UCLouvain

École polytechnique de Louvain

The △Q Oscilloscope: Real-Time Observation of Large Erlang Applications using △QSD

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. Many important questions are hard to answer. What happens when there is an overload? How can we feel something is wrong with the system early enough?

The purpose of the thesis is to create the ΔQ oscilloscope, a real time graphical dashboard that can be used to study the behaviour of running Erlang systems and explore tradeoffs in system design. It is based on the principles of ΔQSD.

Furthermore, we have developed an Erlang interface, the ΔQ adapter (`dqsd_otel`). It allows sending real time insights about the running system to the oscilloscope. The adapter works on top of the OpenTelemetry framework macros. This allows for users who have already instrumented their applications with OpenTelemetry to easily integrate the interface.

The oscilloscope performs statistical computations on the time series data it receives from the adapter and displays the results in real time, thanks to the ΔQSD paradigm. We provide a set of triggers 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. Additionally, the implementation of efficient algorithms for complex operations, such as convolution, allows for the computations to be done rapidly on precise representations of components.

We introduce the work by giving a summary of ΔQSD concepts. We also provide a summary of the observability tools available for Erlang, namely, OpenTelemetry. We then present the overall design of the project, describing how to build a bridge from OpenTelemetry to the oscilloscope. 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 ΔQSD 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. They were 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 Sloughter) for their help in the EEF Slack, which helped me understand 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 Problem . . . 1
- 1.2 Approach . . . 1
- 1.3 Objective . . . 2
- 1.4 Previous work . . . 2
- 1.5 Contributions . . . 3
- 1.6 Roadmap . . . 3

2 Background **5**
- 2.1 An overview of ΔQSD . . . 5
  - 2.1.1 Outcome . . . 5
  - 2.1.2 Failure semantics . . . 6
  - 2.1.3 Quality attenuation (ΔQ) . . . 6
  - 2.1.4 Partial ordering . . . 7
  - 2.1.5 Timeliness . . . 7
  - 2.1.6 QTA, required ΔQ . . . 7
  - 2.1.7 Outcome diagram . . . 8
  - 2.1.8 Outcome diagrams refinement . . . 10
  - 2.1.9 Independence hypothesis . . . 11
- 2.2 Observability . . . 12
  - 2.2.1 erlang:trace . . . 13
  - 2.2.2 OpenTelemetry . . . 13
- 2.3 Current observability problems . . . 16
  - 2.3.1 Handling of long spans . . . 16

3 Design **17**
- 3.1 Measurement concepts . . . 18
  - 3.1.1 Probes . . . 18
  - 3.1.2 Extending failure . . . 18
  - 3.1.3 Time series of outcome instances . . . 19
- 3.2 Application side . . . 20
  - 3.2.1 System under test . . . 20
  - 3.2.2 ΔQ 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 . . . 22
3.4 Sliding execution windows . . . 23
3.4.1 Sampling window . . . 23
3.4.2 Polling window (Observing multiple ΔQs over a time interval) . . . 24
3.5 Triggers . . . 24
3.5.1 Snapshot . . . 24

4 Oscilloscope: User level concepts 26
4.1 ΔQSD concepts . . . 26
4.1.1 Representation of a ΔQ . . . 26
4.1.2 dMax = Δt · N . . . 27
4.1.3 QTA . . . 29
4.1.4 Confidence bounds . . . 29
4.2 ΔQ display . . . 30
4.3 Outcome diagram . . . 31
4.3.1 Causal link . . . 31
4.3.2 Sub-outcome diagrams . . . 31
4.3.3 Outcomes . . . 32
4.3.4 Operators . . . 32
4.3.5 Limitations . . . 32
4.3.6 Outcome diagram example . . . 33
4.4 Dashboard . . . 33
4.4.1 Sidebar . . . 33
4.4.2 Plots window . . . 35
4.5 Triggers . . . 35
4.5.1 Load . . . 35
4.5.2 QTA . . . 35

5 Oscilloscope: implementation 36
5.1 ΔQSD implementation . . . 36
5.1.1 Histogram representation of ΔQ . . . 36
5.1.2 dMax . . . 37
5.1.3 Rebinning . . . 37
5.1.4 Convolution . . . 38
5.1.5 Arithmetical operations . . . 39
5.1.6 Confidence bounds . . . 39
5.2 ΔQ Adapter . . . 40
5.2.1 API . . . 40
5.2.2 Handling outcome instances . . . 42
5.2.3 TCP connection . . . 42
5.3 Parser . . . 43
5.3.1 ANTLR . . . 43
5.3.2 Grammar . . . 44
5.4 Oscilloscope GUI . . . 44

6 Evaluation on synthetic programs **45**
6.1 System with sequential composition . . . 45
6.1.1 System's outcome diagram . . . 46
6.1.2 Determining parameters dynamically . . . 47
6.1.3 Observing non-linearity in the oscilloscope . . . 47
6.1.4 Detecting non-linearity with triggers . . . 51
6.2 Detecting slower workers in operators . . . 52
6.2.1 First-to-finish application . . . 52
6.2.2 All-to-finish application . . . 54

7 Performance evaluation of primitive operations **56**
7.1 Convolution performance . . . 56
7.2 GUI plotting performance . . . 57
7.3 ΔQ adapter performance . . . 58

8 Conclusions and future work **60**
8.1 Future improvements . . . 61
8.1.1 Oscilloscope improvements . . . 61
8.1.2 OpenTelemetry improvements . . . 62
8.1.3 Adapter improvement . . . 62
8.1.4 Real applications . . . 62
8.1.5 Licensing limitations . . . 62

Bibliography **63**

A Tests specifications **67**

I Addendum to Chapters 4 & 5 **68**

B Oscilloscope screenshots **69**
B.1 System/Handle plots tab . . . 69
B.2 Parameters tab . . . 70
B.3 Triggers tab . . . 71
B.4 Snapshot window . . . 72
B.5 Connection controls . . . 73
B.6 Widget view of GUI . . . 74

II User manual **75**

C How to download and launch **76**

D Oscilloscope: How to use **78**
D.1 Establishing the adapter - oscilloscope connection . . . 78
D.2 Sidebar: Outcome diagram and plots . . . 79
D.2.1 Creating the system/outcome diagrams . . . 79
D.2.2 Saving the system definition . . . 81

D.2.3 Loading the system definition . . . 81
D.2.4 Changing the sampling rate . . . 82
D.2.5 Managing the plots . . . 82
D.2.6 Removing a plot . . . 85
D.3 Sidebar: Probes settings . . . 85
D.3.1 Setting a QTA . . . 85
D.3.2 Setting the parameters of a probe . . . 86
D.4 Triggers . . . 88
D.5 Snapshots . . . 88
D.6 Instrumenting the Erlang application . . . 89
D.6.1 Including the adapter . . . 89
D.6.2 Starting spans . . . 90
D.6.3 With spans . . . 90
D.6.4 Ending spans . . . 90
D.6.5 Failing outcome instances . . . 90

III Source Code Appendix 91

E Grammar 92

F C++ Source Files 93
F.1 Root folder . . . 93
F.1.1 main.cpp . . . 93
F.1.2 Application.cpp . . . 94
F.2 Dashboard . . . 99
F.2.1 ColorRegistry.cpp . . . 99
F.2.2 CustomLegendEntry.cpp . . . 99
F.2.3 CustomLegendPanel.cpp . . . 100
F.2.4 DQPlotController.cpp . . . 101
F.2.5 DQPlotList.cpp . . . 111
F.2.6 DelaySettingsWidget.cpp . . . 113
F.2.7 DeltaQPlot.cpp . . . 115
F.2.8 MainWindow.cpp . . . 118
F.2.9 NewPlotList.cpp . . . 123
F.2.10 ObservableSettings.cpp . . . 125
F.2.11 SamplingRateWidget.cpp . . . 125
F.2.12 QTAInputWidget.cpp . . . 126
F.2.13 Sidebar.cpp . . . 129
F.2.14 SnapshotViewerWindow.cpp . . . 131
F.2.15 StubControlWidget.cpp . . . 136
F.2.16 SystemCreationWidget.cpp . . . 138
F.2.17 TriggersTab.cpp . . . 141
F.3 Outcome diagram . . . 146
F.3.1 Observable.cpp . . . 146
F.3.2 Operator.cpp . . . 149
F.3.3 Outcome.cpp . . . 151

F.3.4 Probe.cpp . . . 152
F.3.5 System.cpp . . . 153
F.4 ΔQ (maths) . . . 157
F.4.1 ConfidenceInterval.cpp . . . 157
F.4.2 DeltaQ.cpp . . . 159
F.4.3 DeltaQOperations.cpp . . . 165
F.4.4 Snapshot.cpp . . . 170
F.4.5 TriggerManager.cpp . . . 172
F.4.6 Triggers.cpp . . . 174
F.5 parser . . . 175
F.5.1 SystemBuilder.cpp . . . 175
F.5.2 SystemParserInterface.cpp . . . 180
F.6 server . . . 182
F.6.1 Server.cpp . . . 182

G C++ Header Files **191**
G.1 Root . . . 191
G.1.1 Application.h . . . 191
G.2 dashboard . . . 192
G.2.1 ColorRegistry.h . . . 192
G.2.2 CustomLegendEntry.h . . . 192
G.2.3 CustomLegendPanel.h . . . 193
G.2.4 DQPlotController.h . . . 194
G.2.5 DQPlotList.h . . . 196
G.2.6 DelaySettingsWidget.h . . . 198
G.2.7 DeltaQPlot.h . . . 199
G.2.8 MainWindow.h . . . 202
G.2.9 NewPlotList.h . . . 203
G.2.10 ObservableSettings.h . . . 204
G.2.11 SamplingRateWidget.h . . . 205
G.2.12 QTAInputWidget.h . . . 206
G.2.13 Sidebar.h . . . 208
G.2.14 SnapshotViewerWindow.h . . . 210
G.2.15 StubControlWidget.h . . . 211
G.2.16 SystemCreationWidget.h . . . 212
G.2.17 TriggersTab.h . . . 213
G.3 diagram . . . 215
G.3.1 Observable.h . . . 215
G.3.2 Operator.h . . . 218
G.3.3 OperatorType.h . . . 220
G.3.4 Outcome.h . . . 220
G.3.5 Probe.h . . . 220
G.3.6 Sample.h . . . 222
G.3.7 System.h . . . 222
G.4 ΔQ (maths) . . . 224
G.4.1 ConfidenceInterval.h . . . 224

G.4.2 DeltaQ.h . . . 225
G.4.3 DeltaQOperations.h . . . 227
G.4.4 DeltaQRepr.h . . . 228
G.4.5 QTA.h . . . 229
G.4.6 Snapshot.h . . . 230
G.4.7 TriggerManager.h . . . 232
G.4.8 TriggerTypes.h . . . 234
G.4.9 Triggers.h . . . 234
G.5 parser . . . 235
G.5.1 SystemBuilder.h . . . 235
G.5.2 SystemErrorListener.h . . . 236
G.5.3 SystemParserInterface.h . . . 237
G.6 server . . . 238
G.6.1 Server.h . . . 238

H Build Configuration Files **241**
H.1 src/CMakeLists.txt . . . 241
H.2 dashboard/CMakeLists.txt . . . 241
H.3 diagram/CMakeLists.txt . . . 242
H.4 maths/CMakeLists.txt . . . 243
H.5 parser/CMakeLists.txt . . . 244
H.6 server/CMakeLists.txt . . . 245

I Erlang Source Files **247**
I.1 Root . . . 247
I.1.1 dqsd_otel.erl . . . 247
I.1.2 dqsd_otel_app.erl . . . 251
I.1.3 dqsd_otel_sup.erl . . . 252
I.1.4 dqsd_otel_tcp_client.erl . . . 252
I.1.5 dqsd_otel_tcp_server.erl . . . 254

J Erlang Application Files **258**
J.1 Root . . . 258
J.1.1 dqsd_otel.app.src . . . 258

K Synthetic Applications **259**
K.1 Synthetic applications . . . 259
K.1.1 M/M/1/K queue application . . . 259
K.1.2 First to finish application . . . 262
K.1.3 All to finish application . . . 266

Chapter 1

Introduction

1.1 Problem

Diagnosing and measurement of distributed systems is not a trivial task. The very nature of them makes capturing the behaviour of the different parts that compose it, that may be in different physical locations, a challenging endeavour. [1]

In 2019, the merge of OpenCensus and OpenTracing led to the creation of OpenTelemetry [2]. It aims to provide a standard set of API to monitor running systems and avoid vendor lock-in [3]. As a result, multiple vendors' monitoring tools support OpenTelemetry metrics analysis [4]. They allow engineers to follow requests in system, studying their time of execution, the paths they take, the various events and much more [5] [6]. Nevertheless, open source non-commercial monitoring tools of large distributed system are few, and current solutions which may provide detailed insights about a system are commercial ones. [7]

Furthermore, while the tools may also provide insights about running systems and the execution times of various endpoints [8], they fail to capture essential requirements about timeliness. How can performance issues be recognised instantly? These issues may not be even apparent in these monitoring tools, as they can appear at the microseconds level. They may not be easily spotted by normal analysis of average delays.
This is where the ΔQSD paradigm comes in.

1.2 Approach

In the context of this thesis, the **ΔQSD paradigm** has been used to develop a tool to study the real-time behaviour of running systems.
As described by a tutorial given on ΔQSD [9]:

> "ΔQSD is an industrial-strength approach for large-scale system design that can predict performance and feasibility early on in the design process."

The paradigm has been developed over 30 years by the people around **Predictable Network Solutions Ltd** [10][11]. It has had various successful uses in the context

of distributed and large-scale projects. Moreover, it is the basis of Broadband forum's TR452 standard series, used in instrumenting data networks [12].

Thanks to outcome diagrams and statistical representations of component's behaviour, performance and feasibility can be predicted with the paradigm at high load, even if the system is not fully defined. [9] [11]

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 ΔQSD. This is where the **ΔQ oscilloscope** comes in.

1.3 Objective

This project will develop a practical tool, the **ΔQ 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 [13] [14]. The tool will provide useful information for these applications for understanding their behaviour, for diagnosing performance issues, and for optimising performance over their lifetime. [15]

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

The paradigm ideal target is "large distributed applications handling many independent tasks where performance and reliability are important" [9]. This also applies to the oscilloscope.

1.4 Previous work

The ΔQSD paradigm has been formalised across different papers [11] [16] and was brought to the attention of engineers via tutorials [9], and to students at Université Catholique de Louvain. [17]

A Jupyter notebook workbench has been made available on GitHub [18]. It is meant as an interactive tool to show how the ΔQSD paradigm can be applied to real life examples. It shows real time ΔQ graphs for typical outcome diagrams, but is not adequate to be scaled to real time systems.

Observability tools such as Erlang tracing [19] and OpenTelemetry [20] lack the notion 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.

1.5 Contributions

We make the following principal contributions in the master thesis:

- The ΔQ oscilloscope, from design to implementation, to plot real-time ΔQ graphs. The oscilloscope contributions include:
  - A graphical interface in Qt.
  - The underlying implementation of ΔQSD concepts, including a textual syntax to create outcome diagrams derived from the original algebraic syntax.
  - Efficient convolution algorithms.
  - A system of triggers to catch rare events, when system behaviour fails to meet quality requirements.
- The ΔQ adapter to communicate from the Erlang application to the oscilloscope.
- The evaluation of the effectiveness of the oscilloscope on synthetic applications.
- The evaluation of the efficiency of the basic operations regarding the oscilloscope: convolution, graphing and the adapter overhead.

These contributions can show that the ΔQSD paradigm can be translated from a posteriori analysis to real-time observation of running system. Furthermore, it reinforces the validity of the paradigm.

1.6 Roadmap

This thesis gives the reader everything that is needed to use the oscilloscope and exploit it to its full potential.

We divided the thesis in multiple chapters:

- Chapter 2 gives the reader a background of the theoretical foundations of ΔQSD, which are the basis of the oscilloscope and are fundamental to understand what is shown in the oscilloscope. Secondly, an introduction to OpenTelemetry, the framework our Erlang adapter is built on top of. Lastly, we provide what we believe are the current limitations of the observability framework and how we plan to tackle them.
- Chapter 3 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 into how the different parts of our design interact together and how to correctly apply the concepts we introduced. First, we present the application side, where the Erlang system to be observed is and where the ΔQ adapter will be. Second, the oscilloscope side, where the ΔQ oscilloscope will display graphs about the running system. Lastly, after having introduced the oscilloscope, we explain abstract concepts implemented in it, like sliding windows and triggers.

- Chapter 4 & 5 present the oscilloscope. First providing "user level concepts" of how ΔQSD is used and what the user should expect visually from the dashboard. Chapter 4 also provides a complete explanation of how to write outcome diagrams and what the different sections on the dashboard do. Secondly, in Chapter 5, we give a more low level explanation, which goes into more technical details of the parts that compose the oscilloscope. Namely, we provide mathematical explanations of ΔQSD concepts explained in the previous chapter.
- Chapter 6 provides synthetic applications which have been tested with the oscilloscope that demonstrate the usefulness of the oscilloscope in a distributed setting. In Chapter 7 we perform evaluations of the performance of the different parts we have developed to understand the overheads that are present.

Chapter 8 provides future possibilities which can be explored to improve the application. In the appendix, we provide screenshots of the application. We also 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 background of the concepts key to understanding the ΔQSD paradigm.

Secondly, we explain 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 ΔQSD

ΔQSD is defined as [11]:

> "A metrics-based, quality-centric paradigm that uses formalised outcome diagrams to explore the performance consequences of design decisions".

The key concepts that give the paradigm its name are **quality attenuation (ΔQ)** and **outcome diagrams**. [9]

The dependency and causality properties of a system can be captured by outcome diagrams, while the probability distribution representation (ΔQ) can precisely model a system's behaviour. [11]

The following sections are a summary of multiple articles and presentations formalising the paradigm. Some of the graphs we present have too been adapted from the articles.

2.1.1 Outcome

To build outcome diagrams, we need to first introduce outcomes.

Outcomes represent system behaviours or tasks that "can start at some point in time and ***may*** be observed to complete at some later time" [12]. The result produced by performing a system's task is mapped to an outcome. [11]

The particularity of outcomes is that they can represent multiple levels of granularity. Suppose an outcome is 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. [11]
Even though these outcomes are defined as "black boxes", they still have timeliness constraints like any other outcome.

