
38      * @param timeUpperBound
39      * @return calculated DeltaQ
40      */
41     DeltaQ calculateCalculatedDeltaQ(uint64_t, uint64_t);
42
43     /**
44      * @brief Get the representation of a calculated DeltaQ for
45      plotting
46      * @return the representation of a calculated DeltaQ
47      */
48     DeltaQRepr getCalculatedDeltaQRepr(uint64_t, uint64_t);
49
50     void setProbabilities(const std::vector<double> &);
51
52     /**
53      * @brief Get the probabilities of a probabilistic operator
54      * @return The probabilities
55      */
56     std::vector<double> getProbabilities()
57     {
58         return probabilities;
59     }
60
61     /**

```

```
54     * @brief Get the links for a children
55     * @return The links for a children
56     */
57     std::vector<std::shared_ptr<Observable>> getChildren();
58
59     void setCausalLinks(std::vector<std::vector<std::shared_ptr<
60     Observable>>> links)
61     {
62         causalLinks = links;
63     }
64
65     std::vector<std::vector<std::shared_ptr<Observable>>>
66     getCausalLinks()
67     {
68         return causalLinks;
69     }
70
71     OperatorType getType()
72     {
73         return type;
74     }
75 };
```

D.3.3 OperatorType.h

```
1 #pragma once
2 // AllToFinish, FirstToFinish, Probabilistic Choice
3 enum class OperatorType { ATF, FTF, PRB };
```

D.3.4 Outcome.h

```
1 /**
2  * @author Francesco Nieri
3  * @date 25/10/2024
4  * Class representing an outcome O_n in a system
5  */
6 #pragma once
7
8 #include "Observable.h"
9
10 class Outcome : public Observable
11 {
12 public:
13     Outcome(const std::string &name);
14     ~Outcome();
15 };
```

D.3.5 Probe.h

```
1 #pragma once
2 #include <string>
3 #include <vector>
```

```

4
5 #include "../maths/ConfidenceInterval.h"
6 #include "../maths/DeltaQ.h"
7 #include "Observable.h"
8 #include <map>
9 #include <memory>
10 #include <mutex>
11
12 /**
13  * @class Class representing a probe containing causal link
14  */
15
16 class Probe : public Observable
17 {
18     std::vector<std::shared_ptr<Observable>> causalLinks;
19
20     std::mutex calcMutex;
21
22     ConfidenceInterval calculatedInterval;
23     std::deque<DeltaQ> calculatedDeltaQHistory;
24
25 public:
26     Probe(const std::string &name);
27
28     /**
29      * @brief Construct a probe with its causal links
30      */
31     Probe(const std::string &name, std::vector<std::shared_ptr<
Observable>>);
32
33     ~Probe();
34     /**
35      * @brief calculate a Calculated deltaQ with bounds timeLowerBound
36      , timeUpperBound
37      * @param timeLowerBound
38      * @param timeUpperBound
39      * @return calculated DeltaQ
40      */
41     DeltaQ calculateCalculatedDeltaQ(uint64_t timeLowerBound, uint64_t
timeUpperBound);
42
43     /**
44      * @brief Get the representation of a calculated DeltaQ for
45      plotting
46      * @return the representation of a calculated DeltaQ
47      */
48     DeltaQRepr getCalculatedDeltaQRepr(uint64_t, uint64_t);
49
50     std::vector<Bound> getBounds() const;
51
52     std::vector<Bound> getObservedBounds() const;
53
54     std::vector<Bound> getCalculatedBounds() const;

```

```
55     void setCausalLinks(std::vector<std::shared_ptr<Observable>>
newCausalLinks)
56     {
57         causalLinks = newCausalLinks;
58     }
59
60     std::vector<std::shared_ptr<Observable>> getCausalLinks()
61     {
62         return causalLinks;
63     }
64 };
```

D.3.6 Sample.h

```
1  #pragma once
2
3  #include <stdint>
4
5  enum Status { SUCCESS, TIMEDOUT, FAILED };
6
7  /**
8   * @struct Sample represent an outcome instance
9   */
10 struct Sample {
11     std::uint64_t startTime;
12     std::uint64_t endTime;
13     double long elapsedTime;
14     Status status;
15 };
```

D.3.7 System.h

```
1  /**
2   * @author Francesco Nieri
3   * @date 26/10/2024
4   * Class representing a DeltaQ system
5   */
6  #pragma once
7
8  #include "../maths/DeltaQ.h"
9  #include "Observable.h"
10 #include "Operator.h"
11 #include "Outcome.h"
12 #include "Probe.h"
13 #include <memory>
14 #include <unordered_map>
15
16 class System
17 {
18     std::unordered_map<std::string, std::shared_ptr<Outcome>> outcomes
{}; ///< All outcome
19     std::unordered_map<std::string, std::shared_ptr<Operator>>
operators {}; ///< All operators
```



```

20     std::unordered_map<std::string, std::shared_ptr<Probe>> probes {};
21     /// < All probes
22     std::unordered_map<std::string, std::shared_ptr<Observable>>
23     observables {}; ///< The above grouped together
24
25     std::string systemDefinitionText; ///< The definition of the
26     system
27
28     bool recordingTrigger = false;
29     std::map<uint64_t, std::vector<Snapshot>> snapshots;
30
31 public:
32     System() = default;
33
34     [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<
35     Outcome>> &getOutcomes();
36
37     [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<
38     Probe>> &getProbes();
39
40     [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<
41     Operator>> &getOperators();
42
43     [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<
44     Observable>> &getObservables();
45
46     void setOutcomes(std::unordered_map<std::string, std::shared_ptr<
47     Outcome>> outcomesMap);
48
49     void setOperators(std::unordered_map<std::string, std::shared_ptr<
50     Operator>> operatorsMap);
51
52     void setProbes(std::unordered_map<std::string, std::shared_ptr<
53     Probe>> probesMap);
54
55     bool hasOutcome(const std::string &name);
56
57     bool hasOperator(const std::string &name);
58
59     std::shared_ptr<Outcome> getOutcome(const std::string &outcomeName
60 );
61
62     std::shared_ptr<Operator> getOperator(const std::string &);
63
64     bool hasProbe(const std::string &name);
65
66     std::shared_ptr<Probe> getProbe(const std::string &name);
67
68     std::shared_ptr<Observable> getObservable(const std::string &
69 observableName);
70
71     void setSystemDefinitionText(std::string &text);
72
73     std::string getSystemDefinitionText();
74
75     void setObservableParameters(std::string &, int, int);

```

```

64
65     /**
66      * @brief Add outcome instance for an observable, if it exists
67      */
68     void addSample(std::string &componentName, Sample &sample);
69
70     /**
71      * @brief Set all observables to record snapshots
72      */
73     void setRecording(bool);
74
75     bool isRecording() const;
76
77     /**
78      * @brief Add the snapshots of all observables for a trigger at
79      time t
80      */
81     void getObservablesSnapshotAt(std::uint64_t);
82
83     /**
84      * Get the snapshots of all observables for a trigger at time t
85      */
86     std::map<std::uint64_t, std::vector<Snapshot>> getAllSnapshots();
87
88     /**
89      * @brief Get the name of all components
90      */
91     std::vector<std::string> getAllComponentsName();
92
93     /**
94      * @deprecated This may be used in the future
95      * Calculate the resulting DeltaQ for the whole system
96      */
97     DeltaQ calculateDeltaQ();
98 };

```

D.4 maths

D.4.1 ConfidenceInterval.h

```

1  #pragma once
2
3  #include "DeltaQ.h"
4  #include <vector>
5  // Upper and lower confidence bounds of a DeltaQ's CDF
6  struct Bound {
7      double lowerBound {0};
8      double upperBound {1};
9      double mean {0};
10 };
11
12 /**
13  * @class ConfidenceInterval

```

```

14  * @brief Class representing the confidence interval of a window of
    DeltaQs
15  */
16  class ConfidenceInterval
17  {
18  private:
19      std::vector<Bound> bounds; ///< The bounds at each point
20      unsigned int numBins;
21      unsigned int size {0};
22      std::vector<double> cdfSum; ///< The sum of the cdf at each bin
23      std::vector<double> cdfSumSquares; ///< Variance at each bin
24      std::vector<unsigned int> cdfSampleCounts; ///< Sample per bins
25      double z {1}; ///< Confidence interval
26      void updateConfidenceInterval();
27
28  public:
29      /**
30       * @brief Default constructor, set to 0 bins
31       */
32      ConfidenceInterval();
33      /**
34       * @brief Constructor for ConfidenceInterval with number of bins
35       * @param numBins number of bins
36       */
37      ConfidenceInterval(int numBins);
38
39      void setNumBins(int newNumBins);
40      /**
41       * @brief Add DeltaQ to intervals
42       * @param DeltaQ DeltaQ to add
43       */
44      void addDeltaQ(const DeltaQ &);
45      /**
46       * @brief Remove DeltaQ from intervals
47       * @param DeltaQ DeltaQ to remove
48       */
49      void removeDeltaQ(const DeltaQ &);
50
51      /**
52       * @brief Get current confidence bounds
53       */
54      std::vector<Bound> getBounds() const;
55      /**
56       * @brief Get number of bins
57       * @return Number of bins
58       */
59      unsigned int getBins();
60      /**
61       * @brief Zero the confidence intervals
62       */
63      void reset();
64  };

```

D.4.2 DeltaQ.h

```
1  /**
2   * @author: Francesco Nieri
3   * @date 26/10/2024
4   * Class representing a DeltaQ
5   */
6
7  #ifndef DELTAQ_H
8  #define DELTAQ_H
9
10 #pragma once
11
12 #include "../diagram/Sample.h"
13 #include <array>
14 #include <ostream>
15 #include <vector>
16
17 #include "QTA.h"
18
19 class DeltaQ
20 {
21     double binWidth;
22     std::vector<double> pdfValues;
23     std::vector<double> cdfValues;
24     int bins {0};
25
26     QTA qta;
27     unsigned int totalSamples {0};
28
29     /**
30      * Calculate PDF and CDF values given samples from an outcome
31      */
32     void calculateDeltaQ(std::vector<Sample> &samples);
33     /**
34      * Calculate CDF given PDF values
35      */
36     void calculateCDF();
37     /**
38      * Calculate PDF from a given CDF
39      */
40     void calculatePDF();
41
42 public:
43     DeltaQ() = default;
44     DeltaQ(double binWidth);
45     DeltaQ(double binWidth, const std::vector<double> &values, bool
isPdf);
46     DeltaQ(double binWidth, std::vector<Sample> &);
47     DeltaQ(double binWidth, std::vector<Sample> &, int);
48     /**
49      * Getters
50      */
51     [[nodiscard]] const std::vector<double> &getPdfValues() const;
52     [[nodiscard]] const std::vector<double> &getCdfValues() const;
53     [[nodiscard]] double getBinWidth() const;
54     [[nodiscard]] int getBins() const;
```

```

55 [[nodiscard]] double pdfAt(int x) const;
56 [[nodiscard]] double cdfAt(int x) const;
57 [[nodiscard]] const unsigned int getTotalSamples() const;
58 [[nodiscard]] QTA getQTA() const;
59 void calculateQuartiles(std::vector<Sample> &);
60 void setBinWidth(double newWidth);
61 /**
62  * Operator Overloads
63  */
64 friend DeltaQ operator*(const DeltaQ &deltaQ, double constant);
65 friend DeltaQ operator*(double constant, const DeltaQ &deltaQ);
66 friend DeltaQ operator*(const DeltaQ &lhs, const DeltaQ &rhs);
67 friend DeltaQ operator+(const DeltaQ &lhs, const DeltaQ &rhs);
68 friend DeltaQ operator-(const DeltaQ &lhs, const DeltaQ &rhs);
69
70 friend std::ostream &operator<<(std::ostream &os, const DeltaQ &
deltaQ);
71
72 /**
73  * Comparison Operators
74  */
75 bool operator<(const DeltaQ &other) const;
76 bool operator>(const DeltaQ &other) const;
77
78 bool operator==(const DeltaQ &deltaQ) const;
79 [[nodiscard]] std::string toString() const;
80 };
81
82 #endif // DELTAQ_H

```

D.4.3 DeltaQOperations.h

```

1 #pragma once
2
3 #include "DeltaQ.h"
4
5 /**
6  * Perform discrete convolution between two DeltaQs
7  */
8 DeltaQ convolve(const DeltaQ &lhs, const DeltaQ &rhs);
9 /**
10  * Perform Fast Fourier Transform on two DeltaQs
11  */
12 DeltaQ convolveFFT(const DeltaQ &lhs, const DeltaQ &rhs);
13 DeltaQ convolveN(const std::vector<DeltaQ> &deltaQs);
14
15 /**
16  * Assume two independent outcomes with the same start event
17  * All-to-finish outcome occurs when both end events occur
18  * All-to-finish is defined as  $\Delta Q_{\{LTF(A,B)\}} \Delta Q_A * \Delta Q_B$ 
19  */
20 DeltaQ allToFinish(const std::vector<DeltaQ> &deltaQs);
21
22 /**
23  * Assume two independent outcomes with the same start event

```

```

24  * First-to-finish outcome occurs when at least one end event occurs
25  * We compute the probability that there are zero end events
26  * First-to-finish is defined as
27  *  $\Delta Q_{\{FTF(A, B)\}} = \Delta Q_A + \Delta Q_B - \Delta Q_A * \Delta Q_B$ 
28  */
29 DeltaQ firstToFinish(const std::vector<DeltaQ> &deltaQs);
30
31 /**
32  * Assume there are two possible outcomes OA and OB and
33  * exactly one outcome is chosen during each occurrence of a start
34  * event
35  * O_A occurs with probability p/(p+q)
36  * O_B occurs with probability q/(p+q)
37  * Therefore:
38  *  $\Delta Q_{\{PC(A,B)\}} = p/(p+q) \Delta Q_A + q/(p+q) \Delta Q_B$ 
39  */
39 DeltaQ probabilisticChoice(const std::vector<double> &probabilities,
40                             const std::vector<DeltaQ> &deltaQs);
41
42 DeltaQ rebin(const DeltaQ &source, double targetBinWidth);
43
44 /**
45  * Choose the highest size from a list of DeltaQs
46  */
46 int chooseLongestDeltaQSize(const std::vector<DeltaQ> &deltaQs);

```

D.4.4 DeltaQRepr.h

```

1
2 #pragma once
3
4 #include "ConfidenceInterval.h"
5 #include "DeltaQ.h"
6 #include <cstdint>
7 #include <vector>
8
9 /**
10  * Class storing a DeltaQ representation for graphical plotting
11  */
12
13 struct DeltaQRepr {
14     std::uint64_t time;
15     DeltaQ deltaQ;
16     std::vector<Bound> bounds;
17
18     DeltaQRepr()
19         : time(0)
20         , deltaQ()
21         , bounds()
22     {
23     }
24
25     // DeltaQRepr copy constructor
26     DeltaQRepr(const DeltaQRepr &other)
27         : time(other.time)

```

```

28         , deltaQ(other.deltaQ)
29         , bounds(other.bounds)
30     {
31     }
32
33     // DeltaQRepr move semantics
34     DeltaQRepr(DeltaQRepr &&other) noexcept
35         : time(other.time)
36         , deltaQ(std::move(other.deltaQ))
37         , bounds(std::move(other.bounds))
38     {
39     }
40
41     // DeltaQRepr assignment operators
42     DeltaQRepr &operator=(const DeltaQRepr &other)
43     {
44         if (&this != &other) {
45             time = other.time;
46             deltaQ = other.deltaQ;
47             bounds = other.bounds;
48         }
49         return *this;
50     }
51
52     DeltaQRepr &operator=(DeltaQRepr &&other) noexcept
53     {
54         if (&this != &other) {
55             time = other.time;
56             deltaQ = std::move(other.deltaQ);
57             bounds = std::move(other.bounds);
58         }
59         return *this;
60     }
61
62     // Constructor with parameters
63     DeltaQRepr(std::uint64_t t, const DeltaQ &dq, const std::vector<
Bound> &b)
64         : time(t)
65         , deltaQ(dq)
66         , bounds(b)
67     {
68     }
69 };

```

D.4.5 QTA.h