Let us define some important concepts about outcomes.

Observables Key to outcomes is the notion of event.
An outcome has observable starting events and ending events [9]. As the events may occur in different locations in a distributed setting, we say the outcome has a starting set and ending set of events. They are called "**observables**". [11]
We stated previously that an outcome **may** be observed to complete, this is because a starting event has no guarantee that it will be ended. An outcome can be said "done" if an end event occurs for a start event.

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}. [16]

Graphical representation Outcomes are represented as circles, with the starting and terminating set of events being represented by boxes. [11]

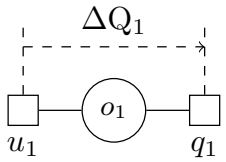

Figure 2.1: The outcome (circle) and the starting set (left) and terminating set (right) of events.

2.1.2 Failure semantics

Failure indicates "an input message m_{in} **that has no output message** m_{out}" [16]. It models the probability that the delay is infinite.

2.1.3 Quality attenuation (ΔQ)

ΔQ is defined as "a cumulative distribution function (CDF) that defines both *latency* and *failure probability* between a start and end event". [9]

As multiple instances of an outcome are executed, multiple delays can be observed for the executions. The observed delays can be represented as a CDF, we call it ΔQ_{obs}. In the ΔQ's CDF, $\mathbf{\Delta Q(x)}$ is the probability that an outcome O occurs in time $t \leq x$ [16]. We may sometimes use the derivative of a ΔQ, which is the probability density function (PDF). We show a typical CDF of a ΔQ in Fig. 2.2.

The key feature that makes ΔQ stand out is the notion of failure incorporated in the representation of an outcome's behaviour.
Ideally, a system would execute without errors, failure or delay. This is never the case. Since the ideal cannot be attained, the quality of the system's responses are "*attenuated relative to the ideal*" [11]. This quality attenuation gives the name to ΔQ.

Since multiple factors can influence the delay of a response (geographical, physical) [9], ΔQ can be firstly modeled as a random variable. Nevertheless, as it incorporates failures, which are discrete variables, and delays, which are continuous random variables, the authors describe it as an *Improper Random Variable*, as the probability of a finite or bounded delay is < 1. [11]

Intangible mass Depicted in Fig. 2.2, the ***intangible mass*** $1 - \lim_{x\to\infty} \Delta Q(x)$ of a ΔQ encodes the probability of failure/timeout/exception occurring. [16]

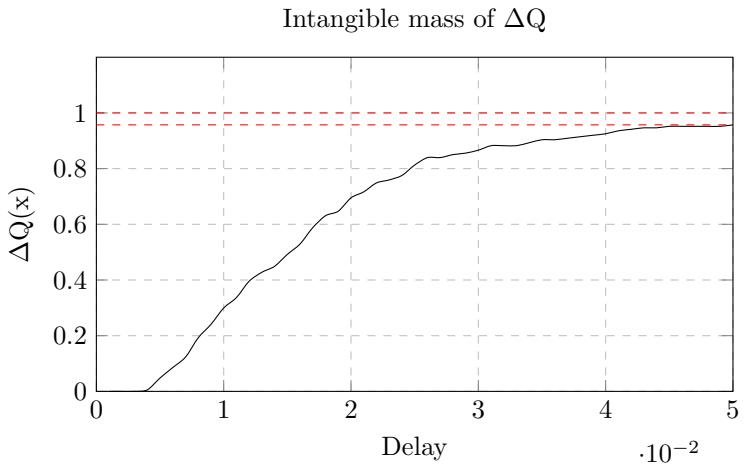

Figure 2.2: ΔQ CDF representation with intangible mass (red, dotted). The failure rate is about 5%

2.1.4 Partial ordering

We say ΔQ_1 is *less than* another ΔQ_2 if its CDF is everywhere to the left and above the other's CDF. This concept is represented mathematically as a partial order. [9]

2.1.5 Timeliness

Timeliness is "delivering results within required time bounds (sufficiently often)". [9]

2.1.6 QTA, required ΔQ

The Quantitative Timeliness Agreement (QTA) specifies the precise timeliness requirements (ΔQ_{req}) of a ΔQ_{obs}. [12] [16]

Leveraging the definition of partial ordering and timeliness, we can say that a system *satisfies timeliness* if $\Delta Q_{obs} \leq \Delta Q_{req}$. [16]

Slack and hazard There is performance *slack* when "a ΔQ is strictly less than the requirement." ($\Delta Q_{obs} < \Delta Q_{req}$).
There is performance *hazard* when "an observed ΔQ_{obs} intersects or is strictly greater than the required ΔQ_{req}" ($\Delta Q_{obs} \not< \Delta Q_{req}$). [11]

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.
We present in Fig. 2.3 an example of systems showing slack and hazard.

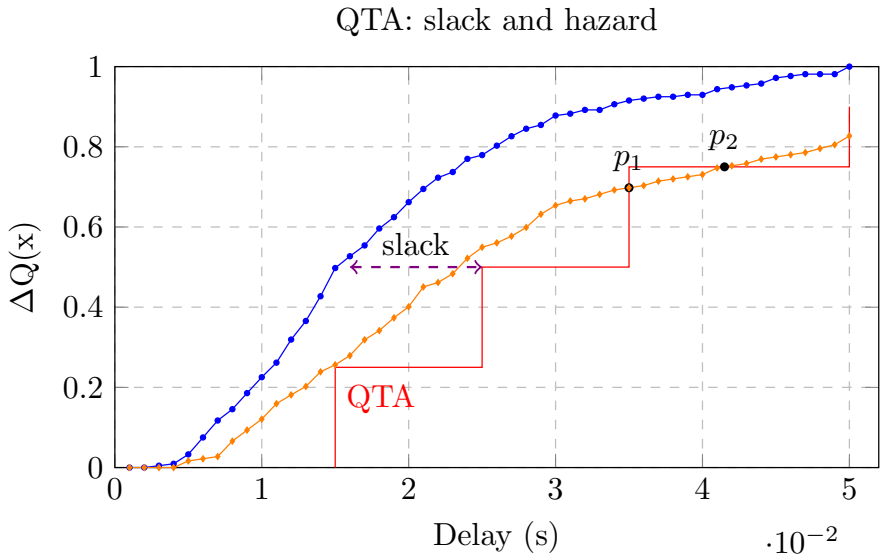

Figure 2.3: The system in blue (circle) is showing slack and satisfies the requirement. The system in orange (diamond) is showing signs that it cannot handle the stress, it is not respecting the system requirements imposed by the QTA. It intersects with the QTA in p_1, p_2, so it is not respecting the timeliness requirements. There is a $p_2 - p_1 > 0$ probability that the requirement is not satisfied.

2.1.7 Outcome diagram

An outcome diagram is central to capture the causal relationships between the outcomes [11]. It shows the causal connections between all the outcomes we are interested in. Given an outcome diagram, one can compute the ΔQ for the whole system. [9]
There are four different operators that represent the relationships between outcomes. [17]

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, then we say the two outcomes are sequentially composed. The total delay ΔQ_{AB} is given by the convolution of the PDFs of O_A and O_B ($O_A \circledast O_B$).

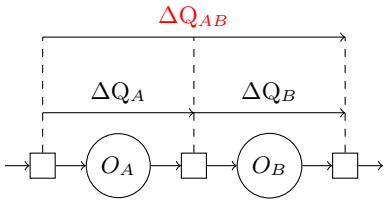

Figure 2.4: Sequential composition of O_A and O_B.

Where convolution ($\circledast$) between two PDF is:

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

Thus, ΔQ_{AB}:

$$\Delta Q_{AB} = \int_0^t PDF_A \circledast PDF_B\ d\delta \tag{2.2}$$

Convolution is the only operation which is based on the PDFs, the following operations are based on the CDF of the ΔQs (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:

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

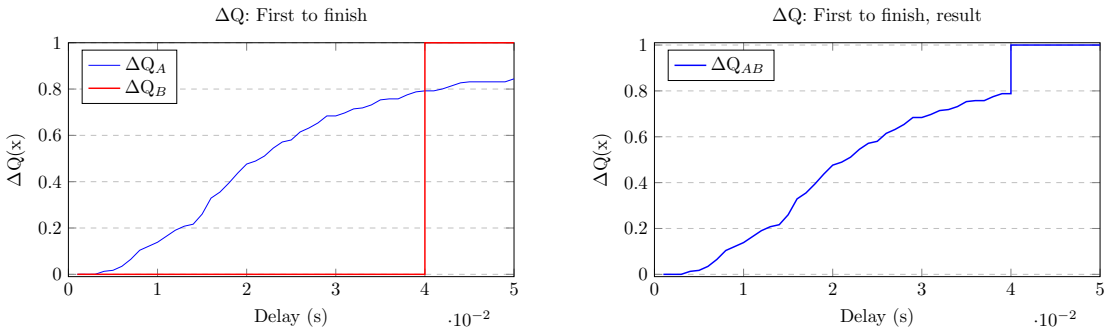

Figure 2.5: Left: $\Delta Q_{(A,B)}$. Right: $\text{FTF}_{(A,B)} = \Delta Q_{AB}$

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:

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

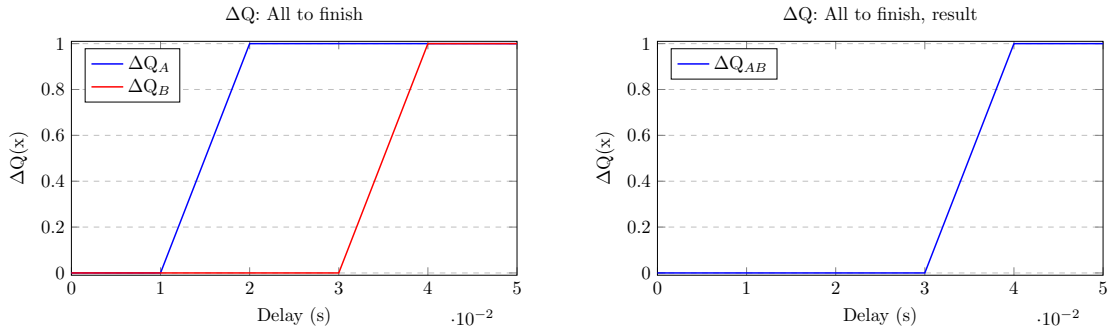

Figure 2.6: Left: $\Delta Q_{(A,B)}$. Right: $\text{ATF}_{(A,B)} = \Delta Q_{AB}$

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

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

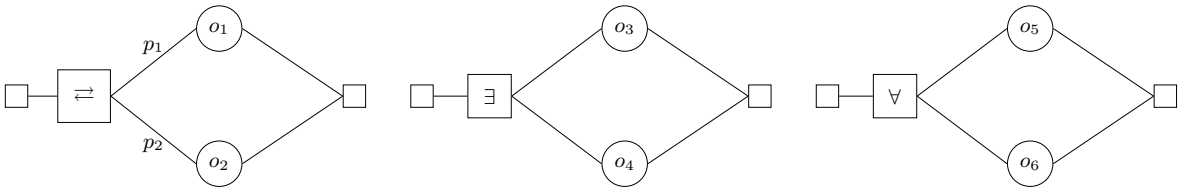

Figure 2.7: Graphical representation of the possible operators in an outcome diagram. From left to right: Probabilistic choice, first-to-finish, all-to-finish. [11]

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

These operators can be assembled together to create an outcome diagram. In Chapter 4 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. [11]
An outcome diagram can be "unboxed" by refining the outcomes that compose it. We can adapt a situation described by the previously cited article, called "Mind your Outcomes", to understand how refinement can allow the user to have a very precise representation of a system.

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 (Fig. 2.8).

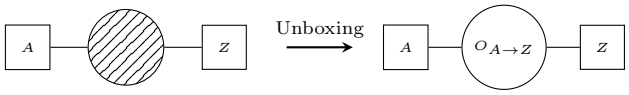

Figure 2.8: Refinement from black box to named outcome.

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 (Fig. 2.9).

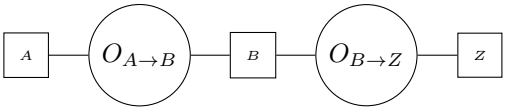

Figure 2.9: Further refinement from one task to two tasks.

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 (Fig. 2.10).

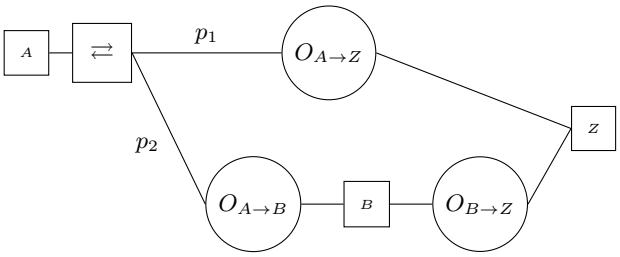

Figure 2.10: Refinement as probabilistic choice of executing either one or two tasks.

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 [15]. Let us explain the following assumption clearly.

Assume the situation described by Fig. 2.11, 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).

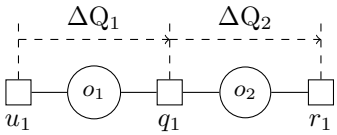

Figure 2.11: Two sequentially composed outcomes, o_1, o_2. The end event of o_1 is the start event of o_2.

At low load, the two components behaviour will be independent (Fig. 2.12), the system will behave **linearly**. According to the superposition principle, the overall delay will be the sum of the two delays, as will the overall processor utilisation. [21]

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 ($o_1 \circledast o_2$). This situation is observed in Fig. 2.12.

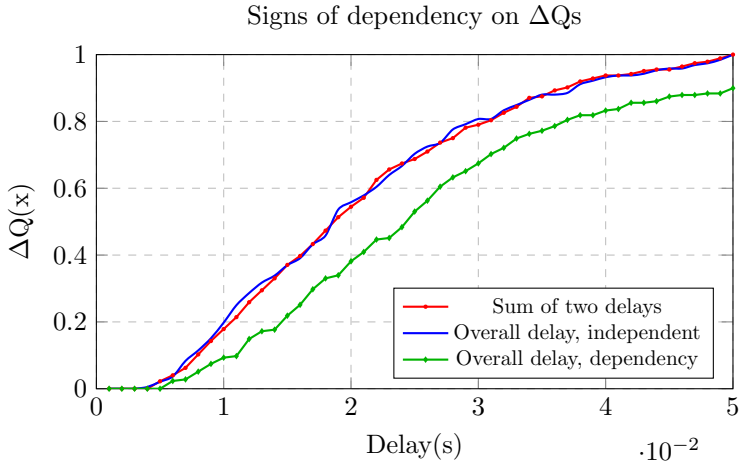

Figure 2.12: When the components are independent, the sum of the two delays (blue) and the overall delay (red) can be superposed. As o_1 and o_2 show signs of dependency, the overall delay (green) can be seen deviating from the sum of the two delays. There is **non-linearity**.

When the system is far from being overloaded, the effect is noticeable. As the cliff edge of overload is approached, the non-linearity will increase [9] [15]. These theoretical results can be observed in the oscilloscope. We will explore such cases in Chapter 6.

2.2 Observability

OpenTelemetry refers to observability as [22]:

> "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."

In the case of the Erlang programming language, we describe respectively two different ways to observe a running Erlang system: erlang:trace and OpenTelemetry.

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 interface `erlang:trace`. According to the documentation: "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" [23]. One can observe function calls, messages being sent and received, process being spawned, garbage collecting and more.

```
-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 2.13: erlang:trace/3 specification. [23]

Nevertheless, in Erlang trace 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

According to OpenTelemetry's website [22]: 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.

OpenTelemetry is available for a plethora of languages [24], including Erlang. As of writing this, logs and metrics are unstable in Erlang. [25]

The Erlang Ecosystem Foundation has a working group focused on evolving the tools related to observability, including OpenTelemetry and the runtime observability monitoring tools. [26]

Traces

Traces are why we are basing our program on top of OpenTelemetry, traces follow the whole path of a request in an application, and they are comprised of one or more spans [6]. Traces can propagate to multiple services and record multiple paths in different microservices [27].

Span A span is a unit of work or operation. Multiple spans can be assembled into a trace and can be causally linked (Fig. 2.15). The spans can have a hierarchy, where *root spans* represent a request from start to finish and a child span the requests that are completed inside the root span [27]. 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 [6]. This relation can be represented in the oscilloscope via **probes**, we will present how spans relate to probes in Chapter 3.