```

1 #ifndef QTA_H
2 #define QTA_H
3
4 #include <iostream>
5 #include <limits>
6 #define QTA_EPSILON std::numeric_limits<double>::epsilon()
7 /**
8  * @struct QTA Quantitative Timeliness Agreement for an observable
9  */

```

```

10 struct QTA {
11     double perc_25 {0}; ///< 25 percentile value
12     double perc_50 {0}; ///< 50 percentile value
13     double perc_75 {0}; ///< 75 percentile value
14     double cdfMax {0}; ///< Least failure rate
15     bool defined = false; ///< If defined or not
16
17     static QTA create(double p25, double p50, double p75, double cdf)
18     {
19         return QTA::create(p25, p50, p75, cdf, true);
20     }
21
22     static QTA create(double p25, double p50, double p75, double cdf,
23                       bool isDefined)
24     {
25         if (!(p25 <= p50 && p50 <= p75)) {
26             throw std::invalid_argument("Percentiles must be ordered:
27             perc_25 < perc_50 < perc_75.");
28         }
29         if (cdf < 0.0 || cdf - 1.0 > QTA_EPSILON) {
30             throw std::invalid_argument("cdfMax must be between 0 and
31             1 (exclusive lower bound, inclusive upper).");
32         }
33         return QTA {p25, p50, p75, cdf, isDefined};
34     }
35 };
36 #endif // QTA_H

```

D.4.6 Snapshot.h

```

1 #pragma once
2
3 #include "ConfidenceInterval.h"
4 #include "DeltaQ.h"
5 #include "DeltaQRepr.h"
6 #include <map>
7 #include <optional>
8
9 /**
10  * @class Snapshot
11  * @brief Represents a snapshot of observed and calculated DeltaQ
12  *        values along with QTAs
13  */
14 class Snapshot
15 {
16     std::string observableName; ///< Name of the observable being
17     tracked.
18     std::map<uint64_t, DeltaQRepr> observedDeltaQs; ///< Map of
19     observed DeltaQ values at times t.
20     std::map<uint64_t, DeltaQRepr> calculatedDeltaQs; ///< Map of
21     calculated DeltaQ values at times t
22     std::map<uint64_t, QTA> QTAs; ///< Map of QTAs at time t

```



```

20 public:
21     /// @brief Default constructor.
22     Snapshot() = default;
23
24     // --- Observed DeltaQ Methods ---
25     /**
26      * @brief Adds an observed DeltaQ to the snapshot.
27      * @param timestamp The lower time bound at which the DeltaQ was
28      * observed.
29      * @param deltaQ The DeltaQ value to store.
30      * @param bounds Confidence interval bounds for the DeltaQ.
31      */
32     void addObservedDeltaQ(std::uint64_t timestamp, const DeltaQ &
33     deltaQ, const std::vector<Bound> &bounds);
34
35     /**
36      * @brief Retrieves the oldest observed DeltaQ.
37      * @return The oldest DeltaQRepr (based on timestamp order).
38      */
39     DeltaQ getOldestObservedDeltaQ() const;
40
41     /// @brief Removes the oldest observed DeltaQ entry (FIFO order).
42     void removeOldestObservedDeltaQ();
43
44     // --- Calculated DeltaQ Methods ---
45     /**
46      * @brief Adds a calculated DeltaQ to the snapshot.
47      * @param timestamp The lower time bound at which the DeltaQ was
48      * calculated.
49      * @param deltaQ The DeltaQ value to store.
50      * @param bounds Confidence interval bounds for the DeltaQ.
51      */
52     void addCalculatedDeltaQ(std::uint64_t timestamp, const DeltaQ &
53     deltaQ, const std::vector<Bound> &bounds);
54
55     /**
56      * @brief Retrieves the oldest calculated DeltaQ.
57      * @return The oldest DeltaQRepr (based on timestamp order).
58      */
59     DeltaQ getOldestCalculatedDeltaQ() const;
60
61     /// @brief Removes the oldest calculated DeltaQ entry (FIFO order)
62     .
63     void removeOldestCalculatedDeltaQ();
64
65     // --- QTA Methods ---
66     /**
67      * @brief Adds a QTA to the snapshot.
68      * @param timestamp The time associated with the QTA.
69      * @param qta The QTA value to store.
70      */
71     void addQTA(std::uint64_t timestamp, const QTA &qta);
72
73     // --- Size Management ---
74     /// @return The number of observed DeltaQs stored.
75     std::size_t getObservedSize() const;

```

```

71
72     /// @return The number of calculated DeltaQs stored.
73     std::size_t getCalculatedSize() const;
74
75     /**
76      * @brief Truncates observed/calculated DeltaQs to a specified
77      * size (removes oldest entries).
78      * @param size The maximum number of entries to retain.
79      */
80     void resizeTo(size_t size);
81
82     // --- Name Management ---
83     /**
84      * @brief Sets the name of the observable.
85      * @param name New name for the observable.
86      */
87     void setName(const std::string &name);
88
89     /// @return The current observable name.
90     std::string getName() &;
91
92     // --- Data Retrieval ---
93     /// @return A vector of all observed DeltaQReprs (sorted by
94     timestamp).
95     std::vector<DeltaQRepr> getObservedDeltaQs() const &;
96
97     /// @return A vector of all calculated DeltaQReprs (sorted by
98     timestamp).
99     std::vector<DeltaQRepr> getCalculatedDeltaQs() const &;
100
101     /**
102      * @brief Retrieves an observed DeltaQ at a specific timestamp.
103      * @param timestamp The time to query.
104      * @return The DeltaQRepr if found, or `std::nullopt` otherwise.
105      */
106     std::optional<DeltaQRepr> getObservedDeltaQAtTime(std::uint64_t
107     timestamp);
108
109     /**
110      * @brief Retrieves a calculated DeltaQ at a specific timestamp.
111      * @param timestamp The time to query.
112      * @return The DeltaQRepr if found, or `std::nullopt` otherwise.
113      */
114     std::optional<DeltaQRepr> getCalculatedDeltaQAtTime(std::uint64_t
115     timestamp);
116
117     /// @return A vector of all QTAs (sorted by timestamp).
118     std::vector<QTA> getQTAs() const &;
119 };

```

D.4.7 TriggerManager.h

```

1 #pragma once
2
3 #include "TriggerTypes.h"

```

```
4 #include "Triggers.h"
5 #include <memory>
6 #include <optional>
7 #include <vector>
8
9 /**
10  * @class TriggerManager
11  * @brief Manages triggers for an observable
12  */
13 class TriggerManager
14 {
15 public:
16     struct Trigger {
17         TriggerType type;
18
19         TriggerDefs::Condition condition;
20
21         TriggerDefs::Action action;
22
23         bool enabled;
24
25         std::optional<int> sampleLimitValue;
26
27         Trigger(TriggerType t, TriggerDefs::Condition c, TriggerDefs::
Action a, bool e = true);
28     };
29     /**
30      * @brief Add a trigger for an observable,
31      * @param type The type of the trigger
32      * @param condition The condition for the trigger to be fired
33      * @param action The action to perform when fired
34      * @param enabled It the trigger is enabled
35      */
36     void addTrigger(TriggerType type, TriggerDefs::Condition condition
, TriggerDefs::Action action, bool enabled = true);
37
38     /**
39      * @brief Get all triggers set for a type
40      */
41     std::vector<Trigger> getTriggersByType(TriggerType type);
42
43     /**
44      * @brief Evaluate a DeltaQ to see if a trigger should be fired
45      * @param dq The DeltaQ to evaluate
46      * @param qta The qta to compare against to
47      * @param std::uint64_t Keep the time to log it if the trigger
fires
48      */
49     void evaluate(const DeltaQ &dq, const QTA &qta, std::uint64_t)
const;
50
51     /**
52      * @brief Remove triggers if their type matches the param type
53      * @param type The trigger's type'
54      */
55     void removeTriggersByType(TriggerType type);
```

```
56
57     /**
58      * @brief Remove all triggers
59      */
60     void clearAllTriggers();
61
62     /**
63      * @brief Enable/Disable all triggers with a type
64      * @param type The type of trigger to enable/disable
65      * @param enabled
66      */
67     void setTriggersEnabled(TriggerType type, bool enabled);
68
69     /**
70      * @brief Get all triggers
71      */
72     std::vector<Trigger> getAllTriggers() const;
73
74 private:
75     std::vector<Trigger> triggers_;
76 };
```

D.4.8 TriggerTypes.h

```
1 #ifndef TRIGGERTYPE_H
2 #define TRIGGERTYPE_H
3 #pragma once
4
5 enum class TriggerType {
6     SampleLimit,
7     QTAViolation,
8     Failure,
9     Hazard
10 };
11
12 #endif //TRIGGERTYPE_H
```

D.4.9 Triggers.h

```
1 #pragma once
2
3 #include "DeltaQ.h"
4 #include "QTA.h"
5 #include "TriggerTypes.h"
6 #include <functional>
7 #include <iostream>
8 #include <string>
9
10 namespace TriggerDefs
11 {
12     using Condition = std::function<bool(const DeltaQ &, const QTA &)>;
13     using Action = std::function<void(const DeltaQ &, const QTA &, std:::
        uint64_t)>;
```

```
14
15 namespace Conditions
16 {
17     Condition SampleLimit(int maxSamples);
18     Condition QTABounds();
19     Condition FailureRate(double threshold);
20 }
21
22 namespace Actions
23 {
24     Action LogToConsole(const std::string &message);
25     Action notify();
26     Action SaveSnapshot(const std::string &filename);
27 }
28 }
```

D.5 parser

D.5.1 SystemBuilder.h

```
1 #ifndef SYSTEMBUILDER_H
2 #define SYSTEMBUILDER_H
3
4 #include "../diagram/System.h"
5 #include "DQGrammarVisitor.h"
6 #include <memory>
7 #include <unordered_map>
8
9 /**
10  * @brief Visitor class that builds a System from a parsed grammar
11  * tree.
12  */
13 class SystemBuilderVisitor : public parser::DQGrammarVisitor
14 {
15 private:
16     std::unordered_map<std::string, std::shared_ptr<Outcome>> outcomes
17     ; ///< Map of outcome names to Outcome objects.
18     std::unordered_map<std::string, std::shared_ptr<Operator>>
19     operators; ///< Map of operator names to Operator objects.
20     std::unordered_map<std::string, std::shared_ptr<Probe>> probes;
21     ///< Map of probe names to Probe objects.
22
23     std::vector<std::string> definedProbes; ///< List of defined probe
24     names.
25     System system; ///< The final system constructed by the visitor.
26
27     std::string currentlyBuildingProbe; ///< Tracks the probe
28     currently being built for dependency management.
29     std::map<std::string, std::vector<std::string>> dependencies; ///<
30     Graph of probe dependencies.
31
32     std::unordered_map<std::string, std::vector<std::string>>
33     definitionLinks; ///< For debugging: links between definitions.
34 }
```

```

26     std::unordered_map<std::string, std::vector<std::vector<std::
27     string>>> operatorLinks; ///< For debugging: operator chains.
28     std::vector<std::string> systemLinks; ///< Top-level system
29     observable links.
30     std::unordered_set<std::string> allNames; ///< Tracks all used
31     names to detect duplicates.
32
33     /**
34     * @brief Checks the dependency graph for cycles.
35     * @throws std::invalid_argument if a cycle is detected.
36     */
37     void checkForCycles() const;
38
39     /**
40     * @brief Recursive utility to detect cycles in a graph.
41     * @param node Current node.
42     * @param visited Set of visited nodes.
43     * @param recursionStack Stack of nodes in the current DFS path.
44     * @return True if a cycle is found.
45     */
46     bool hasCycle(const std::string& node,
47                   std::set<std::string>& visited,
48                   std::set<std::string>& recursionStack) const;
49
50 public:
51     /**
52     * @brief Returns the constructed System.
53     * @return The system object.
54     */
55     System getSystem() const;
56
57     // Visitor overrides from DQGrammarVisitor:
58     std::any visitStart(parser::DQGrammarParser::StartContext *context
59     ) override;
60     std::any visitDefinition(parser::DQGrammarParser::
61     DefinitionContext *context) override;
62     std::any visitSystem(parser::DQGrammarParser::SystemContext *
63     context) override;
64     std::any visitComponent(parser::DQGrammarParser::ComponentContext
65     *context) override;
66     std::any visitBehaviorComponent(parser::DQGrammarParser::
67     BehaviorComponentContext *context) override;
68     std::any visitProbeComponent(parser::DQGrammarParser::
69     ProbeComponentContext *context) override;
70     std::any visitProbability_list(parser::DQGrammarParser::
71     Probability_listContext *context) override;
72     std::any visitComponent_list(parser::DQGrammarParser::
73     Component_listContext *context) override;
74     std::any visitOutcome(parser::DQGrammarParser::OutcomeContext *
75     context) override;
76     std::any visitComponent_chain(parser::DQGrammarParser::
77     Component_chainContext *context) override;
78 };
79
80 #endif // SYSTEMBUILDER_H

```

D.5.2 SystemErrorListener.h

```
1 #pragma once
2 #include "antlr4-runtime.h"
3 #include <iostream>
4 #include <stdexcept>
5
6 class SystemErrorListener : public antlr4::BaseErrorListener
7 {
8 public:
9     void syntaxError(antlr4::Recognizer *recognizer, antlr4::Token *
10        offendingSymbol, size_t line, size_t charPositionInLine, const std
11        ::string &msg,
12        std::exception_ptr e) override
13     {
14         throw std::runtime_error("Syntax error at line " + std::
15            to_string(line) + ":" + std::to_string(charPositionInLine) + " - "
16            + msg);
17     }
18 };
```

D.5.3 SystemParserInterface.h

```
1 #pragma once
2
3 #include "../diagram/System.h"
4 #include "DQGrammarLexer.h"
5 #include "DQGrammarParser.h"
6 #include "SystemBuilder.h"
7 #include "antlr4-runtime.h"
8 #include <memory>
9 #include <optional>
10 #include <string>
11
12 /**
13  * @class SystemParserInterface
14  * @brief Provides an interface for parsing system definitions from
15  *        files or strings.
16  */
17 class SystemParserInterface
18 {
19 public:
20     /**
21      * @brief Parses a system definition from a file.
22      * @param filename Path to the file containing system definition.
23      * @return Optional containing the parsed System if successful,
24      *         nullptr otherwise.
25      */
26     static std::optional<System> parseFile(const std::string &filename
27 );
28
29     /**
30      * @brief Parses a system definition from a string.
31      * @param input String containing system definition.
32      */
33 }
```

```
29     * @return Optional containing the parsed System if successful,  
    nullopt otherwise.  
30     */  
31     static std::optional<System> parseString(const std::string &input)  
    ;  
32  
33 private:  
34     /**  
35     * @brief Internal parsing method using ANTLR input stream.  
36     * @param input ANTLR input stream containing system definition.  
37     * @return Optional containing the parsed System if successful,  
    nullopt otherwise.  
38     */  
39     static std::optional<System> parseInternal(antlr4::  
    ANTLRInputStream &input);  
40 };
```

D.6 server

D.6.1 Server.h

```
1  #ifndef SERVER_H  
2  #define SERVER_H  
3  
4  #include "../diagram/System.h"  
5  #include <atomic>  
6  #include <condition_variable>  
7  #include <memory>  
8  #include <mutex>  
9  #include <netinet/in.h>  
10 #include <queue>  
11 #include <sys/socket.h>  
12 #include <thread>  
13  
14 /**  
15  * @class Server  
16  * @brief TCP server for handling client connections and Erlang  
    communication.  
17  */  
18 class Server  
19 {  
20 public:  
21     /**  
22     * @brief Constructs a Server instance.  
23     * @param port The TCP port to listen on.  
24     */  
25     Server(int port);  
26  
27     /**  
28     * @brief Destructor cleans up sockets and threads.  
29     */  
30     ~Server();  
31  
32     /**
```