```
{
  "name": "oscilloscope-span",
  "context": {
    "trace_id": "5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "5fb397be34d26b51"
  },
  "parent_id": "051581bf3cb55c13",
  "start_time": "2022-04-29T18:52:58.114304Z",
  "end_time": "2022-04-29T22:52:58.114561Z",
  "attributes": {
    "http.route": "some_route"
  },
}
```

Figure 2.14: Example of span from the OpenTelemetry website [6]. The span has a parent, indicating that child and parent spans are related and are both part of the same trace.

Monitoring OpenTelemetry spans

In OpenTelemetry, the user can export their traces export to backends and monitoring such as Jaeger (Fig. 2.15), Zipkin, Datadog [28]. There, a user can analyse the traces to troubleshoot their programs by observing the flow of the requests [29]. These monitoring tools give extensive details about a running system, but may fail to capture essential timeliness 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 [30] might have, where you can observe cloud instances, instances cost, dependency graphs. But the oscilloscope can nevertheless provide precise insights about dependency, overload and much more, thanks to the ΔQSD paradigm.

This is also the reason why the adapter includes OpenTelemetry macros. The oscilloscope can be put next to a monitoring tool where one exports spans to, so that an engineer might consult the monitoring tool to get the global picture of a running app, and the oscilloscope to provide precise insights to understand the system's behaviour.

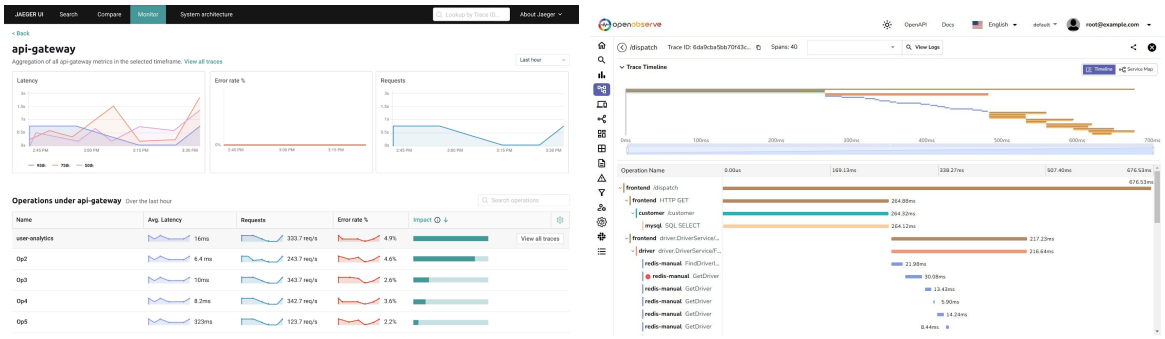

Figure 2.15: **Left**: Jaeger interface [31]. **Right**: Analysis of a span on OpenObserve. [32]

Span macros

OpenTelemetry provides macros to start, end and interact with spans in Erlang. The following code excerpts are taken from the OpenTelemetry instrumentation wiki. [25]

?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. [33]

```
parent_function() ->
    ?with_span(parent, #{}, fun child_function/0).
child_function() ->
%% this is the same process, 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() ->
     %% 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),
                        ?end_span(SpanCtx)
                    end),
```

?end_span `?end_span` ends a span started with `?start_span`.

2.3 Current observability problems

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. The engineer knows it should take at most 1 second. In the 10 seconds, no information about the span appears on the dashboard.

We believe that this is a problem. One would like to know right away if something is wrong with their application. This is where the ΔQSD paradigm and the ΔQ oscilloscope come in handy.
Using ΔQSD, we can set a maximum delay ($dMax$) for a task. As soon as the maximum delay is hit, the oscilloscope is notified right away that there is a problem.

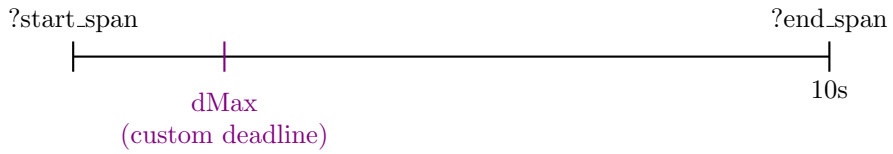

Figure 2.16: Execution of a span in OpenTelemetry. Normally, the user will be notified after 10 seconds that the function has ended. The $dMax$ deadline gives an early notification that the span has taken too long.

2.3.1 Handling of long spans

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

An article has already tackled this problem [34]. In the article it is stated that OpenTelemetry limits the length of its spans, moreover, those which are not terminated are lost and not exported.
In addition, it is said that if the span is the parent/root span, its effect could trickle down to child spans. We can quickly see how this can become problematic, all the information about an execution of your task is lost. A span can apparently not be terminated for trivial reasons: refreshing a tab, network failures, crashes. The author states that there are a few solutions 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. The solutions have been implemented by the Embrace (commercial) monitoring tool [35], but it is apparently the only tool out of the ecosystem of monitoring tools.

We believe that the adapter we provide can be a great start to improve observability requirements surrounding OpenTelemetry. Data about spans will always be carried to the oscilloscope, whether the span is long or non-terminated. This is further developed in Chapters 3 & 5.

Chapter 3

Design

This chapter aims to first extend the concepts of ΔQSD, giving more insights into how the different parts of the system need to be instrumented to correctly work together.

- We first provide concepts of probes, we extend the ΔQSD 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 explain the global design of the system (see Fig. 3.1) in two parts:
  Firstly, we explain the application side, where the Erlang running system is. Consequently, it's where the ΔQ adapter interface will be. It performs the translation of spans to outcome instances thanks to the inserted probes.
  Secondly, the oscilloscope side. There, the server receives information about outcome instances from the adapter. The ΔQ oscilloscope can then plot ΔQ graphs from the received instances. In the oscilloscope one can define outcome diagrams, set parameters for probes and control the adapter.
- Lastly, we provide high level concepts of execution windows, triggers and snapshot. These are the key design elements of the oscilloscope.

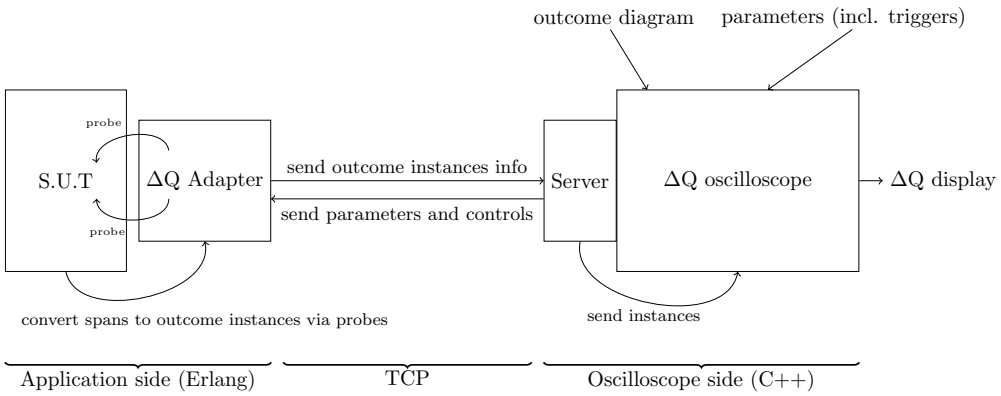

Figure 3.1: Global system design diagram. The two sides communicate via TCP sockets to share information about outcome instances and probe parameters.

3.1 Measurement concepts

3.1.1 Probes

A system instrumented with OpenTelemetry has spans and traces to observe the execution of an operation [6]. 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. **Note** that a definition of probes has already been introduced in a previous article related to ΔQSD [12], but the concept we present here is not the same.

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 [15].

Consider Fig. 3.2 below. A probe is attached at every component (for example, a database [9]) to measure their ΔQs (p_2, p_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 ΔQ *"observed at p_1"*, which is the ΔQ which was calculated from the data received by inserting probe p_1. The ΔQ *"calculated at p_1"* is the resulting ΔQ from the convolution of the observed ΔQs 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. p_2 and p_3 are child spans which represent single tasks.

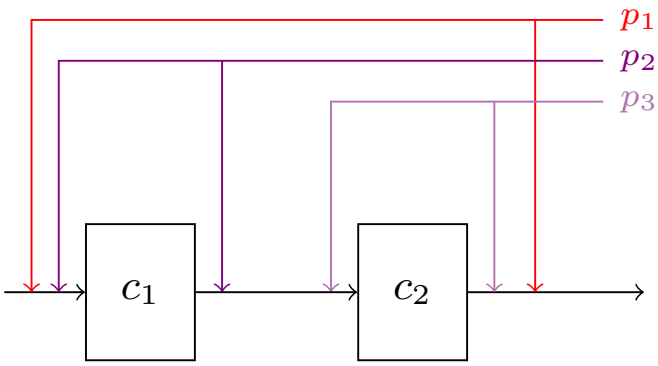

Figure 3.2: Probes inserted in a component diagram. In an application instrumented with OpenTelemetry, p_1 could be considered the root span, c_1 and c_2 its children spans sharing a causal link.

3.1.2 Extending failure

Recall the definition of failure (Section 2.1.2): *"an input message m_{in} that has no output message m_{out}"*. We also introduced the notion of a maximum delay $dMax$ (Fig. 2.16). The notion of failure is extended to the following definition:

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

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 [11]. The adapter does not interfere with OpenTelemetry, 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.

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 name p.
- 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 three possible 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 is equal to $t_s +$ 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. The collected elapsed times of execution (delays) from the outcome instances can be represented as a CDF, which is a ΔQ.

What can be considered a failed execution? 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.

Imagine a queue with a buffer: the buffer queue being full and dropping incoming messages can be modeled as a failure.

On another note, the way of handling "errored" spans in OpenTelemetry can differ from user to user [36], so the adapter will not handle ending and setting statuses for "failed" spans.

In any case, **timed out and failed will both be considered as a failure** in a ΔQ. The distinction in an outcome instance is there for future refinements of the oscilloscope, where more statistics can be displayed about a ΔQ.

3.2 Application side

3.2.1 System under test

The system under test **(S.U.T)** is the Erlang system the engineer wishes to observe (Fig. 3.1). 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 [9].

3.2.2 ΔQ Adapter

The ΔQ adapter is the `dqsd_otel` Erlang interface [37]. 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 after 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 sockets.

The adapter can receive messages from the oscilloscope, the messages are about updating a 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 [6]. 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. The name given with the adapter functions `"start_span/with_span"` is the name of the probe (Fig. 3.3). The PID which is returned by starting a span must be carried throughout the whole execution, and used when ending spans to create the correlation between a probe and an outcome instance.

```
% 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">>),

% Start and fail span directly
{WorkerCtx, WorkerPid} = dqsd_otel:start_span(<<"worker_1">>),
dqsd_otel:fail_span(WorkerPid),
%Here, you would need to end the span manually with ?end_span

%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 time
↪  from start_span to end_span > dMax), 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),
```

Figure 3.3: Example usage of the adapter

Further details about the implementation of the adapter are explained in Chapter 5. A user guide on how to include the adapter in a project and how to instrument a program is found in the appendix (Section D.6).

3.3 Oscilloscope side

3.3.1 Server

The server is a simple TCP server in C++. It 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 ΔQs of probes inserted in the system under test [38]. 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 allows the user to set parameters 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 observables and to the sub-outcome diagrams observables (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 outcomes, 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 received by the adapter to the probes' time series.

In the system below (Fig. 3.4), 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.

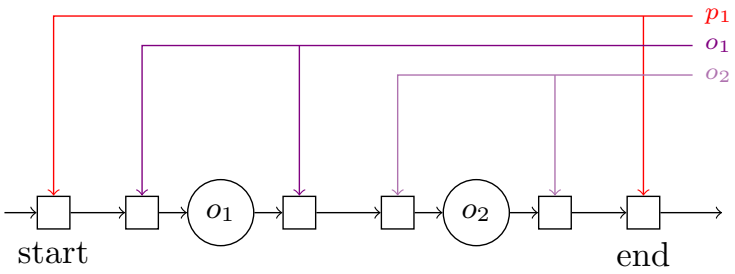

Figure 3.4: Probes inserted in the outcome diagram of the previous component diagram in Fig. 3.2.

The **observables** are an abstract representation of events. Consider the previous code snippet Fig. 3.3: the *start* event of "probe" and worker_1's start event are subsequent instructions. The probe's start event is practically the same as worker_1's start event, indeed, they could be overlapped in the graph above (Fig. 3.4). We nevertheless show the distinction to show that probe and worker_1 need to be started differently in Erlang as the information they carry is about two distinct instances. It is the same concept as starting a parent/child span. Furthermore, this difference is remarked in the definition of outcome diagrams for the oscilloscope, for which we provide a syntax in the Chapter 4.

As for operators, probes are automatically attached to the components inside them and to the operators' observables (Fig. 3.5).

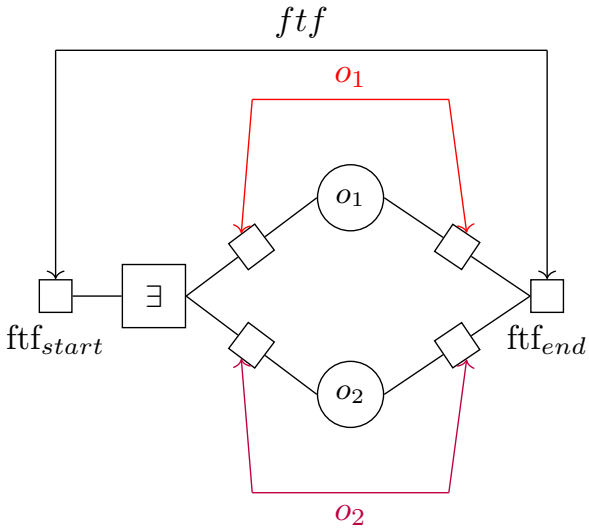

Figure 3.5: Probes inserted into a first-to-finish operator.

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 Sliding execution windows

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

3.4.1 Sampling window

Suppose we are at time t, the observed ΔQ of a sampling window at time t we display is the ΔQ obtained from the outcome instances which ended within a **window of time** $(t-1)_l$ **-** $(t-1)_u$, with $(t-1)_u$ equal to $t-x$, and x the sampling rate (see Fig. 3.6). The sampling rate x is how often ΔQs are calculated.
This is to account for various overheads that need to be taken into consideration. They could be network overheads, the adapter overhead, C++ latency and more. 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.

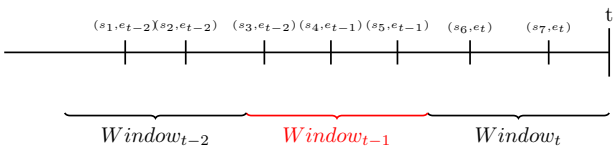

Figure 3.6: The window_{t-1} is the current window being displayed at time t.

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

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

The ΔQs which are observed and calculated in a sampling window are not precise, this is why we need to introduce the polling window.

3.4.2 Polling window (Observing multiple ΔQs over a time interval)

The polling window is the window of past ΔQs which are stored to keep a snapshot of the system over time and over which confidence bounds are calculated. The polling window serves to improve the precision of the measurement of the confidence bounds for ΔQs.

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

The current limit of ΔQs for a polling window is 30 ΔQs. At $t = 31$, the older ΔQs will be removed from the polling window and in turn from the confidence bounds. Newer sampled ΔQs will be added, keeping the limit of ΔQs in a polling window to 30. A larger polling window could be used to increase precision of ΔQ computation, i.e. to reduce the width of the confidence interval. This is nevertheless only valid for stationary systems where the ΔQ does not vary over time.

3.5 Triggers

Much like an oscilloscope that has a trigger mechanism to capture periodic signals or investigate a transient event [39], 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 of a sampling window 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.

3.5.1 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.

We can define a **"snapshot window"** which is equal to the polling window when no triggers are fired. The maximum size of the snapshot window under normal conditions (no triggers fired) is 30 ΔQs, like the polling window. All the ΔQs which are observed and calculated in a polling window are stored away in both the snapshot and polling windows. Then, if no trigger is fired, older ΔQs are removed from both.

Instead, if a trigger is fired, the ΔQs sampled for the next 5 seconds are added to the snapshot window, without removing older ones from the window. This it to allow the user to look at the state of the system before and after the trigger. After these 5

seconds the recorded snapshot window will contain the 30 ΔQs before the trigger being fired, which is equal to the polling window before the trigger, plus the ΔQs sampled in the 5 seconds after the trigger being fired. This snapshot window is stored away and can be observed by the user.

The polling window will always keep 30 ΔQs, it still follows the same strategy of removing older ΔQs, even if a trigger was fired. Meanwhile, the snapshot window's maximum size increases during the recording of one, keeping older ΔQs even if the size of the snapshot window is > 30. When the snapshot finishes recording at a time t, the snapshot window will be the same as the polling window at time t, they will contain the same ΔQs.

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 the concepts related ΔQSD are implemented in the oscilloscope, the parameters that define a probe's ΔQ, its representation. We show how probe's ΔQ(s) will be shown in the oscilloscope.
- We then provide a language to write outcome diagrams based on an already existing syntax and provide an example.
- Lastly, we explain the different widgets on the oscilloscope dashboard and the available triggers.

4.1 ΔQSD concepts

We provide in this section the concepts needed to understand what is displayed on the oscilloscope.

4.1.1 Representation of a ΔQ

We provide a class to calculate the ΔQ of a probe in a sampling window 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 whose end time is between t_l and t_u, calculating the PDF of the delays of the outcome instances, and calculating the resulting CDF based on the PDF. This is called the **Observed ΔQ**.

Calculated ΔQ A ΔQ can also be calculated by performing operations on two or more observed ΔQs (convolution, operators operations). The notion of outcome instances is lost between calculations, as the interest shifts towards calculating the

resulting PDFs and CDFs. 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 Fig. 3.4, the probes o_1 and o_2 observe simple outcomes, they can only display the observed ΔQs of o_1, o_2. 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 ΔQs of o_1, o_2

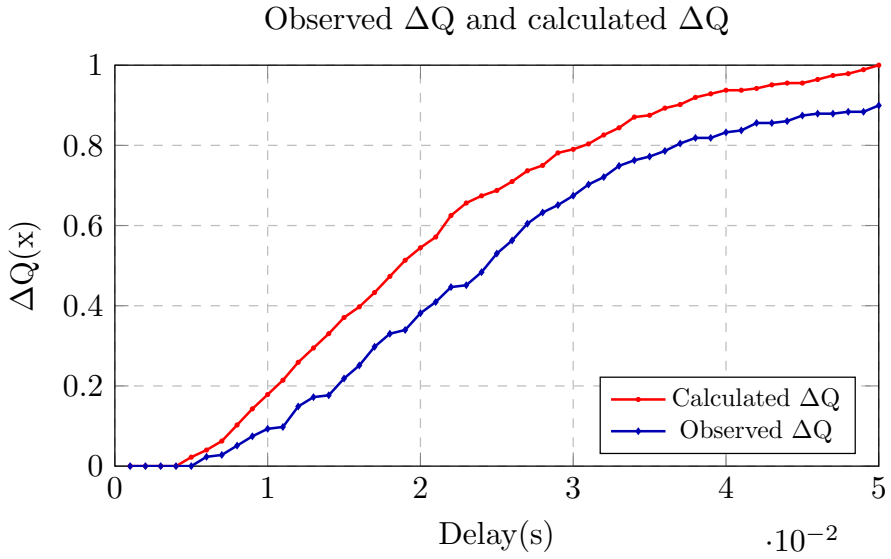

Figure 4.1: (Red, circle, above): Sampling window **Calculated ΔQ**. (Blue, diamond, below): Sampling window **Observed ΔQ**

4.1.2 dMax = Δt · 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

Let us explain the following equation:

$$dMax = \Delta t \cdot N \tag{4.1}$$

- $dMax$: 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.

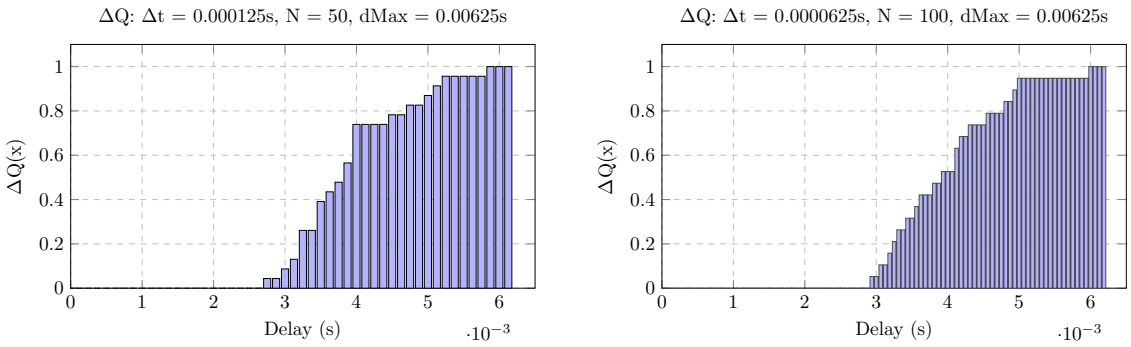

Figure 4.2: **Left**: Sample ΔQ representation as a histogram with higher resolution but lower precision.
Right: Sample ΔQ representation as a histogram with lower resolution but higher precision.
Both ΔQs have the same $dMax$, but the amount of precise information they provide is far different.

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 ΔQs.

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.

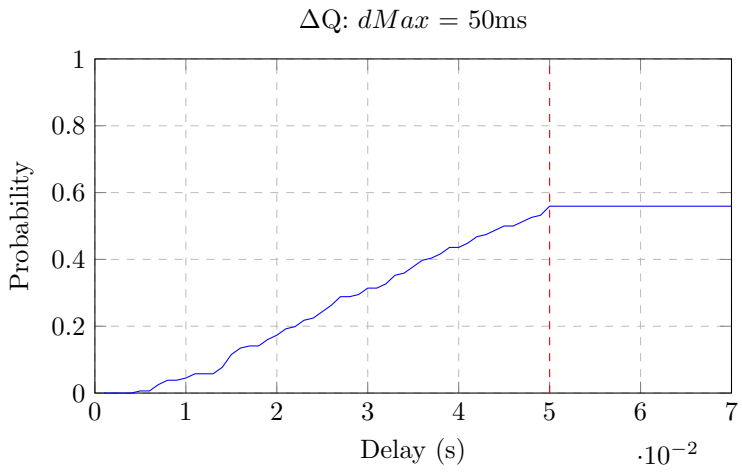

Figure 4.3: ΔQ: $dMax$ = 50ms, the ΔQ will stay constant when delay > $dMax$.

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 minimum percentage p of successful instances accepted for an observable ($0.75 < p < 1$). The QTA is comparable to the one we have shown in Fig. 2.3. The user can set the QTAs they want for the probes they have inserted, but the delays at the percentiles must be $< dMax$.

4.1.4 Confidence bounds

To observe the stationarity of a probe we must observe its ΔQs over a polling window and calculate confidence bounds over said ΔQs. A single ΔQ may fluctuate. This is why we include the mean and confidence bounds of ΔQs in the plot, which give a probability range over which the true CDF of the ΔQ should fall. [40]

Confidence bounds are given for *observed* and *calculated* ΔQs in a polling window (Fig. 4.4).
We first calculate a mean of the ΔQs 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.

The bounds are updated dynamically by inserting or removing a ΔQ to the current polling window. Every time a new ΔQ is sampled, the oldest ΔQ in a window is removed if #ΔQs(polling window) > limit. The new ΔQ is added to the calculation of the mean and confidence bounds as it is sampled.

This allows us to consider a small window of execution rather than observing the execution since the start of the system for the bounds. This can help in observing stationarity of the system, where a fewer number of past ΔQs can help observe short term behaviour.

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

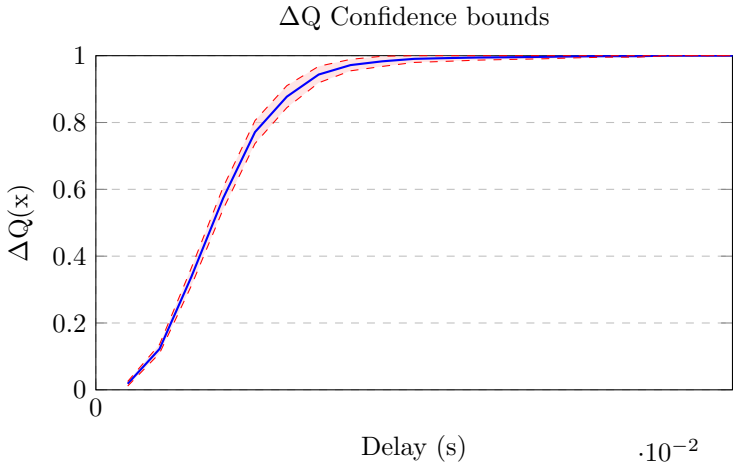

Figure 4.4: Upper and lower bounds (dashed, red) of the mean (blue) of multiple ΔQs. Confidence bounds can help recognise non-linearity. If the confidence bounds of calculated ΔQs are outside the observed ΔQs bounds, we know that the difference between the two is not just a measurement fluctuation, there is instead divergence from linear behaviour.

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 observed ΔQ of the sampling window, with the mean and confidence bounds calculated over the polling window of observed ΔQs.
- If applicable, the calculated ΔQ of the sampling window from the causally linked components observed in a probe, with the mean and confidence bounds calculated over the polling window of calculated ΔQs.
- Its QTA (if defined).

This allows for the user to observe if a ΔQ has deviated from normal execution, analyse its stationarity, non-linearity and observe a sampled ΔQ over a polling window.

In the screenshot below (Fig. 4.5) we can observe the multiple elements as they are displayed in real time in the oscilloscope.

- (1, green): The mean of the polling window observed ΔQs (yellow) with the confidence bounds. Upper bound (dark green) and lower bound (light green). The observed ΔQ of the sampling window (blue) can be observed going out of the confidence bounds at delay 0.00125 s. The ΔQ in a sampling window is less precise than the mean and confidence bounds calculated in the polling window.
- (2, red): The mean of the calculated ΔQs (ochre) with the confidence bounds of the mean. Upper bound (purple) and lower bound (magenta). The calculated ΔQ of the sampling window (red) is inside its confidence bounds.
- (3, blue): The **QTA**.

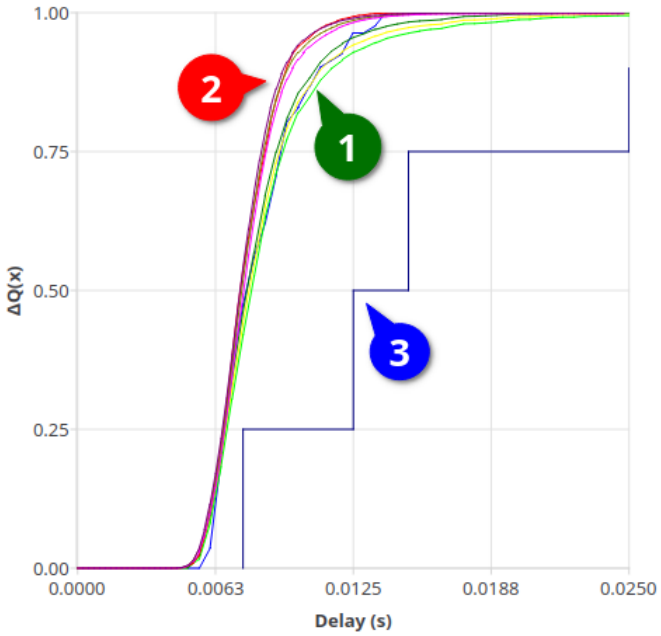

Figure 4.5: ΔQs, confidence bounds, means and QTA for a probe observing the causal link of multiple components.

4.3 Outcome diagram

An abstract syntax for constructing outcome diagrams has already been defined in a previous paper [16]. Nevertheless, the oscilloscope needs a textual way to define an outcome diagram.
We define a grammar to create an outcome diagram in our oscilloscope, which 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. They can then be linked together to form the global system outcome diagram. Sub-outcome diagrams can observe one or multiple components. Recall Fig. 3.4, 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 start and end events 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 dtheir efined 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.

4.3.6 Outcome diagram example

We provide a sample example of an outcome diagram definition. We also provide its resulting outcome diagram with probes inserted.