```

33     * @brief Sends a command to the Erlang process.
34     * @param command The command string to send.
35     */
36     void sendToErlang(const std::string &command);
37
38     bool startServer(const std::string &ip = "0.0.0.0", int port =
39     8080);
40
41     void stopServer();
42
43     bool setErlangEndpoint(const std::string &ip, int port);
44
45     bool isServerRunning() const
46     {
47         return server_started;
48     }
49
50     /**
51     * @brief Stops the server and worker threads.
52     */
53     void stop();
54 private:
55     /**
56     * @brief Main server loop running in a separate thread.
57     */
58     void run();
59
60     int server_fd; ///< Server socket file descriptor
61     int new_socket; ///< Client socket file descriptor
62     struct sockaddr_in address; ///< Server address structure
63     int port; ///< Listening port number
64
65     std::thread serverThread; ///< Thread for server operations
66
67     /**
68     * @brief Updates the system reference from Application.
69     */
70     void updateSystem();
71
72     /**
73     * @brief Parses messages from Erlang.
74     * @param buffer The message buffer.
75     * @param len Length of the message.
76     */
77     void parseErlangMessage(const char *buffer, int len);
78
79     std::shared_ptr<System> system; ///< Reference to the system being
80     monitored
81
82     std::vector<std::thread> clientThreads; ///< Active client handler
83     threads
84     std::mutex clientsMutex; ///< Mutex for client threads access
85     std::atomic<bool> running {false}; ///< Server running state flag
86
87     /**

```

```
86     * @brief Handles communication with a client.
87     * @param clientSocket The client socket file descriptor.
88     */
89     void handleClient(int clientSocket);
90
91     /**
92     * @brief Cleans up finished client threads.
93     */
94     void cleanupThreads();
95
96     int erlang_socket = -1; ///< Socket for Erlang communication
97     std::mutex erlangMutex; ///< Mutex for Erlang socket operations
98
99     /**
100    * @brief Establishes connection to Erlang.
101    * @return true if connection succeeded.
102    */
103    bool connectToErlang();
104
105    int client_socket; ///< Current client socket
106
107    // Asynchronous sample processing
108    std::queue<std::pair<std::string, Sample>> sampleQueue; ///<
Sample processing queue
109    std::mutex queueMutex; ///< Mutex for queue access
110    std::condition_variable queueCond; ///< Condition variable for
queue notifications
111    std::thread workerThread; ///< Worker thread for sample processing
112    bool shutdownWorker = false; ///< Flag to signal worker thread
shutdown
113
114    std::string erlang_ip = "127.0.0.1"; ///< Erlang server IP
115    int erlang_port = 8081; ///< Erlang server port
116    std::string server_ip = "0.0.0.0"; ///< Current C++ server IP
117    bool server_started = false;
118 };
119
120 #endif
```

Appendix E

Build Configuration Files

E.1 Root

E.1.1 CMakeLists.txt

```
1 add_library(${PREFIX}_application
2     Application.cpp
3     Application.h
4 )
5
6
7 target_link_libraries(${PREFIX}_application
8     PRIVATE
9     ${PREFIX}_server
10 )
11
12 # Make sure other libraries can find Application headers
13 target_include_directories(${PREFIX}_application
14     PUBLIC ${CMAKE_SOURCE_DIR}
15 )
16
17 add_subdirectory(dashboard)
18 add_subdirectory(diagram)
19 add_subdirectory(maths)
20 add_subdirectory(server)
21 add_subdirectory(parser)
22
23 add_executable(RealTimeDeltaQSD main.cpp)
24
25 target_include_directories(RealTimeDeltaQSD PUBLIC ${CMAKE_SOURCE_DIR}
26     })
27
28 target_link_libraries(RealTimeDeltaQSD ${PREFIX}_server ${PREFIX}
29     _diagram ${PREFIX}_dashboard ${PREFIX}_parser)
```

E.2 dashboard

E.2.1 CMakeLists.txt

```
1 add_library(${PREFIX}_dashboard
2     ColorRegistry.cpp
3     ColorRegistry.h
4     CustomLegendEntry.h
5     CustomLegendEntry.cpp
6     CustomLegendPanel.h
7     CustomLegendPanel.cpp
8     DelaySettingsWidget.h
9     DelaySettingsWidget.cpp
10    DQPlotController.h
11    DQPlotController.cpp
12    DQPlotList.h
13    DQPlotList.cpp
14    DeltaQPlot.h
15    DeltaQPlot.cpp
16    MainWindow.h
17    MainWindow.cpp
18    NewPlotList.h
19    NewPlotList.cpp
20    ObservableSettings.h
21    ObservableSettings.cpp
22    PollingRateWidget.h
23    PollingRateWidget.cpp
24    Sidebar.h
25    Sidebar.cpp
26    SnapshotViewerWindow.h
27    SnapshotViewerWindow.cpp
28    StubControlWidget.h
29    StubControlWidget.cpp
30    SystemCreationWidget.h
31    SystemCreationWidget.cpp
32    QTAInputWidget.cpp
33    QTAInputWidget.h
34    TriggersTab.cpp
35    TriggersTab.h
36 )
37
38
39 find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS Core Gyu Widgets
40             Charts Graphs)
41
42 find_package(Qt6 REQUIRED COMPONENTS Core Gui Charts Widgets Graphs)
43
44 target_link_libraries(${PREFIX}_dashboard
45     PUBLIC Qt6::Core Qt6::Gui Qt6::Widgets Qt6::Charts Qt6::Graphs
46     ${PREFIX}_parser
47 )
48
49 target_include_directories(${PREFIX}_dashboard
50     PUBLIC
51     ${CMAKE_SOURCE_DIR}
52     ${PREFIX}_parser
53 )
```

E.3 diagram

E.3.1 CMakeLists.txt

```
1 add_library(${PREFIX}_diagram
2     Outcome.h
3     Outcome.cpp
4     Operator.h
5     Operator.cpp
6     Probe.h
7     Probe.cpp
8     System.h
9     System.cpp
10    Sample.h
11    Observable.h
12    Observable.cpp
13 )
14
15 target_link_libraries(${PREFIX}_diagram
16     PUBLIC
17         ${PREFIX}_maths
18         ${PREFIX}_application
19 )
20
21
22 target_include_directories(${PREFIX}_diagram
23     PUBLIC
24         ${CMAKE_SOURCE_DIR}
25 )
26
27 add_executable(diagram
28     main.cpp
29 )
30
31 target_link_libraries(diagram
32     PRIVATE
33         ${PREFIX}_diagram
34 )
```

E.4 maths

E.4.1 CMakeLists.txt

```
1 add_library(${PREFIX}_maths
2     ConfidenceInterval.h
3     ConfidenceInterval.cpp
4     DeltaQ.h
5     DeltaQ.cpp
6     DeltaQRepr.h
7     DeltaQOperations.h
```

```
8     DeltaQOperations.cpp
9     QTA.h
10    Triggers.h
11    Triggers.cpp
12    TriggerTypes.h
13    TriggerManager.cpp
14    TriggerManager.h
15    Snapshot.h
16    Snapshot.cpp
17 )
18
19 target_include_directories(${PREFIX}_maths
20     PUBLIC
21     ${CMAKE_SOURCE_DIR}
22 )
23
24 target_link_libraries(${PREFIX}_maths
25     PRIVATE
26     fftw3
27 )
28
29 add_executable(maths
30     main.cpp
31 )
32
33
34 target_link_libraries(maths
35     PRIVATE
36     ${PREFIX}_maths
37 )
```

E.5 parser

E.5.1 CMakeLists.txt