```
two_hops = o2 -> o3;
total = p:pc[0.9, 0.1](o1, s:two_hops);
```

Figure 4.6: Sample textual definition of an outcome diagram.

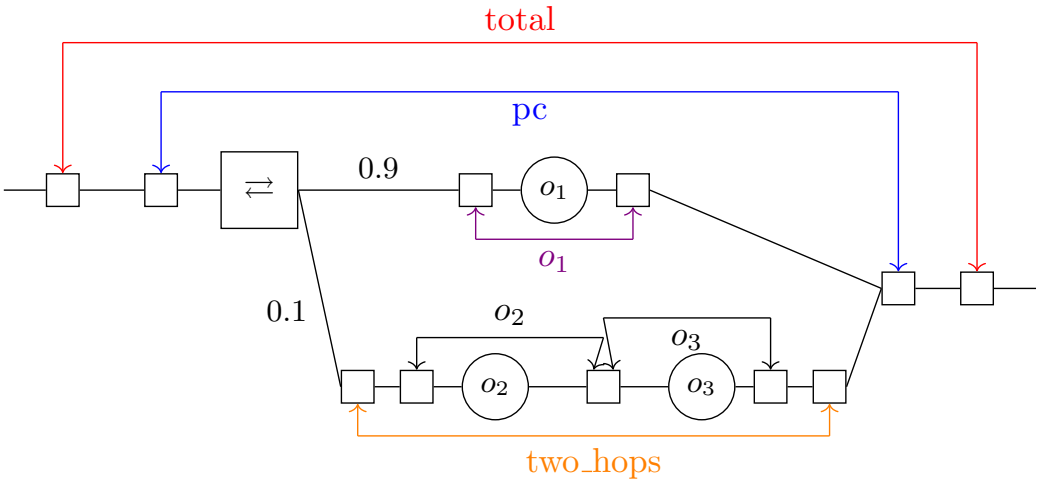

Figure 4.7: Resulting outcome diagram for definition Fig. 4.6.

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

In this tab, one can create the outcome diagram, add plots and modify current plots. A screenshot of the tab can be found in Section B.1.

System creation In this widget the user can create its system with the outcome diagram grammar. They can save the outcome diagram text or load it. The outcome diagram definition 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 system is wrong, they 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 they want 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 ΔQs are sampled 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. A screenshot of this tab can be found in Section B.2.

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 minimum amount of successful instances accepted. They can change this dynamically during execution. If a new $dMax$, which is smaller than the defined delays which were set in the QTA, is defined by the user, the QTA is reset.

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 probe.

Triggers tab

In the triggers tab the user can set triggers and observe the snapshots of the system. A screenshot of the tab can be found in Section B.3.

Set triggers The user can set which triggers to activate 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 for 5 seconds, during which all probes start recording the observed ΔQs (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.

Snapshot window Clicking on a snapshot, a new window opens. The user can explore a frozen state of the system, being able to explore all the ΔQs saved in a snapshot. A screenshot of the snapshot window is provided in Section B.4.

Connection controls

Here, the user can connect to the Erlang endpoint. They can also start the oscilloscope server to receive outcome instances from the adapter. An image of the tab can be found in Section B.5.

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, where it will receive outcome instances from the adapter.

4.4.2 Plots window

To the left, the main window shows the plots of the probes being updated in real time. The window can be seen in Section B.2, Section B.3.

4.5 Triggers

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

Part of future work is extending the available triggers in the oscilloscope. We are aware that the number of available triggers may not seem like much. We believe nevertheless that the triggers we provide are a sufficient basis to observe rare events and detect non-linearity or overload.

4.5.1 Load

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

$$\#\text{instances } (\Delta Q_{obs}) > \text{maxAllowedInstances}_{\text{sampling window}}$$

4.5.2 QTA

A trigger on an observed ΔQ *of a sampling window* can be fired if:

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

Chapter 5

Oscilloscope: implementation

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

- We provide a more in-depth look at the ΔQSD 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.
- Lastly, we briefly talk about the parser generator used to parse the outcome diagram syntax and the dashboard graphical framework.

5.1 ΔQSD implementation

In this section we provide the mathematical foundations of ΔQSD as they are implemented in the oscilloscope.

5.1.1 Histogram representation of ΔQ

We approximate the probability distribution of ΔQ via histograms (Fig. 5.1).

PDF

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 [41]:

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

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

CDF

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

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

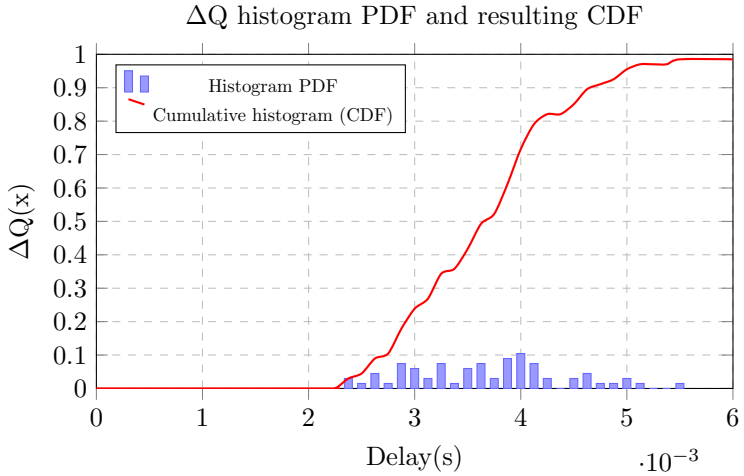

Figure 5.1: Blue bins: PDF of a sample ΔQ. Red: Resulting CDF of ΔQ PDF, the CDF is what is displayed on the dashboard.

5.1.2 dMax

We introduced $dMax$ in the previous chapters (Eq. (4.1)), we provide here the full equation that allows $dMax$ to be calculated:

$$\begin{aligned} \Delta t &= \Delta t_{base} \cdot 2^n \\ dMax &= \Delta t_{base} \cdot 2^n \cdot N \end{aligned} \tag{5.3}$$

Where:

- Δt_{base} represents the base width of a bin, equal to 1ms.
- n the exponent that is set by the user in the dashboard. It is limited to [-10, 10].
- N the number of bins, it is limited to [1, 1000].

We chose 1 ms in combination with 2^n as it allows us to go from very fine bin widths ($\approx$ 1 μs) to large bin widths ($\approx$ 1 s), thanks to the [-10, 10] bounds. Moreover, scaling by a power of 2 allows all the probe's ΔQs to have a common factor to perform operations on them after rebinning (Eq. (5.4)).

5.1.3 Rebinning

Rebinning refers to the aggregation of multiple bins of a bin width i to another bin width j [42]. Previous operations between ΔQs must be done on ΔQs that have the

same bin width. This is why it is fundamental that all probes have a common Δt_{base} and why we have a 2^n factor to calculate the total bin width.

Given two ΔQs ΔQ_i, ΔQ_j, the common bin width Δt_{ij} is:

$$\Delta t_{ij} = \max\{\Delta t_i, \Delta t_j\}$$

The PDF of the rebinned ΔQ (see Fig. 5.2) at bin b, from the original PDF of n bins, where $\Delta t_i > \Delta t_j$ and $k = \frac{\Delta t_i}{\Delta t_j}$:

$$p'_b = \sum_{n=b\cdot k}^{b+1\cdot k-1} p_n, \quad b = 0, 1, \ldots \lceil \frac{N}{k} \rceil \tag{5.4}$$

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.

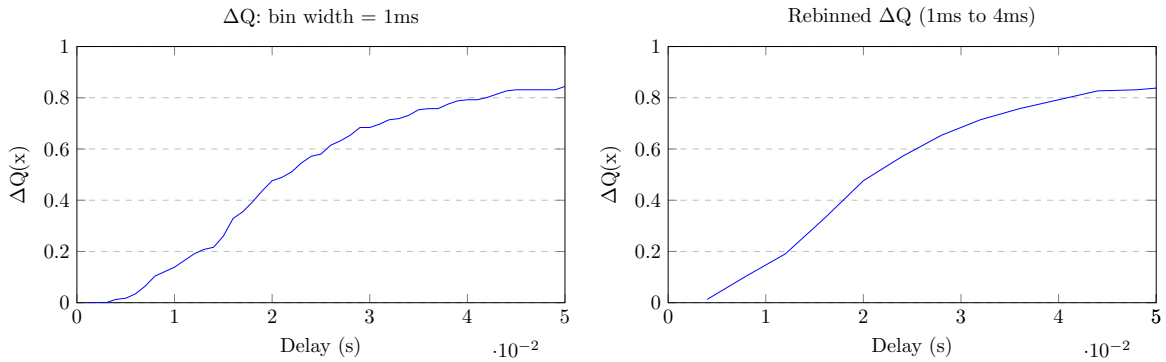

Figure 5.2: **Left**: Sample ΔQ with 1ms bins. **Right:** ΔQ on the left after rebinning to 4ms bins.

5.1.4 Convolution

We present the two solutions to perform convolution we explored during the implementation.

Naïve convolution

Given two ΔQ binned PDFs f and g with equal bin widths, the result of the convolution $f \circledast g$ is given by [43]:

$$(f \circledast g)[n] = \sum_{m=0}^{N} = f[m]g[n-m] \tag{5.5}$$

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. The oscilloscope presented noticeable lag and frame skipping with high resolution. This is why we decided to explore Fast Fourier Transform convolution.

Fast Fourier Transform Convolution

FFTW (Fastest Fourier Transform in the West) is a C subroutine library [44] 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 ΔQs. We adapt our script from an already existing one found on GitHub. [45]

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 Chapter 7.

FFT and naïve convolution produce the same results in our program, barring ε differences (around 10^{-18}) in bins whose result should be 0. This is most likely due to rounding error.

FFTs algorithms are plenty, the choice of the one to use is left up to the subroutine via the parameter `FFTW_ESTIMATE`. [46]

The explanation of the algorithm is beyond the scope of this master thesis, we refer the reader to the following book as a reference. [47]

5.1.5 Arithmetical operations

The FTF, ATF and PC operators on ΔQs use a simple set of arithmetical operations to calculate a ΔQ. The time complexity of FTF, ATF and PC is trivially $\mathcal{O}(N)$ where N is the number of bins.

Scaling (multiplication) A ΔQ can be scaled w.r.t. a constant $0 \leq j \leq 1$. It is equal to binwise multiplication on CDF bins. It is used for the probabilistic choice operator.

$$\hat{f}_r(i) = \hat{f}(i) \cdot j \tag{5.6}$$

Operations between ΔQs Addition, subtraction and multiplication can be done between two ΔQ of equal bin width (but not forcibly of equal length) by calculating the operation between the two CDFs of the ΔQs:

$$\Delta Q_{AB}(i) = \hat{f}_A(i)[\cdot, +, -]\hat{f}_B(i) \tag{5.7}$$

They are used for all operators.

5.1.6 Confidence bounds

Here is how we calculate the mean and lower/upper confidence for the ΔQs CDF at bin $i \quad \forall i < N$. [41]

For x_{ij}, the value of a CDF j at bin i, the mean of n CDFs for the bin i in a polling window is:

$$\mu_i = \frac{1}{n}\sum_{j=1}^{n} x_{ij} \tag{5.8}$$

Its variance:

$$\sigma_i^2 = \frac{1}{n}\sum_{j=1}^{n} x_{ij}^2 - \mu_i^2 \tag{5.9}$$

The lower confidence bound $CI_{i,l}$ and upper confidence bound $CI_{i,u}$ for a bin i can be calculated as:

$$CI_{i,l} = \mu_i - \frac{\sigma_i}{\sqrt{n}} \text{ , } CI_{i,u} = \mu_i + \frac{\sigma_i}{\sqrt{n}} \tag{5.10}$$

The confidence bounds for the CDFs in a polling window are the lower and confidence bounds for every bin i.

5.2 ΔQ Adapter

The ΔQ adapter, called `dqsd_otel` is a rebar3 [48] 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 `?start_span` and `?with_span` and `?end_span` macros. This is to make the adapter less of an encumbrance for the user and to extend the tool usefulness in observing distributed programs.

The adapter will always start OpenTelemetry spans but only start outcome instances if the adapter has been activated. The adapter can be activated in 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: Name of the probe in Erlang binary representation. For example, if the name of the probe is "probe", the binary representation would be constructed with "`«"probe"»`" [49] [50]. We will refer to this as "binary name" from now on.
- 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/2, with_span/3

```
with_span/2: -spec with_span(binary(), fun(() -> any())) -> any().
with_span/3: -spec with_span(binary(), fun(() -> any()), map()) ->
↪  any().
```

Parameters:

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

`with_span` incorporates, thus replaces the OpenTelemetry `?with_span` macro.

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

end_span/2

```
-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`.

`end_span` the OpenTelemetry `?end_span(Ctx)` macro.

fail_span/1

```
-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 a user for the probe being observed, thanks to `erlang:send_after` [51]. 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. It will send an outcome instance to the oscilloscope indicating that an execution of a probe has timed out.

The `span_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.* [52]

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 $dMax$.

The start time and end time are supplied upon starting/ending a span 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 then 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:probe name,
b:Start time (beginning),
e:End time (end time or deadline),
s:The status
```

5.2.3 TCP connection

The adapter is composed of two `gen_server` [53] 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 `dqsd_otel_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 $dMax$s 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 `dqsd_otel_tcp_client` allows the adapter to send the outcome instances 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 [54].

5.3.1 ANTLR

According to ANTLR website [54]:

> "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."

ANTLR is just one of the many parsers generators available in C++ (flex/bison [55], lex/yacc [56]). Although it presents certain limitations, its generated code is simpler to handle and less convoluted with respect to the other possibilities.

ANTLR uses an Adaptive LL(*) *(ALL(*))* parsing strategy, namely, it will "move grammar analysis to parse-time, without the use of static grammar analysis". [57]

5.3.2 Grammar

ANTLR provides a yacc-like metalanguage [57] to write grammars. Due to page limitations, the grammar we have written can be found in Chapter E.

Limitations

A previous version was implemented in Lark [58], 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 ANTLR 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" [59]. 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. We provide a screenshot of the "widget view" in Section B.6.

Chapter 6

Evaluation on synthetic programs

This chapter evaluates the usefulness of the oscilloscope by testing it on three distinct synthetic Erlang applications which can be found in Section K.1. Each application was first represented by an outcome diagram in the oscilloscope. It was then instrumented with the adapter to communicate to the oscilloscope. Although these use cases may be simple, we show that the oscilloscope can be a powerful tool in detecting non-linear behaviour with microseconds precision. The examples are:

- A system with sequentially composed outcomes. It shows how non-linear behaviour can be detected in the oscilloscope at the microseconds level. The example leverages M/M/1/K queues to show how typical queue behaviour is represented on the oscilloscope.
- Then, we provide two applications that perform synchronisation between components. For this application we use two different operators: first-to-finish and all-to-finish. There, we show how these operators can aid in detecting slower components in the system.

6.1 System with sequential composition

The first application has two sequentially composed components. We choose to model the two components as M/M/1/K queues. This is because an average component in a distributed system can be modeled as an M/M/1/K queue. We assume inter-arrival times, execution times for both components are exponentially distributed and a buffer of size K [9]. When the buffer is full, messages are discarded.

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

- λ: The arrival rate.
- The service time s: 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. This is because 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 ΔQ will show that, as the delay will tend to 1.
- Once ρ is approaching high load ($\rho > 0.8$) we can observe the failure increasing quickly. However, we can observe the system starting to get bad after $\rho > 0.5$. [9]

6.1.1 System's outcome diagram

The system (Fig. 6.1) has two components `worker_1`, `worker_2`. Each individual component is composed of a 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 notifies its worker if the worker is not busy. The worker performs N loops of fictional work, they are defined upon start and are done to simulate a component performing a task. Once done, `worker_1` then passes a message to `worker_2`'s queue, which has another queue of size = 1000, it then passes the message to `worker_2`'s worker, which does the same amount of loops as `worker_1`. When a worker completes its work, it notifies its queue, freeing one message from its buffer size and allowing the next message to be processed.

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

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

Lastly, both workers share the same processor, to observe the effects of non-linearity in a distributed setting.

The system can be defined via the previously defined syntax (Section 4.3) as:

```
probe = worker_1 -> worker_2;
```

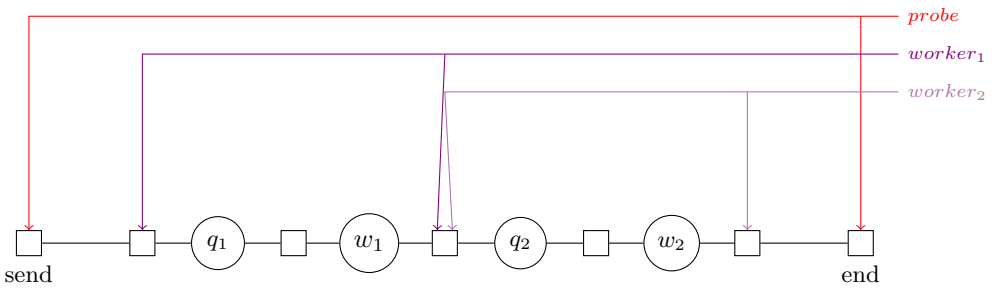

Figure 6.1: Refined outcome diagram of the system. The labeled coloured lines represent the probes that were inserted. $q_{1,2}$ outcomes represent the queues. $w_{1,2}$ represent the workers. As we do not wish to observe the queues, but the whole behaviour of the worker components, we can insert probes from when the message is received to when the worker loops end. We can imagine that $q_{1,2}$ and $w_{1,2}$ are refinements of outcomes $worker_{1,2}$, this is why the probes names differ from the outcomes they observe.

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 (Section 4.1.2). The oscilloscope can provide knowledge of the system. Fig. 6.2 shows an example of worker_1 and worker_2 as observed in the oscilloscope.

The engineer supposes that the workers executions should take a maximum of 4 ms to complete, but doesn't actually know how long the executions should take. The engineer, after having set $dMax = 4ms$, observes the graph Fig. 6.2 on the left in the oscilloscope.

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 worker's behaviour. By setting $dMax$ to 8 ms, they can observe the worker's ΔQs failure approaching 0 on the right Figure in Fig. 6.2.

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

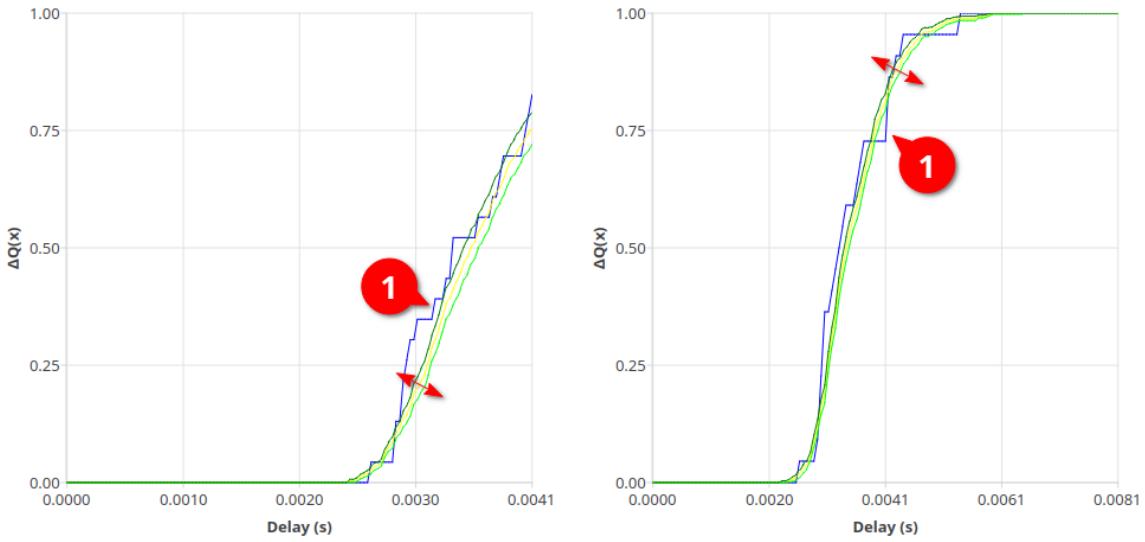

Figure 6.2: **Left**: worker_1 sampling window observed ΔQ with $dMax = 4ms$. Failure tends to 0.25. **Right**: worker_1 sampling window observed ΔQ with $dMax = 8$ ms. Failure tends to 0. Legend for both: (1, red) Sampling window observed ΔQ. (Between red arrow) Its confidence bounds.
In both graphs we can observe how the sampling window observed ΔQ (1) is not precise. The step function representation of it fluctuates. The confidence bounds (between red arrow) provide a more precise representation of worker_1.

6.1.3 Observing non-linearity in the oscilloscope

We can observe the system under different loads to observe how non-linearity can appear in the oscilloscope and how it can be detected.

Low Load

We will send 50 messages per second to observe the system under test to get key properties. The workers do a million loops.
We can observe in the left graph in Fig. 6.3 that the average worker's execution takes $\approx$ 3.3ms. We then have $\mu_{worker} = \frac{1}{0.0033s} \approx 300$ req/s. Thus, $\rho = \frac{50}{300} = 0.1\overline{6}$, we can assume the system is at low load.

At low load (Fig. 6.3), the system is behaving **linearly**. Recall Section 2.1.9, at low load the sum of the two delays will overlap with the observed total delay. We can observe in the oscilloscope the probe **observed ΔQ** and **calculated ΔQ** confidence bounds overlap.

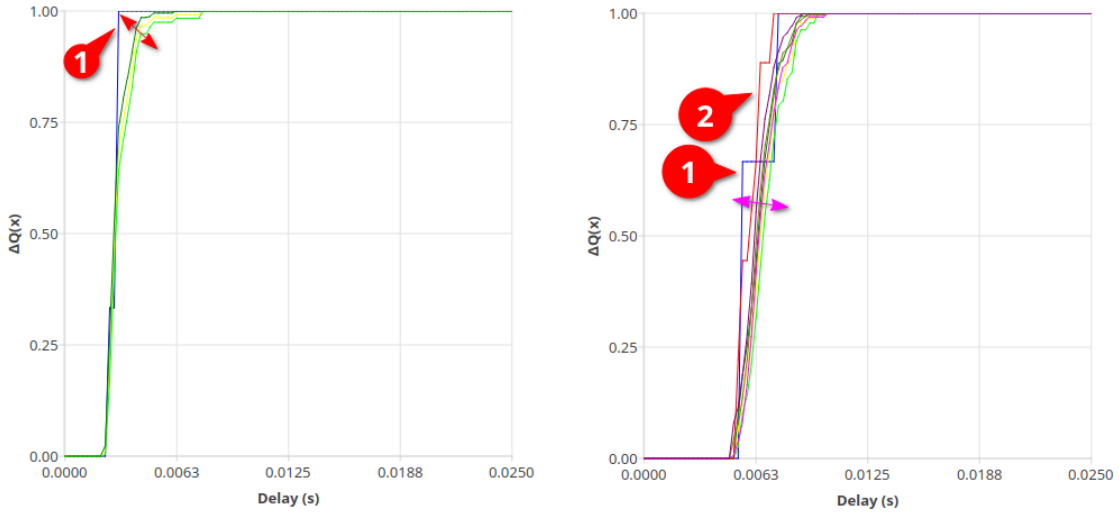

Figure 6.3: Graphs for $\lambda = 50$. **Left**: (1, red) worker_1 sampling window observed ΔQ. (Between red arrow): Its confidence bounds.
Right: (1, red) "probe" sampling window observed ΔQ. (2, red) Sampling window calculated ΔQ. (Between magenta arrows) The observed and calculated ΔQs confidence bounds overlapping.

Early signs of overload

At load = 0.5 the system should start showing early signs of dependent behaviour. We can observe what happens when $\lambda = 150 \rightarrow \rho = 0.5$.

Recall Fig. 2.12, a non-linear system does not obey superposition. The sum of the delay of the workers will deviate from the overall delay of "probe". In Fig. 6.4, this is the case. We can start to observe early signs of dependency even at $\rho = 0.5$. The mean of the observed ΔQ is deviating from the calculated one. At the 50th percentile the deviation is *minimal*, around 0.4 ms. Nevertheless, the precision of the paradigm and the oscilloscope allows even for these minimal deviations to appear on the graphs, being able to recognise *non-linearity* and *early signs of overload*.
The superposition principle is not respected anymore, there is apparent **non-linearity**, and the fact that ΔQ can recognise non-linear behaviour with **0.4ms precision** is

impressive. The oscilloscope is a powerful tool that can help assess dependent behaviour in running systems early on.

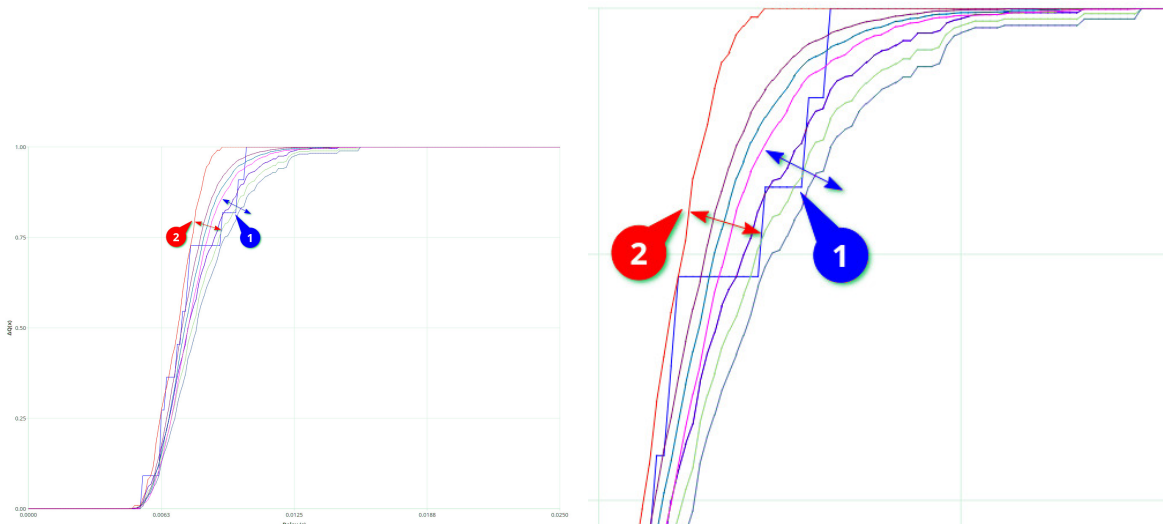

Figure 6.4: **Left**: The probe "probe" ΔQs as observed in a snapshot. **Right**: Zooming in on the oscilloscope, we can observe non-linearity. The observed and calculated means of the ΔQs are diverging. Legend for both: (1, blue) Sampling window observed ΔQ. (Between blue arrow, above): Its confidence bounds. (2, red) Sampling window calculated ΔQ. (Between red arrow, below) Its confidence bounds.
The observed confidence bounds are > than the calculated ΔQ bounds. The deviation at the median is of 0.4 ms, but it is noticeable in the oscilloscope.

High load

Performance degradation at 0.5 offered load is already remarkable, the observed ΔQs and the calculated ones are slowly but surely deviating. The difference is seemingly minimal, but noticeable. We can go even further and observe the system under high load situations (Fig. 6.5). We set $\lambda = 250 \rightarrow \rho = 0.83$, just above the high load threshold.

The results in Fig. 6.5 confirm what is expected by queueing theory. The oscilloscope is capable of observing the basic observation requirements and capable of recognising dependency. While the worker's observed ΔQ is a nice normally distributed CDF with little to no failure, what we observe on the probe is degraded performance in its observed ΔQ mean.
Due to dependency, high load and processor utilisation, the queue is filling up and the workers are taking more time to complete. If you recall Fig. 6.3, the worker's delay distribution was less than the current one. We can see that it has completely degraded, with the average request taking almost double the time as under normal queueing conditions.