```
1
2 find_package(antlr4-runtime REQUIRED CONFIG)
3
4 if(NOT antlr4-runtime_FOUND)
5     # Manually specify paths (adjust according to your installation)
6     set(ANTLR4_INCLUDE_DIR "/usr/local/include")
7     set(ANTLR4_LIB_DIR "/usr/local/lib")
8
9     find_library(ANTLR4_RUNTIME_LIB antlr4-runtime PATHS ${
10     ANTLR4_LIB_DIR})
11     if(NOT ANTLR4_RUNTIME_LIB)
12         message(FATAL_ERROR "ANTLR4 runtime library not found")
13     endif()
14
15     add_library(antlr4-runtime SHARED IMPORTED)
16     set_target_properties(antlr4-runtime PROPERTIES
17         IMPORTED_LOCATION ${ANTLR4_RUNTIME_LIB}
18         INTERFACE_INCLUDE_DIRECTORIES ${ANTLR4_INCLUDE_DIR}
19     )
```

```
19 endif()
20
21 set(ANTLR_GENERATED_DIR ${CMAKE_CURRENT_BINARY_DIR}/generated)
22 file(MAKE_DIRECTORY ${ANTLR_GENERATED_DIR})
23
24 # Generate ANTLR files
25 find_program(ANTLR_EXECUTABLE antlr4)
26
27 execute_process(
28     COMMAND ${ANTLR_EXECUTABLE}
29         -Dlanguage=Cpp
30         -visitor -no-listener
31         -o ${ANTLR_GENERATED_DIR}
32         -package parser
33         ${CMAKE_CURRENT_SOURCE_DIR}/DQGrammar.g4
34     WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
35     OUTPUT_VARIABLE ANTLR_OUTPUT
36     ERROR_VARIABLE ANTLR_ERROR
37     RESULT_VARIABLE ANTLR_RESULT
38 )
39 if(NOT ANTLR_RESULT EQUAL 0)
40     message(FATAL_ERROR "ANTLR generation failed: ${ANTLR_ERROR}")
41 endif()
42
43 message(STATUS "ANTLR generated: ${ANTLR_OUTPUT}")
44
45 add_library(${PREFIX}_parser
46     ${ANTLR_GENERATED_DIR}/DQGrammarLexer.h
47     ${ANTLR_GENERATED_DIR}/DQGrammarLexer.cpp
48     ${ANTLR_GENERATED_DIR}/DQGrammarParser.h
49     ${ANTLR_GENERATED_DIR}/DQGrammarParser.cpp
50     ${ANTLR_GENERATED_DIR}/DQGrammarBaseVisitor.cpp
51     ${ANTLR_GENERATED_DIR}/DQGrammarVisitor.cpp
52     ${ANTLR_GENERATED_DIR}/DQGrammarBaseVisitor.h
53     ${ANTLR_GENERATED_DIR}/DQGrammarVisitor.h
54     SystemBuilder.h
55     SystemBuilder.cpp
56     SystemErrorListener.h
57     SystemParserInterface.h
58     SystemParserInterface.cpp
59 )
60 target_link_libraries(${PREFIX}_parser
61     PUBLIC antlr4-runtime
62 )
63
64 target_include_directories(${PREFIX}_parser
65     PUBLIC
66     ${CMAKE_SOURCE_DIR}
67     ${ANTLR_GENERATED_DIR}
68     ${ANTLR4_INCLUDE_DIR}
69 )
70
71 add_executable(parser
72     main.cpp
73 )
74
```

```
75 target_link_libraries(parser
76     PRIVATE
77     ${PREFIX}_parser
78     antlr4-runtime
79     ${PREFIX}_diagram
80 )
```

E.6 server

E.6.1 CMakeLists.txt

```
1 add_library(${PREFIX}_server
2     Server.cpp
3     Server.h
4 )
5
6 target_link_libraries(${PREFIX}_server
7     PUBLIC
8     ${PREFIX}_diagram
9 )
10
11 target_include_directories(${PREFIX}_server
12     PUBLIC
13     ${CMAKE_SOURCE_DIR}
14 )
```


Appendix F

Erlang Source Files

F.1 Root

F.1.1 dqsd_otel.erl

```
1
2 -module(dqsd_otel).
3 -behaviour(application).
4 -author("Francesco Nieri").
5
6 -export([start/0, start_span/1, start_span/2, end_span/2, fail_span
7         /1, with_span/2, with_span/3, span_process/3]).
8 -export([start/2, stop/1]).
9 -export([init_ets/0]).
10 -export([set_stub_running/1]).
11 -export([handle_c_message/1]).
12
13 -include_lib("opentelemetry_api/include/otel_tracer.hrl").
14
15 %% @moduledoc
16 %% `dqsd_otel` is an Erlang module built on top of OpenTelemetry
17 %% to pair with the DeltaQ oscilloscope. It tracks spans start, end,
18 %% uses a custom timeout defined by the user in the oscilloscope,
19 %%
20 %% Features:
21 %% - Span lifecycle management (start/end/fail/timeout)
22 %% - Dynamic timeouts for spans
23 %% - Supports toggling stub behavior at runtime
24 %%
25 %% Usage:
26 %% {Ctx, Pid} = dqsd_otel:start_span(<<"my_span">>).
27 %% dqsd_otel:end_span(Ctx, Pid).
28 %% dqsd_otel:fail_span(Pid)
29
30 %%%=====
31 %%% Application Callbacks
32 %%%=====
```

```

33
34
35 start(_Type, _Args) ->
36     dqsd_otel_sup:start_link().
37
38 stop(_State) ->
39     ok.
40
41 init_ets() ->
42     ets:new(timeout_registry, [named_table, public, set]),
43     ets:new(otel_state, [named_table, public, set]),
44     ets:insert(otel_state, {stub_running, false}),
45     {ok, self()}.
46
47
48 %%%=====
49 %%% For testing purposes
50 %%% =====
51
52 set_stub_running(Bool) when is_boolean(Bool) ->
53     ets:insert(otel_state, {stub_running, Bool}),
54     io:format("Stub running set to: ~p~n", [Bool]),
55     ok.
56
57 %%%=====
58 %%% Public API
59 %%%=====
60 %% @doc Starts the otel_wrapper application and all dependencies.
61 -spec start() -> {ok, [atom()]} | {error, term()}.
62 start() ->
63     application:ensure_all_started(dqsd_otel).
64
65 %% @doc Starts a span with the given name, if the stub is running.
66 %% Returns a tuple of SpanContext and the internal span process PID or
67 %% `ignore`.
68 -spec start_span(binary()) -> {opentelemetry:span_ctx(), pid() |
69     ignore}.
70 start_span(Name) ->
71     SpanCtx = ?start_span(Name),
72     case ets:lookup(otel_state, stub_running) of
73         [{_, true}] ->
74             case ets:lookup(timeout_registry, Name) of
75                 [{_, T}] ->
76                     StartTime = erlang:system_time(nanosecond),
77                     Pid = spawn(?MODULE, span_process, [Name,
78                         StartTime, T]),
79                     {SpanCtx, Pid};
80                 [] ->
81                     {SpanCtx, ignore}
82             end;
83         _ ->
84             {SpanCtx, ignore}
85     end.
86
87 %% @doc Starts a span with attributes.

```