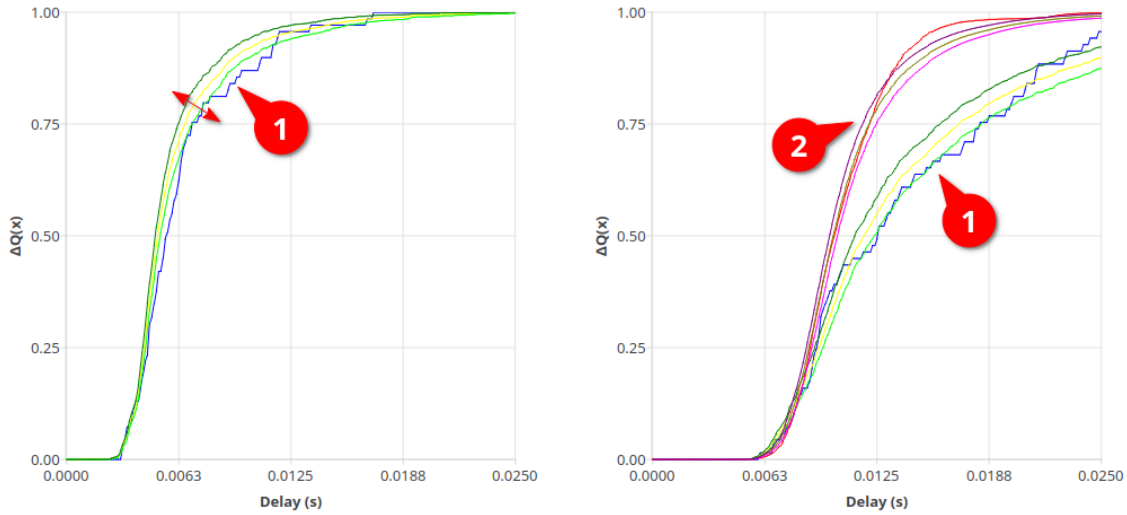

Figure 6.5: Graphs for $\lambda = 250$. **Left**: (1, red) worker_1 sampling observed ΔQ. (Between red arrow) Its confidence bounds. **Right**: "probe" ΔQs (1, red) Sampling window observed ΔQ inside its confidence bounds, (2, red) Sampling window calculated ΔQ inside its confidence bounds.

Further degradation can be observed by increasing $\lambda = 300, 350 \rightarrow \rho \geq 1$ (Fig. 6.6).

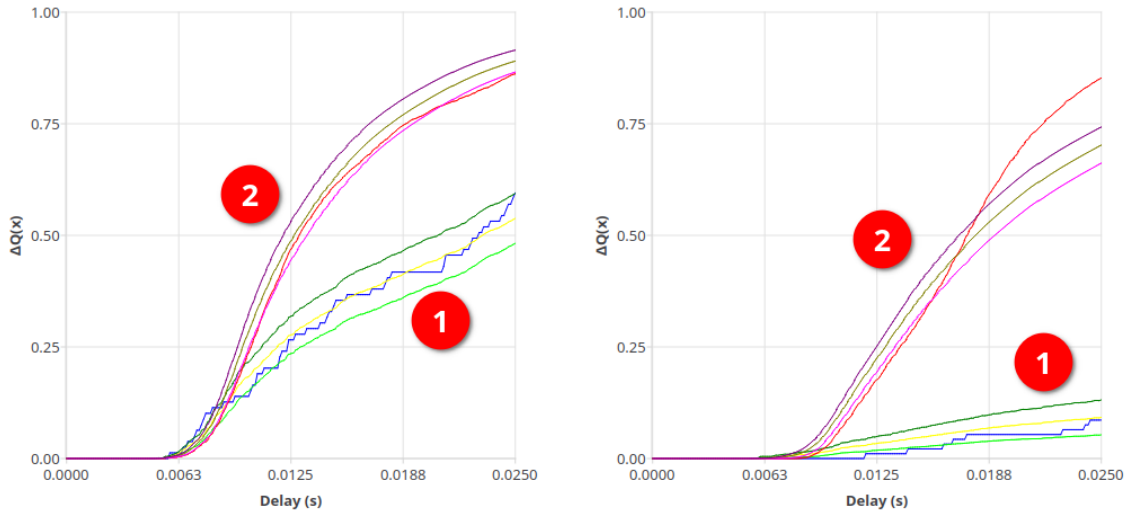

Figure 6.6: "probe" ΔQs. **Left**: $\lambda = 300$. **Right**: $\lambda = 350$. Legend for both: (1, red) Sampling window observed ΔQ inside its confidence bounds. (2, red) Sampling window calculated ΔQ inside its confidence bounds.

The system degradation is clear, the ΔQs show how almost all messages are being dropped or take $> dMax$. We can now look at triggers and how they can be useful to diagnose such cases.

6.1.4 Detecting non-linearity with triggers

We show examples of how triggers can be set to detected non-linearity after having observed the running system in the oscilloscope.

QTA trigger

Recall the previous definition of QTA (Section 4.1.3), by observing the system, we create a QTA for the probe "probe" with: 25% = 0.0075 s, 50% = 0.0125 s, 75% = 0.015s and minimum success rate = 0.9.

By setting the trigger to fire for $\Delta Q_{obs} \not< QTA$. We captured a handful of snapshots (Fig. 6.7). Here, $\lambda = 150$. We can observe how the sampling window observed ΔQ has violated the QTA in 2 minutes. It is apparent that the system is overloaded. It can be seen both by the observed failure, which tends to 0.5, and by the calculated ΔQ which is slightly violating the QTA.

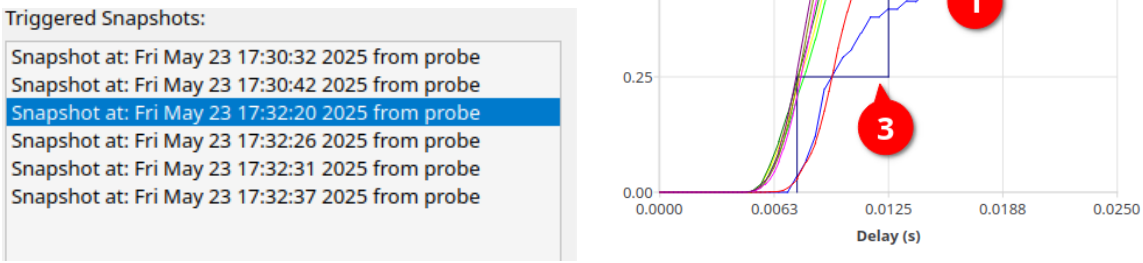

Figure 6.7: **Left**: Snapshots for fired triggers. **Right**: "probe" ΔQs. (1, red) Sampling window observed ΔQ violating the QTA (3, red). (Between blue arrow, below) Its confidence bounds are deviating from the calculated confidence bounds. (2) The sampling window calculated ΔQ is behaving worse than its confidence bounds (Red arrow, above). The system is overloaded, degrading and showing non-linear behaviour, this is captured by the QTA violation.

Trigger on number of instances

QTA triggers can help detect non-linearity even before it becomes evident.
By observing the system under test in high load cases, we have found that the system starts showing non-linear behaviour with $\lambda = 150$. We set the sampling window to 1 second, and then set a trigger on the load of "probe" to fire when outcome instances $\gtrapprox 150$.

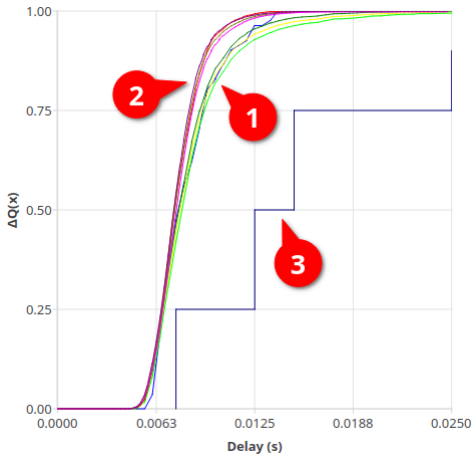

Figure 6.8: Graph of a snapshot for a fired trigger on the load of "probe". (1, red) Sampling window observed ΔQ inside its confidence bounds. (2, red) Sampling window calculated ΔQ inside confidence bounds. The trigger fires for observed load > 150 in a sampling window of 1 second. Even if the QTA requirement (3, red) is not being violated, the system is showing early signs of non-linear behaviour.

By knowing the inner details of the system, setting a QTA on the number of instances can be useful. Fig. 6.8 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 operators

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. The outcome diagram is what is shown in Fig. 2.7. Assume a send request to "the cloud" that waits for a response or a timeout, this is an example of a system that could be modeled by an FTF operator [9].

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 in Fig. 6.1 (1 in Fig. 6.9)? The result on the oscilloscope will clearly show that something is wrong.

```
ftf = worker_1 -> worker_2; (1)
... = f:ftf(worker_1, worker_2); (2)
```

Figure 6.9: Two outcome diagram definitions proposed by the engineer for ftf.

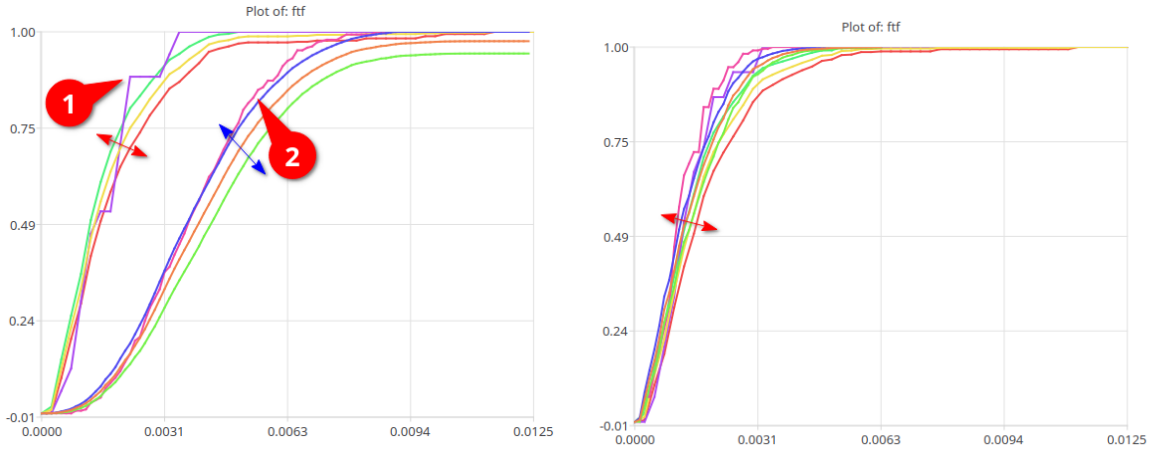

Figure 6.10: **Left**: FTF plot **with wrong outcome diagram definition** as shown in the oscilloscope. (1, red) Observed ΔQ in sampling window with its confidence bounds between red arrows next to it. (2, red) Calculated ΔQ in sampling window with its confidence bounds between blue arrows next to it.
Right: FTF plot **with correct outcome diagram definition**. (Red between arrow) Observed ΔQ and calculated ΔQ confidence bounds overlapping.

On the left in Fig. 6.10, we can observe how the sampling window **calculated ΔQ** and its confidence bounds (2) are clearly greater than the sampling window **observed ΔQ** (1) and its confidence bounds. This difference 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 (2 in Fig. 6.9) is chosen, the two ΔQs (observed and calculated) will overlap, as shown on the right in Fig. 6.10.

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 ΔQs of the FTF operator. Moreover, the FTF ΔQ could be overlaid on top of the faster worker (`worker_1`). In Fig. 6.11, this is the case, the FTF operator can be overlaid on top of worker_1's graph.

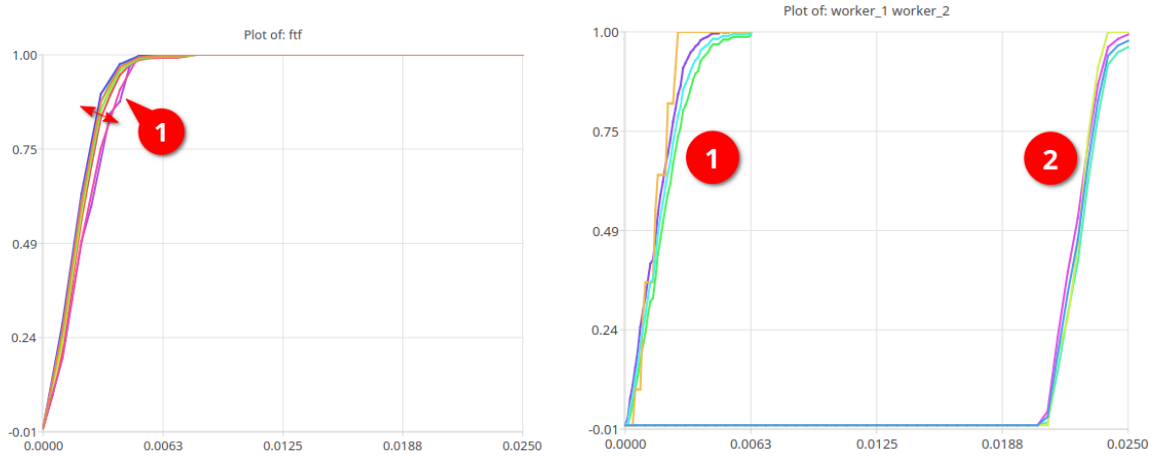

Figure 6.11: **Left**: "ftf" ΔQs. Sampling window observed and calculated ΔQs (1) and their confidences bounds (between red arrow) overlapping. **Right**: worker_1 (1, red) and worker_2 (2, red) ΔQs confidence bounds. The FTF plot correctly displays how worker_2 does not have an effect on the FTF plot. The FTF plot can be overlaid on top of worker_1.

6.2.2 All-to-finish application

We can extend the previous application to an all-to-finish operator. This operator can for instance represent parallel work: a task that requires a lot of computation and can be done in separate pieces by separate workers. [9]
The outcome diagram is the one we presented in Fig. 2.7. It can be represented textually by:

```
... = a:atf(worker_1, worker_2);
```

Introducing a slower component

Like we did for the FTF operator, let's introduce a slower worker into the system. 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). Worker_2 is 2ms slower.

The difference in the worker's ΔQ can be noticed with $\Delta Q_{w2} > \Delta Q_{w1}$ on the right in Fig. 6.12. The difference can then be observed in the all-to-finish plot on the left in Fig. 6.12, where the operator's ΔQs confidence bounds (both observed and calculated) can be overlaid on top of worker_2 ΔQ. This shows once again that the ΔQSD algebraic foundation is sound. Moreover, the oscilloscope can be useful in detecting slower components in a system.

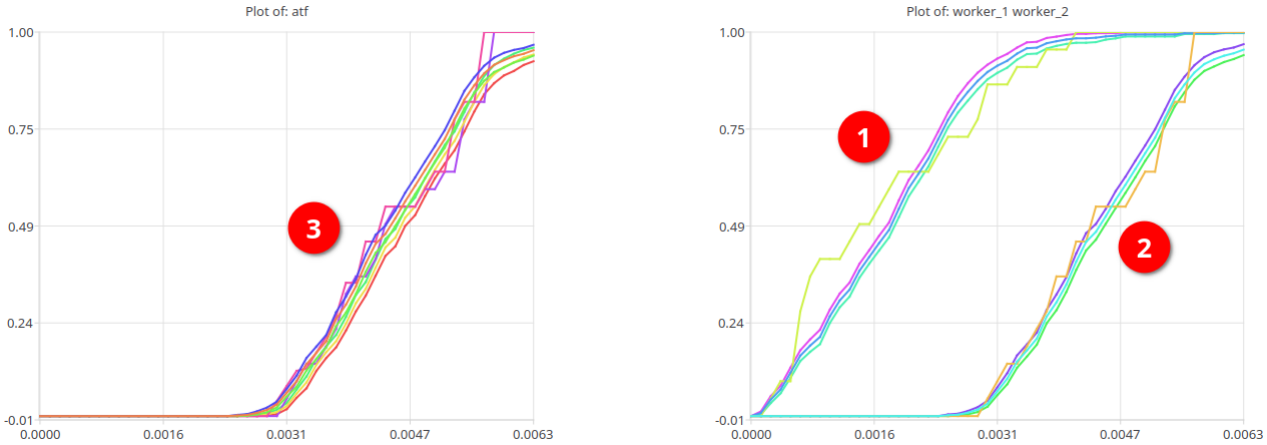

Figure 6.12: **Left**: ATF plot (3, red) with observed and calculated ΔQ confidence bounds overlapping. **Right**: Worker's plots, worker_2 (2, red) is slower than worker_1 (1, red).

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

Chapter 7

Performance evaluation of primitive operations

This chapter evaluates the primitive operations we introduced in previous sections. We analyse their performances in three sections:

- We first evaluate the convolutions algorithms we previously introduced. Previously, 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 both algorithms 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 [60] class when plotting ΔQs.

7.1 Convolution performance

We implemented two versions of the convolution algorithm as described before (Section 5.1.4), the naïve version and the FFT version (Section F.4.3). We compared their performance when performing convolution on two ΔQs of equal bins. In theory, we should observe the naïve version delay quickly grow, while the FFT version have a log-linear growth. We observe the results in Fig. 7.1. 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 $600 > 512$, $300 > 256$ ...). This is because the algorithm is based on ΔQs which are zero-padded to the nearest power of 2, to make the calculation faster.

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

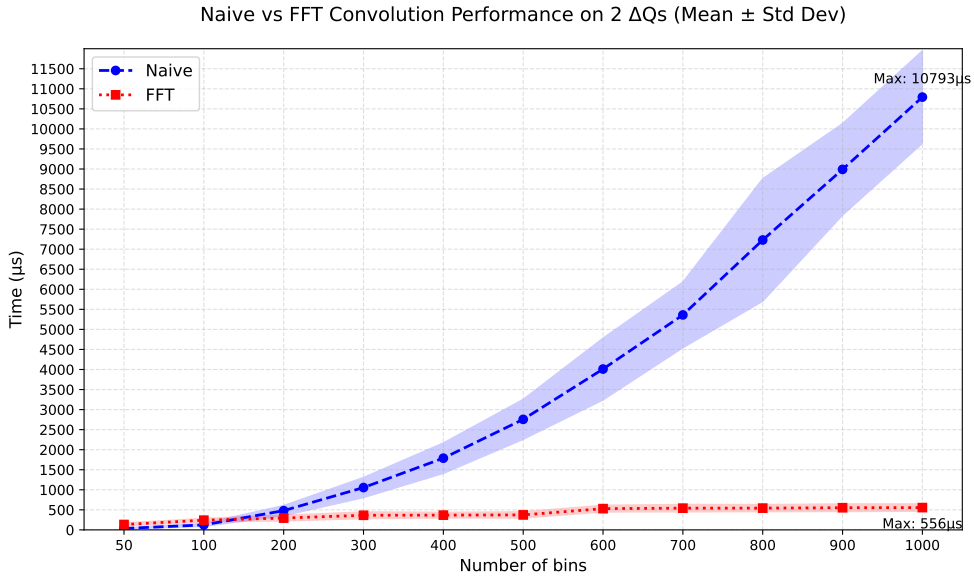

Figure 7.1: Performance comparison of FFT convolution and naïve convolution algorithms.

7.2 GUI plotting performance

We evaluated the performance of the GUI plotting routine. We evaluated the routine when plotting the sampling window observed ΔQ and the mean and confidence bounds of the polling window of observed ΔQs.

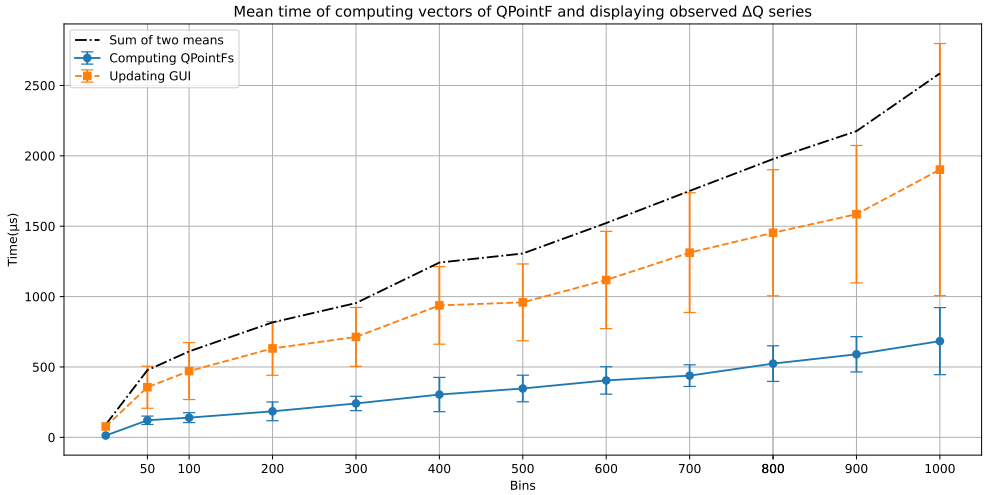

Figure 7.2: Performance of plotting sampling window and polling window observable ΔQ. **(Blue, circle)**: QPointF vectors setup performance. **(Orange, square)**: Plotting performance. **(Black, dotted)**: Sum of the previous two.

The procedure for preparing and plotting the observed and calculated ΔQ (along with the confidence bounds) is the same, so we would need to double the results we have to obtain the total time for the plot of a sub-outcome diagram or an operator. The routine first prepares vectors of QPointF [61], representing all the x and y values of the ΔQs CDF. The vectors are created for the lower bounds, the upper bounds, the mean of the window of ΔQs and the observed ΔQ.

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

The result scales up to almost 2.5 ms for 1000 bins. We believe that these performances are a big bottleneck of the oscilloscope. If we were to plot the calculated ΔQ and its confidence bounds, the time increase would be twofold. If the sampling rate was 100ms and many probes plots were being displayed, some frames would probably be skipped if the number of bins is high. The results we obtained may nevertheless be explained by the specifications of the PC where we ran the tests, namely by the CPU and the GPU (Chapter A).

7.3 ΔQ adapter performance

We evaluated the performance of the adapter to measure its impact in a distributed application. We tested the following calls which represent a normal usage of the adapter.

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

We ran the simulation for 25000 subsequent iterations, the results are shown in Fig. 7.3.

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. This shows that the adapter can be integrated seamlessly into an application that was instrumented with OpenTelemetry without presenting noticeable overhead.

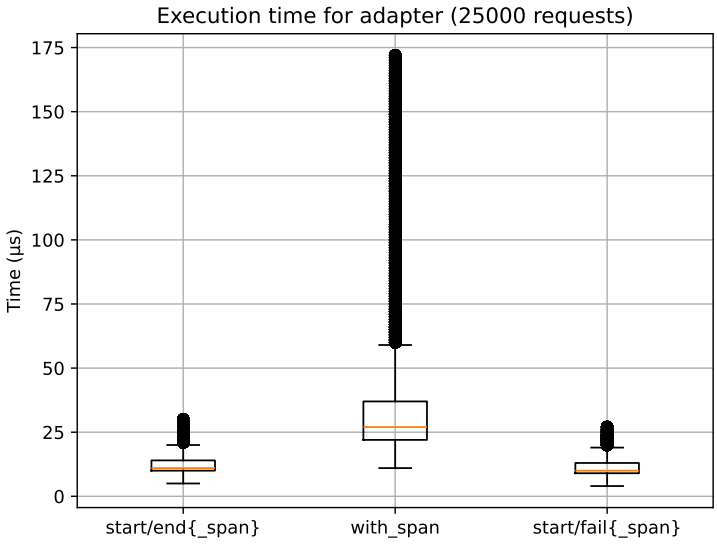

Figure 7.3: Adapter performance evaluation.

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's behaviour in real time thanks to the ΔQSD paradigm. While we can not say that we are fully 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 interface 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 of the adapter client and server allows for this to be done fast and asynchronously.

The ΔQ oscilloscope is 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 to the user of the outcome diagrams, the parameters of probes, their QTAs, the triggers they want to set for a given probe, and the snapshots to observe the system as if it was frozen in time. In fine, the FFT convolution algorithms 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.

8.1 Future improvements

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 adapter - 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 [62] 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 ΔQSD, 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 and part of the dissemination project of ΔQSD. 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 ΔQSD in analysing its behaviour.

8.1.2 OpenTelemetry improvements

As we explored in a previous chapter (Section 2.3.1), long-lived spans are currently a problem. ΔQSD requires that it must be possible to consider a span as failed if it takes longer than the maximum delay required by the user. This delay can be set by the user and depends on the application's required behaviour (Section 4.1.2). This is not trivial to do within the current implementation of the OpenTelemetry standard in Erlang. We believe that what we have shown in this thesis should be the standard to improve observability requirements and to ensure essential timeliness requirements. This could be done via events that are triggered when the delay surpasses the required one and are directly exported to a monitoring tool. We are nevertheless aware that events are currently unstable in the current version of OpenTelemetry in Erlang.

8.1.3 Adapter improvement

As suggested by Bryan Naegele, a member of the observability group of Erlang, the adapter, instead of calling OpenTelemetry macros inside the interface, could directly use the OpenTelemetry span processor. Leveraging `erlang:send_after` as we already do, we could create outcome instances with telemetry [63] events (which must not be mistaken with OpenTelemetry ones) to handle successful executions and timeouts. The span processor will then be responsible for creating outcome instances, without creating the need for calling OpenTelemetry in the adapter, like we do now.

8.1.4 Real applications

The current prototype of the ΔQ oscilloscope has not been tested on real distributed applications. While its usefulness has been proven on synthetic applications, the lack of real life applications is a weakness.

8.1.5 Licensing limitations

Lastly, a notable limitation is created by **Qt**, namely, QtCharts. QtCharts has a GPLv3 license. Therefore, the usage of Qt does not allow us to release our project under BSD/MIT licenses or LGPL license, but rather a GPLv3 one. [64]

Bibliography

[1] Solarwinds Loggly. *Distributed Systems Monitoring: The Essential Guide*. Accessed: (31/05/2025). URL: https://www.loggly.com/use-cases/distributed-systems-monitoring-the-essential-guide/.

[2] Aaron Abbott. *Sunsetting OpenCensus*. Accessed: (31/05/2025). 2023. URL: https://opentelemetry.io/blog/2023/sunsetting-opencensus/.

[3] Juraci Paixao Krohling David Allen. *OpenTelemetry and vendor neutrality*. Accessed: (31/05/2025). 2024. URL: https://grafana.com/blog/2024/09/12/opentelemetry-and-vendor-neutrality-how-to-build-an-observability-strategy-with-maximum-flexibility/.

[4] OpenTelemetry Authors. *Vendors who natively support OpenTelemetry*. Accessed: (31/05/2025). 2025. URL: https://opentelemetry.io/ecosystem/vendors/.

[5] The Zipkin Authors. *Zipkin*. Accessed: (01/06/2025). 2025. URL: https://zipkin.io/.

[6] OpenTelemetry Authors. *OpenTelemetry - Traces*. Accessed: (19/05/2025). 2025. URL: https://opentelemetry.io/docs/concepts/signals/traces/.

[7] Magsther. *Awesome OpenTelemetry*. Accessed: (31/05/2025). 2025. URL: https://github.com/magsther/awesome-opentelemetry?tab=readme-ov-file.

[8] The Jaeger Authors. *Service Performance Monitoring (SPM)*. Accessed: (31/05/2025). 2025. URL: Service%20Performance%20Monitoring%20(SPM).

[9] 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.

[10] *PNSol: the network performance science company*. Accessed: (31/05/2025). 2025. URL: https://www.pnsol.com/.

[11] 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.

[12] 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.

[13] J. H. Nyström, P. W. Trinder, and D. J. King. "Evaluating distributed functional languages for telecommunications software". In: *Proceedings of the 2003 ACM SIGPLAN Workshop on Erlang*. ERLANG '03. Uppsala, Sweden: Association for

Computing Machinery, 2003, pp. 1–7. ISBN: 1581137729. DOI: 10.1145/940880.940881. URL: https://doi.org/10.1145/940880.940881.

[14] Erlang/OTP. *What is Erlang*. Accessed: (26/05/2025). 2025. URL: https://www.erlang.org/faq/introduction.html.

[15] Francesco Nieri. *UCLouvain Master thesis poster presentation - ΔQ oscilloscope*. 2024.

[16] 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.

[17] Peter Van Roy. *LINFO2345 lessons on ΔQSD*. Accessed: (19/05/2025). UCLouvain, 2023. URL: https://www.youtube.com/watch?v=tF7fbU9Gce8.

[18] Neil J. Davies and Peter W. Thompson. *ΔQSD workbench - GitHub*. Accessed: (19/05/2025). 2022. URL: https://github.com/DeltaQ-SD/dqsd-workbench.

[19] Erlang programming language. *Erlang tracing*. Accessed: (19/05/2025). 2024. URL: https://www.erlang.org/doc/apps/erts/tracing.html.

[20] OpenTelemetry Authors. *OpenTelemetry in Erlang/Elixir*. Accessed: (19/05/2025). 2025. URL: https://opentelemetry.io/docs/languages/erlang/.

[21] 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.

[22] OpenTelemetry Authors. *What is OpenTelemetry?* Accessed: (19/05/2025). 2025. URL: https://opentelemetry.io/docs/what-is-opentelemetry/.

[23] Erlang Programming Language. *Erlang - trace*. Accessed: (01/06/2025). 2025. URL: https://www.erlang.org/doc/apps/kernel/trace.html.

[24] OpenTelemetry Authors. *Language API & SDKs*. Accessed: (28/05/2025). 2025. URL: https://opentelemetry.io/docs/languages/.

[25] OpenTelemetry Authors. *Instrumentation for OpenTelemetry Erlang/Elixir*. Accessed: (19/05/2025). 2025. URL: https://opentelemetry.io/docs/languages/erlang/instrumentation/.

[26] Erlang Ecosystem Foundation. *Observability Working Group*. Accessed: (28/05/2025). 2025. URL: https://erlef.org/wg/observability.

[27] OpenTelemetry Authors. *Observability primer - Distributed traces*. Accessed: (28/05/2025). 2024. URL: https://opentelemetry.io/docs/concepts/observability-primer/.

[28] OpenTelemetry Authors. *Exporters*. Accessed: (28/05/2025). 2025. URL: https://opentelemetry.io/docs/languages/erlang/exporters/.

[29] The Jaeger Authors. *Jaeger*. Accessed: (19/05/2025). 2025. URL: https://www.jaegertracing.io/.

[30] Datadog. *Datadog - Product*. Accessed: (28/05/2025). 2025. URL: https://www.datadoghq.com/product/.

[31] 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/.

[32] 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/.

[33] OpenTelemetry Authors. *Active spans, C++ Instrumentation*. Accessed: (19/05/2025). 2025. URL: https://opentelemetry.io/docs/languages/cpp/instrumentation/.

[34] Hazel Weakly. *OpenTelemetry Challenges: Handling Long-Running Spans*. Accessed: (21/05/2025). 2024. URL: https://thenewstack.io/opentelemetry-challenges-handling-long-running-spans/.

[35] Inc. Embrace Mobile. *Embrace*. Accessed: (31/05/2025). 2025. URL: https://embrace.io/.

[36] OpenTelemetry authors. *Error handling*. Accessed: (29/05/2025). 2022. URL: https://opentelemetry.io/docs/specs/otel/error-handling/.

[37] Francesco Nieri. *dqsd_otel*. Accessed: (28/05/2025). 2025. URL: https://github.com/fnieri/dqsd_otel.

[38] Francesco Nieri. *ΔQOscilloscope*. Accessed: (28/05/2025). URL: https://github.com/fnieri/DeltaQOscilloscope.

[39] 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.

[40] 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.

[41] FM Dekking et al. *A modern introduction to probability theory and statistics*. Vol. Springer texts in statistics. Springer, 2005. ISBN: 1-85233-896-2.

[42] Brian Pauw. *What does rebinning do?* Accessed: (31/05/2025). 2015. URL: https://lookingatnothing.com/index.php/archives/2097.

[43] Steven B. Damelin and Willard Miller Jr. *The Mathematics of Signal Processing*. USA: Cambridge University Press, 2012. ISBN: 1107601045.

[44] FFTW3. *Fastest Fourier Transform in The West*. Accessed: (19/05/2025). 2025. URL: https://www.fftw.org/.

[45] Jeremy Fix. *FFTConvolution*. Accessed: (21/05/2025). 2013. URL: https://github.com/jeremyfix/FFTConvolution/blob/master/Convolution/src/convolution_fftw.h.

[46] *Planning-rigor flags*. Accessed: (23/05/2025). URL: https://www.fftw.org/doc/Planner-Flags.html.

[47] K. R. Rao, D. N. Kim, and J.-J. Hwang. *Fast Fourier Transform - Algorithms and Applications*. 1st. Springer Publishing Company, Incorporated, 2010. ISBN: 1402066287.

[48] Rebar3. *Rebar3 Basic Usage*. Accessed: (25/05/2025). 2025. URL: https://rebar3.org/docs/basic_usage/.

[49] Erlang. *Binary module*. Accessed: (30/05/2025). URL: https://www.erlang.org/docs/24/man/binary.

[50] Erlang System Documentation. *Bit syntax*. Accessed: (30/05/2025). 2025. URL: https://www.erlang.org/doc/system/bit_syntax.html.

[51] Erlang. *Timer functions*. Accessed: (01/06/2025). 2025. URL: https://www.erlang.org/doc/apps/stdlib/timer.html.

[52] Erlang System Documentation. *Signals/Sending signals*. Accessed: (24/05/2025). 2025. URL: https://www.erlang.org/doc/system/ref_man_processes.

[53] Erlang Reference Manual. *gen_server*. Accessed: (01/06/2025). 2024. URL: https://www.erlang.org/docs/24/man/gen_server.

[54] ANTLR. *What is ANTLR4?* Accessed: (19/05/2025). 2025. URL: https://www.antlr.org/.

[55] skenz.it. *Flex Bison*. Accessed: (31/05/2025). 2024. URL: https://www.skenz.it/compilers/flex_bison.

[56] Thomas Niemann. *Lex/Yacc*. Accessed: (31/05/2025). URL: https://arcb.csc.ncsu.edu/~mueller/codeopt/codeopt00/y_man.pdf.

[57] 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.

[58] lark-parser. *Lark - A parsing toolkit for Python*. Accessed: (24/05/2025). 2025. URL: https://github.com/lark-parser/lark.

[59] Wikipedia. *Qt (software) Wikipedia, The Free Encyclopedia*. Accessed: (24/05/2025). 2025. URL: https://en.wikipedia.org/wiki/Qt_(software).

[60] The Qt Company. *QtCharts*. Accessed: (01/06/2025). 2025. URL: https://doc.qt.io/qt-6/qtcharts-index.html.

[61] The Qt Company. *QPointF class*. Accessed: (01/06/2025). 2025. URL: https://doc.qt.io/qt-6/qpointf.html.

[62] The ZeroMQ authors. *ZeroMQ*. Accessed: (01/06/2025). 2025. URL: https://zeromq.org/.

[63] ExDoc for the Erlang programming language. *telemetry*. Accessed: (01/06/2025). 2025. URL: https://hexdocs.pm/telemetry/telemetry.html.

[64] 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

Part I

Addendum to Chapters 4 & 5

Appendix B

Oscilloscope screenshots

B.1 System/Handle plots tab

Here is a screenshot from the tab to create the system and handle plots (Fig. B.1).

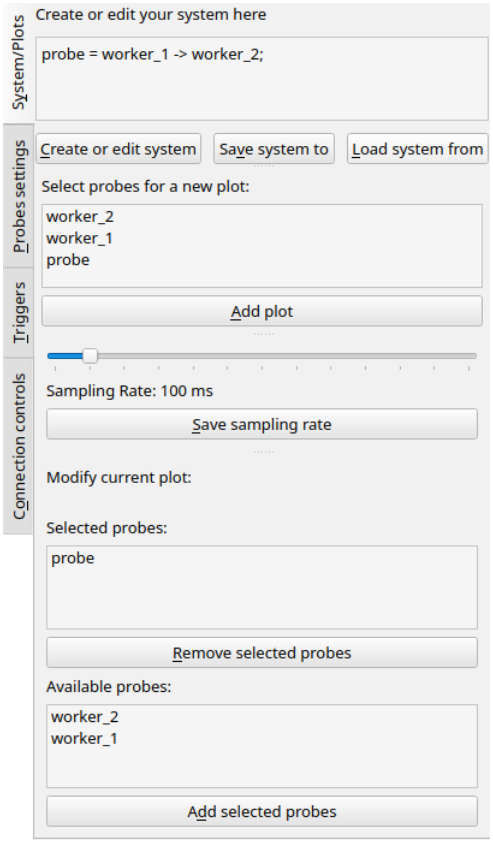

Figure B.1: System/Handle plots tab in oscilloscope.

B.2 Parameters tab

Here is a screenshot of the parameters tab, with a set QTA showing up on a plot (Fig. B.2).

Figure B.2: Parameters tab in oscilloscope. The QTA set for worker_1 can be seen on the plot.

B.3 Triggers tab

Here is a screenshot of the triggers tab, with saved snapshots, QTA violation set for probe and a running plot (Fig. B.3).

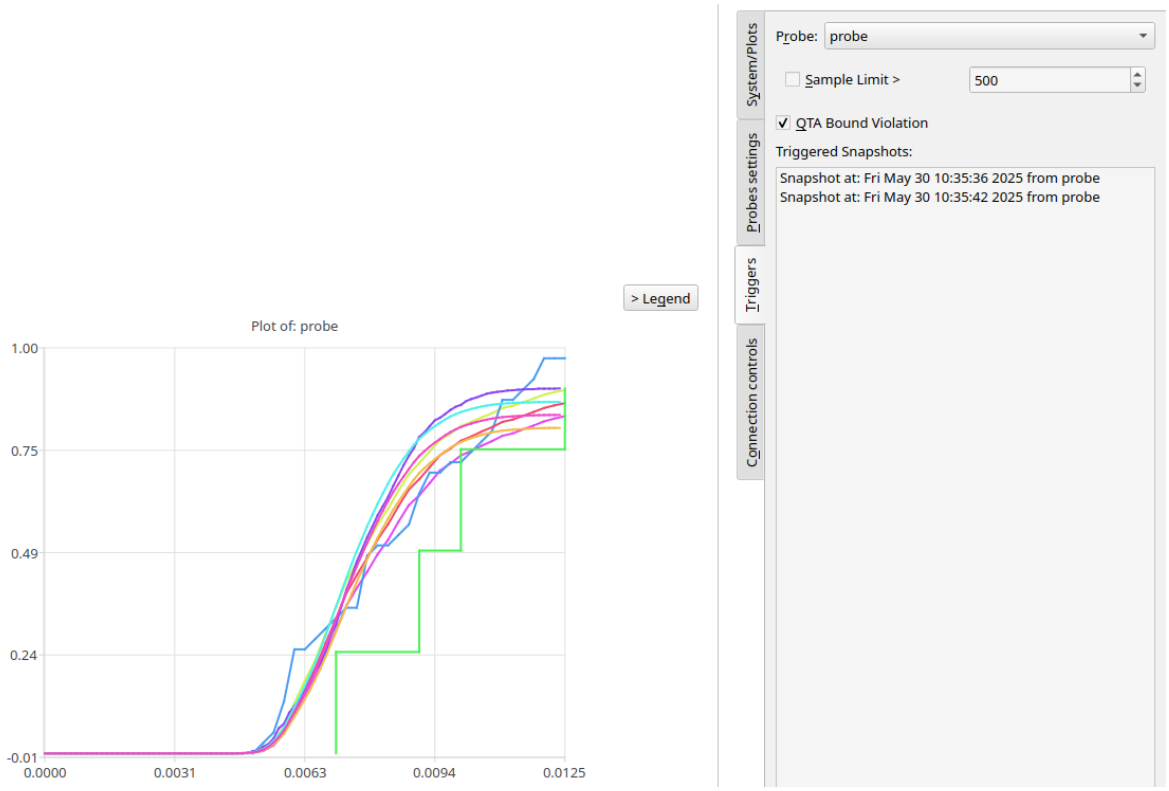

Figure B.3: Triggers tab in oscilloscope. The snapshots for the whole system are saved and can be observed.

B.4 Snapshot window

Here is a screenshot from the window observing a snapshot (Fig. B.5).

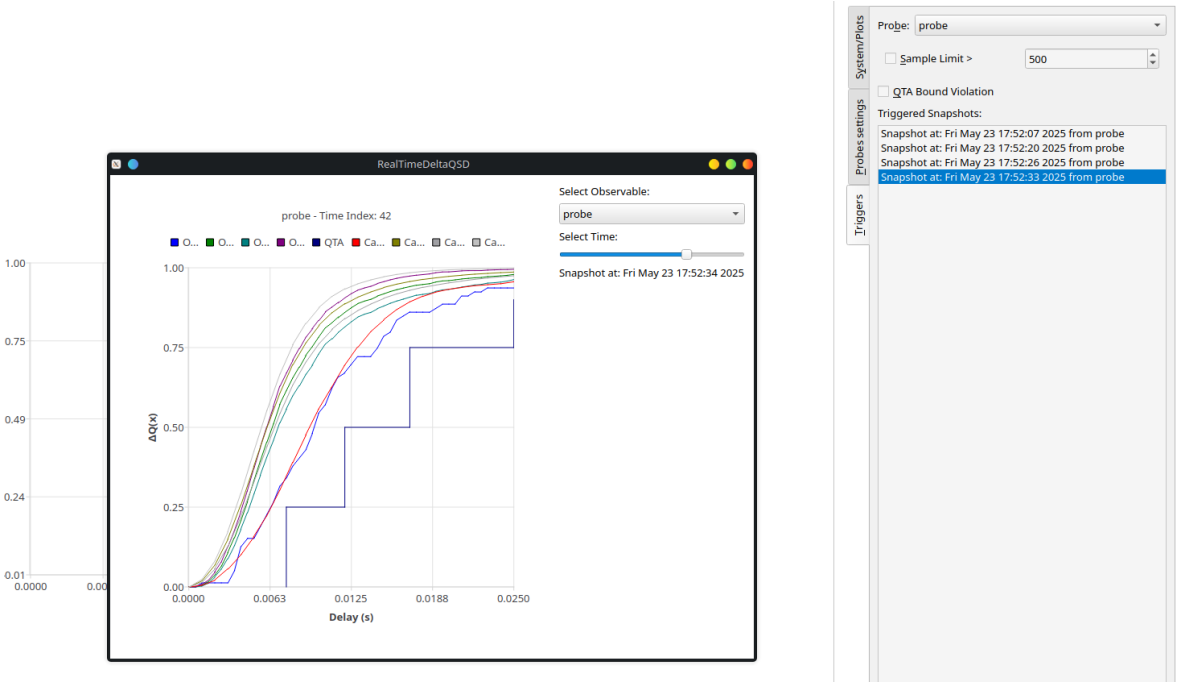

Figure B.4: Snapshot window (above) over the running oscilloscope below.

B.5 Connection controls

Here is a screenshot of the available controls in the oscilloscope (Fig. B.5).

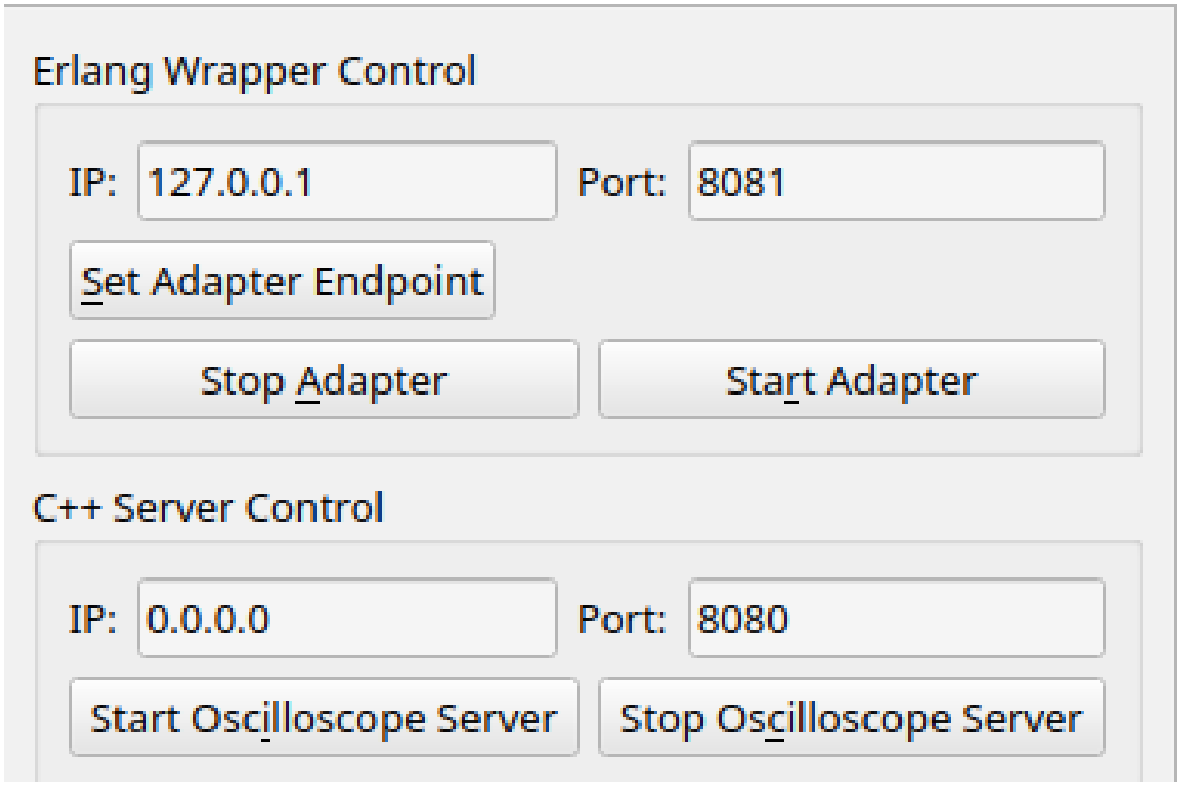

Figure B.5: Connections control tab in the oscilloscope with the erlang and oscilloscope endpoints.

B.6 Widget view of GUI

The GUI is composed of multiple building block, the widgets. The screenshot we provide here highlights the multiple widgets present in the GUI (Fig. B.6). These widgets could be broken down into the multiple subwidgets that compose them.

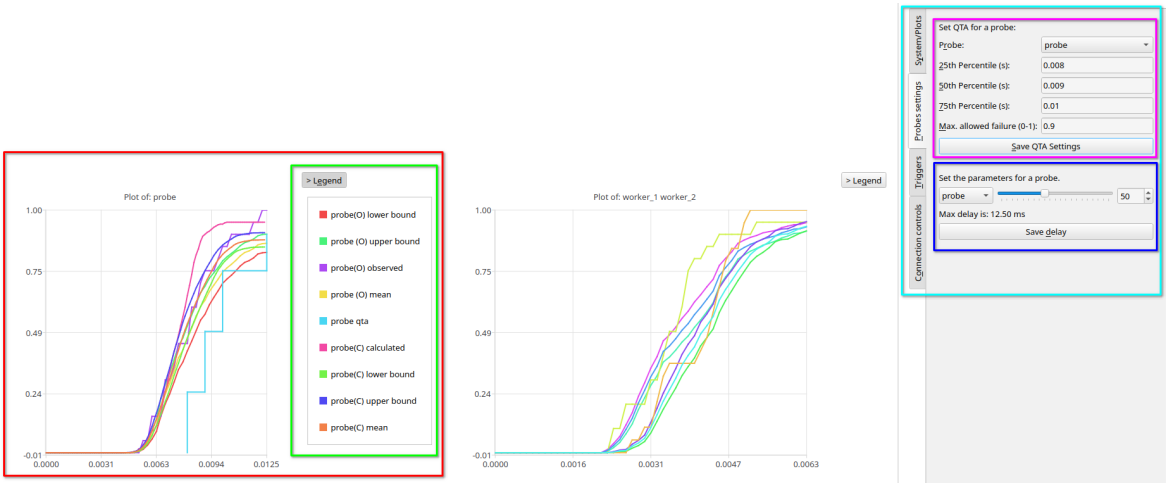

Figure B.6: Widget view of the oscilloscope dashboard.

Part II

User manual

Appendix C

How to download and launch

To download the oscilloscope, go to the GitHub repository (`https://github.com/fnieri/DeltaQOscilloscope/`) and go to the releases page, there, you will find the different versions of the oscilloscope. For instruction purposes, we have made a pre-release to show where to find it.

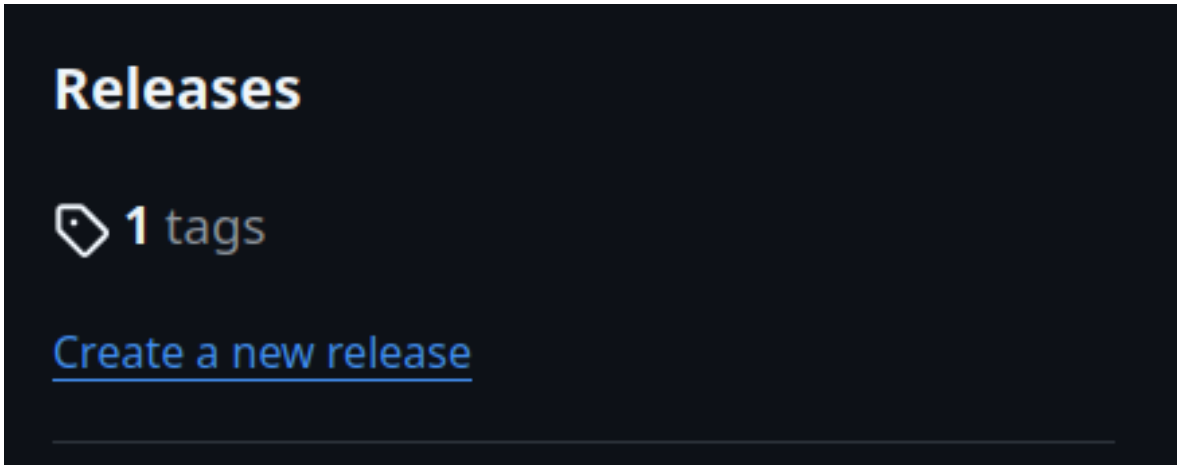

Figure C.1: Releases tab.

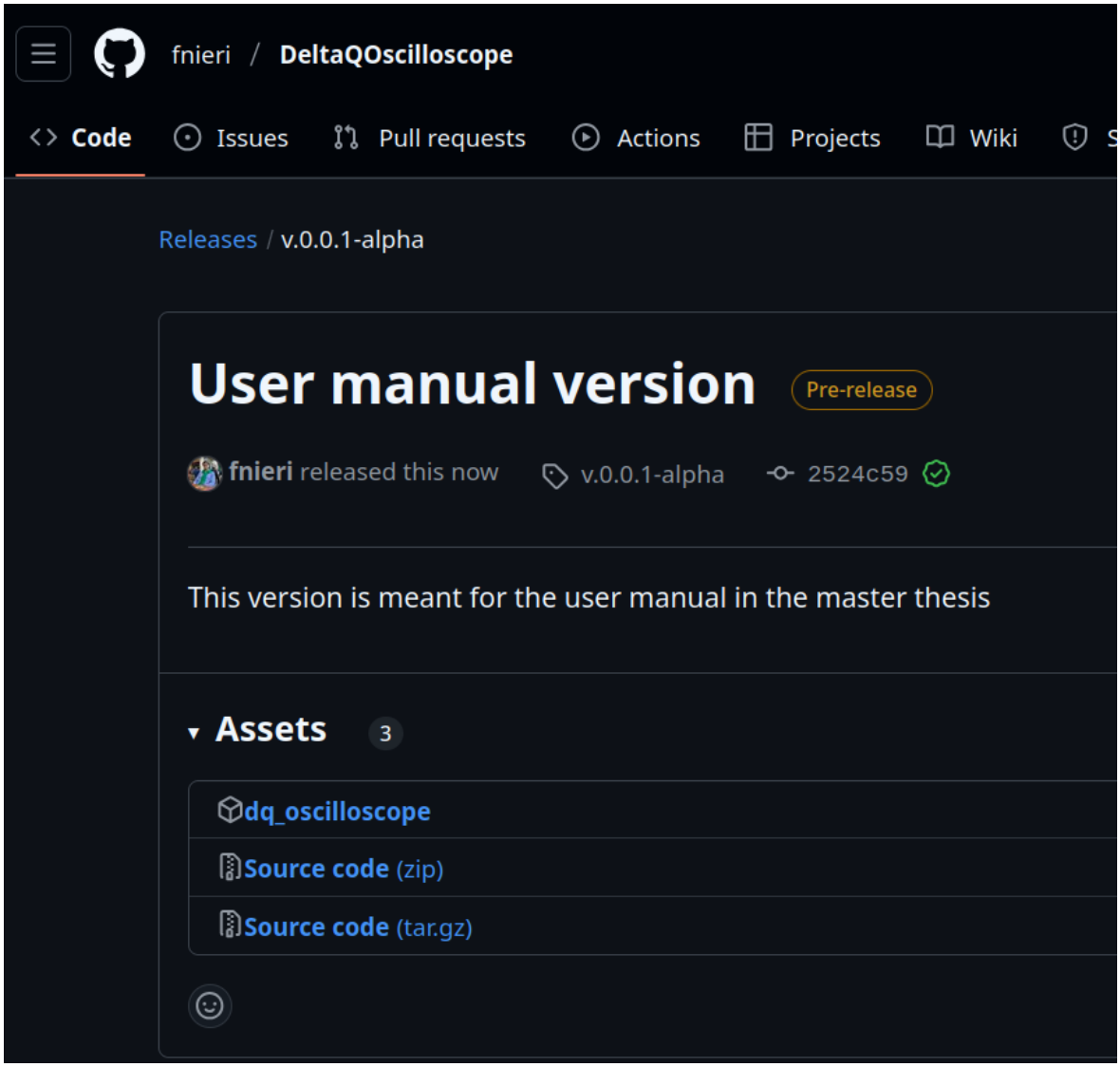

Figure C.2: Sample release

The dq_oscilloscope binary is the binary to download. By launching the binary on the CLI with:

```
./dq_oscilloscope
```

The oscilliscope will launch.

Appendix D

Oscilloscope: How to use

D.1 Establishing the adapter - oscilloscope connection

To connect to the oscilloscope to the server, you first need to start the oscilloscope server by setting the oscilloscope listening IP and port on the dashboard (Fig. D.1).