```

85 -spec start_span(binary(), map()) -> {opentelemetry:span_ctx(), pid()
    | ignore}.
86 start_span(Name, Attrs) when is_map(Attrs) ->
87     SpanCtx = ?start_span(Name, Attrs),
88     case ets:lookup(otel_state, stub_running) of
89         [{_, true}] ->
90             case ets:lookup(timeout_registry, Name) of
91                 [{_, T}] ->
92                     StartTime = erlang:system_time(nanosecond),
93                     Pid = spawn(?MODULE, span_process, [Name,
94                         StartTime, T]),
95                     {SpanCtx, Pid};
96                 [] ->
97                     {SpanCtx, ignore}
98             end;
99         _ ->
100             {SpanCtx, ignore}
101     end.
102 %% @doc Ends the span and reports it, unless stub is disabled or Pid
    is `ignore`.
103 -spec end_span(opentelemetry:span_ctx(), pid() | ignore) -> ok | term
    ().
104 end_span(Ctx, Pid) ->
105     ?end_span(Ctx),
106     case Pid of
107         ignore -> ok;
108         _ when is_pid(Pid) ->
109             Pid ! {<<"end_span">>, erlang:system_time(nanosecond)}
110     end.
111
112 %% @doc Fail the span and reports it to the oscilloscope, unless stub
    is disabled or Pid is `ignore`.
113 -spec fail_span(pid() | ignore) -> ok | term().
114 fail_span(Pid) ->
115     case Pid of
116         ignore -> ok;
117         _ when is_pid(Pid) ->
118             Pid ! {<<"fail_span">>, erlang:system_time(nanosecond)}
119     end.
120
121
122
123 %% @doc Executes Fun inside a span with attributes.
124 -spec with_span(binary(), fun(() -> any())) -> any().
125 with_span(Name, Fun) ->
126     ?with_span(Name, #{},
127         fun(_SpanCtx) ->
128             Pid = start_with_span(Name),
129             Result = Fun(),
130             end_with_span(Pid),
131             Result
132         end).
133
134 %% @doc Executes Fun inside a span with attributes.
135

```

```

136 -spec with_span(binary(), fun(() -> any()), map()) -> any().
137 with_span(Name, Fun, Attrs) when is_map(Attrs), is_function(Fun, 0) ->
138     ?with_span(Name, Attrs,
139         fun(_SpanCtx) ->
140             Pid = start_with_span(Name),
141             Result = Fun(),
142             end_with_span(Pid),
143             Result
144         end).
145
146 start_with_span(Name) ->
147     case ets:lookup(otel_state, stub_running) of
148     [{_, true}] ->
149         case ets:lookup(timeout_registry, Name) of
150         [{_, T}] ->
151             StartTime = erlang:system_time(nanosecond),
152             Pid = spawn(?MODULE, span_process, [Name,
153                 StartTime, T]),
154             Pid;
155         [] ->
156             ignore
157         end;
158     _ ->
159         ignore
160     end.
161
162 end_with_span(Pid) ->
163     case Pid of
164     ignore -> ok;
165     _ when is_pid(Pid) ->
166         Pid ! {<<"end_span">>, erlang:system_time(nanosecond)}
167     end.
168
169
170 %%%=====
171 %%% Span Worker
172 %%%=====
173
174 span_process(NameBin, StartTime, Timeout) ->
175     Deadline = StartTime + (Timeout * 1000000),
176     Timer = erlang:send_after(Timeout, self(), {<<"timeout">>,
177         Deadline}),
178     receive
179     {<<"fail_span">>, EndTime} ->
180         io:format("failure"),
181         erlang:cancel_timer(Timer),
182         send_span(NameBin, StartTime, EndTime, <<"fa">>);
183     {<<"end_span">>, EndTime} ->
184         erlang:cancel_timer(Timer),
185         send_span(NameBin, StartTime, EndTime, <<"ok">>);
186     {<<"timeout">>, Deadline} ->
187         send_span(NameBin, StartTime, Deadline, <<"to">>)
188     end.
189

```

```

190 %%%=====
191 %%% Handle Incoming Messages from C
192 %%%=====
193
194 handle_c_message(Bin) when is_binary(Bin) ->
195     case binary:split(Bin, <<";">>, [global]) of
196     [<<"set_timeout">>, Name, TimeoutBin] ->
197         case string:to_integer(binary_to_list(TimeoutBin)) of
198         {Timeout, _} when is_integer(Timeout) ->
199             ets:insert(timeout_registry, {Name, Timeout}),
200             io:format("dqsd_otel: Timeout set: ~p = ~p~n", [
201 Name, Timeout]);
202             ->
203                 io:format("dqsd_otel: Invalid timeout: ~p~n", [
204 TimeoutBin])
205             end;
206         [<<"start_stub">>] ->
207             ets:insert(otel_state, {stub_running, true}),
208             io:format("dqsd_otel: Stub enabled~n");
209         [<<"stop_stub">>] ->
210             ets:insert(otel_state, {stub_running, false}),
211             io:format("dqsd_otel: Stub stopped~n");
212         ->
213             io:format("dqsd_otel: Unknown command: ~p~n", [Bin])
214         end.
215
216 %%%=====
217 %%% Sending Span Data to C
218 %%%=====
219
220 send_span(NameBin, Start, End, StatusBin) ->
221     Data = io_lib:format("n:~s;b:~p;e:~p;s:~s~n", [
222 NameBin,
223 Start,
224 End,
225 StatusBin
226 ]),
227     dqsd_otel_tcp_client:send_span(lists:flatten(Data)).

```

F.1.2 dqsd_otel_app.erl

```

1 %%%-----
2 %%% @doc dqsd_otel public API
3 %%% @end
4 %%%-----
5
6 -module(dqsd_otel_app).
7
8 -behaviour(application).
9
10 -export([start/2, stop/1]).
11
12 start(_StartType, _StartArgs) ->
13     dqsd_otel_sup:start_link().
14

```

```
15 stop(_State) ->
16     ok.
```

F.1.3 dqsd_otel_sup.erl

```
1
2 -module(dqsd_otel_sup).
3 -behaviour(supervisor).
4 -export([start_link/0, init/1]).
5
6 start_link() ->
7     supervisor:start_link({local, ?MODULE}, ?MODULE, []).
8
9 init([]) ->
10     ChildSpecs = [
11         #{id => ets_init,
12           start => {dqsd_otel, init_ets, []},
13           restart => temporary,
14           shutdown => brutal_kill,
15           type => worker,
16           modules => [dqsd_otel]},
17
18         #{id => tcp_server,
19           start => {dqsd_otel_tcp_server, start_link, []},
20           restart => permanent,
21           shutdown => brutal_kill,
22           type => worker,
23           modules => [dqsd_otel_tcp_server]},
24
25         #{id => tcp_client,
26           start => {dqsd_otel_tcp_client, start_link, []},
27           restart => permanent,
28           shutdown => brutal_kill,
29           type => worker,
30           modules => [dqsd_otel_tcp_client]}
31     ],
32     {ok, {{one_for_one, 5, 10}, ChildSpecs}}.
```

F.1.4 dqsd_otel_tcp_client.erl

```
1
2 -module(dqsd_otel_tcp_client).
3 -behaviour(gen_server).
4
5 -export([start_link/0, send_span/1]).
6 -export([init/1, handle_call/3, handle_cast/2, handle_info/2,
7 terminate/2]).
8 -export([disconnect/0, try_connect/2]).
9
10 -define(SERVER, ?MODULE).
11
12 -record(state, {
13     socket = undefined,
```

```

13     logged_disconnected = false
14 }).
15
16 %% Public API
17
18 start_link() ->
19     gen_server:start_link({local, ?SERVER}, ?MODULE, [], []).
20
21 send_span(Data) ->
22     gen_server:cast(?SERVER, {send, Data}).
23
24 %% @doc Connect to a custom IP and Port. Replaces any existing
25 connection.
26 -spec try_connect(string() | binary(), integer()) -> ok.
27 try_connect(IP, Port) ->
28     gen_server:cast(?SERVER, {try_connect, IP, Port}).
29
30 -spec disconnect() -> ok.
31 disconnect() ->
32     gen_server:cast(?SERVER, disconnect).
33
34 %% gen_server callbacks
35
36 init([]) ->
37     State = #state{socket = undefined, logged_disconnected = false},
38     {ok, State}.
39
40 handle_cast({send, _Data}, State = #state{socket = undefined,
41 logged_disconnected = false}) ->
42     io:format("dqsd_otel: No socket. Dropping subsequent spans.\n"),
43     {noreply, State#state{logged_disconnected = true}};
44
45 handle_cast({send, _Data}, State = #state{socket = undefined}) ->
46     %% Already logged, suppress further logs
47     {noreply, State};
48
49 handle_cast({send, Data}, State = #state{socket = Socket}) ->
50     case gen_tcp:send(Socket, Data) of
51         ok ->
52             {noreply, State};
53         {error, Reason} ->
54             io:format("dqsd_otel: TCP send failed: ~p~n", [Reason]),
55             NewState = State#state{socket = undefined,
56 logged_disconnected = true},
57             {noreply, NewState}
58     end;
59
60 handle_cast({try_connect, IP, Port}, State) ->
61     IPStr = case IP of
62         Bin when is_binary(Bin) -> binary_to_list(Bin);
63         Str when is_list(Str) -> Str
64     end,

```

```

62         case gen_tcp:connect(IPStr, Port, [binary, {active, false}])
of
63         {ok, Socket} ->
64             io:format("dqsdtel: Wrapper connected to ~s:~p~n", [
IPStr, Port]),
65
66             case State#state.socket of
67                 undefined -> ok;
68                 OldSocket -> catch gen_tcp:close(OldSocket)
69             end,
70             {noreply, State#state{socket = Socket,
logged_disconnected = false}};
71         {error, Reason} ->
72             io:format("dqsdtel: Connection to ~s:~p failed: ~p~n
", [IPStr, Port, Reason]),
73             {noreply, State}
74         end;
75
76         handle_cast(disconnect, State = #state{socket = undefined}) ->
77             io:format("dqsdtel: Socket already disconnected.~n"),
78             {noreply, State};
79
80         handle_cast(disconnect, State = #state{socket = Socket}) ->
81             io:format("dqsdtel: Disconnecting TCP socket.~n"),
82             gen_tcp:close(Socket),
83             {noreply, State#state{socket = undefined, logged_disconnected
= false}}.
84
85         handle_info(_, State) ->
86             {noreply, State}.
87
88         handle_call(_, _From, State) ->
89             {reply, ok, State}.
90
91         terminate(_Reason, State) when is_record(State, state) ->
92             case State#state.socket of
93                 undefined -> ok;
94                 Socket -> gen_tcp:close(Socket)
95             end;
96         terminate(_Reason, _Other) ->
97             ok.

```

F.1.5 dqsdtel_tcp_server.erl

```

1  %%%-----
2  %%% @doc
3  %%% TCP server for receiving and processing messages from an
   oscilloscope.
4  %%% Allows starting and stopping a TCP server on a given IP and port.
5  %%%
6  %%% When a line-delimited binary message is received, it is passed to
7  %%% `dqsdtel:handle_c_message/1` for further processing.
8  %%%
9  %%%-----
10

```



```

11 -module(dqsd_otel_tcp_server).
12 -behaviour(gen_server).
13
14 %%% API
15 -export([start_link/0, start_server/2, stop_server/0]).
16
17 %%% gen_server callbacks
18 -export([init/1, handle_call/3, handle_cast/2, handle_info/2]).
19
20 -record(state, {
21     socket,      %% Listening socket
22     acceptor     %% PID of acceptor process
23 }).
24
25 -spec start_link() -> {ok, pid()} | ignore | {error, term()}.
26 start_link() ->
27     gen_server:start_link({local, ?MODULE}, ?MODULE, [], []).
28
29 %-----
30 %% @doc Starts the TCP listener on the given IP and Port.
31 %% Spawns an accept loop to handle incoming connections.
32 %%
33 %% IP can be a string, binary, or tuple (e.g., "127.0.0.1" or {
34 %%     127,0,0,1}).
35 %% Port is an integer.
36 %-----
37 -spec start_server(string() | binary() | tuple(), integer()) -> ok | {
38     error, term()}.
39 start_server(IP, Port) ->
40     gen_server:call(?MODULE, {start, IP, Port}).
41
42 %-----
43 %% @doc Stops the TCP server and closes the listening socket.
44 %% Also shuts down the acceptor process.
45 %%
46 %% @spec stop_server() -> ok | {error, not_running}
47 %-----
48 stop_server() ->
49     gen_server:call(?MODULE, stop).
50
51 %-----
52 %% @private
53 %% gen_server init callback.
54 %% Initializes the state without an open socket or acceptor.
55 %-----
56 init([]) ->
57     {ok, #state{}}.
58
59 %-----
60 %% @private
61 %% Handles the start request. Binds to the given IP and port.
62 %% On success, starts the accept loop.
63 %-----
64 handle_call({start, IP, Port}, _From, State) ->

```

```

64     Options = [binary, {packet, line}, {active, false}, {reuseaddr,
65     true}, {ip, parse_ip(IP)}],
66     case gen_tcp:listen(Port, Options) of
67         {ok, Socket} ->
68             Acceptor = spawn(fun() -> accept_loop(Socket) end),
69             io:format("dqsd_otel: Listening socket started on ~p:~p~n"
70             , [IP, Port]),
71             {reply, ok, State#state{socket = Socket, acceptor =
72             Acceptor}};
73         {error, Reason} ->
74             io:format("dqsd_otel: Could not start listening socket on
75             ~p:~p~n", [IP, Port]),
76             {reply, {error, Reason}, State}
77     end;
78
79     %%-----
80     %% @private
81     %% Handles stop request. Stops acceptor and closes socket.
82     %% Returns error if server is not running.
83     %%-----
84     handle_call(stop, _From, #state{socket = undefined, acceptor =
85     undefined} = State) ->
86         io:format("dqsd_otel: Server not running.~n"),
87         {reply, {error, not_running}, State};
88
89     handle_call(stop, _From, #state{socket = Socket, acceptor = Acceptor})
90     ->
91         catch exit(Acceptor, shutdown),
92         gen_tcp:close(Socket),
93         io:format("dqsd_otel: Stopped listening from oscilloscope~n"),
94         {reply, ok, #state{socket = undefined, acceptor = undefined}}.
95
96     handle_cast(_, State) -> {noreply, State}.
97     handle_info(_, State) -> {noreply, State}.
98
99     %%-----
100    %% @private
101    %% Accept loop that runs in a separate process.
102    %% Accepts TCP connections and spawns a handler for each one.
103    %%-----
104    accept_loop(ListenSocket) ->
105        {ok, Socket} = gen_tcp:accept(ListenSocket),
106        spawn(fun() -> handle_client(Socket) end),
107        accept_loop(ListenSocket).
108
109    %%-----
110    %% @private
111    %% Handles a client socket connection. Reads line-by-line.
112    %% Forwards trimmed binary lines to `dqsd_otel:handle_c_message/1`.
113    %%-----
114    handle_client(Socket) ->
115        case gen_tcp:recv(Socket, 0) of
116            {ok, Line} ->
117                Trimmed = binary:replace(Line, <<"\n">>, <<>>, [global]),
118                dqsd_otel:handle_c_message(Trimmed),

```

```
114         handle_client(Socket);
115         {error, closed} ->
116             gen_tcp:close(Socket)
117     end.
118
119     %%-----
120     %% @private
121     %% Parses an IP address from string, binary, or already-parsed tuple.
122     %%-----
123     parse_ip("127.0.0.1") -> {127,0,0,1};
124     parse_ip("0.0.0.0") -> {0,0,0,0};
125     parse_ip(IP) when is_list(IP) ->
126         {ok, Addr} = inet:parse_address(IP), Addr;
127     parse_ip(IP) -> IP.
```

Appendix G

Erlang Application Files

G.1 Root

G.1.1 dqsd_otel.app.src

```
1 {application, dqsd_otel,  
2   [{description, "An OpenTelemetry Wrapper for the  $\Delta Q$  Oscilloscope"}  
3   ,  
4     {vsn, "git"},  
5     {modules, [dqsd_otel, dqsd_otel_sup, dqsd_otel_tcp_client,  
6       dqsd_otel_tcp_server]},  
7     {registered, [dqsd_otel_sup]},  
8     {applications, [kernel, stdlib, opentelemetry_exporter,  
9       opentelemetry, opentelemetry_api]},  
10    {mod, {dqsd_otel, []}},  
11    {env, []},  
    {maintainers, ["Francesco Nieri"]},  
    {licenses, ["Apache 2.0"]},  
    {links, [{"https://github.com/fnieri/dqsd_otel"}]}}.
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl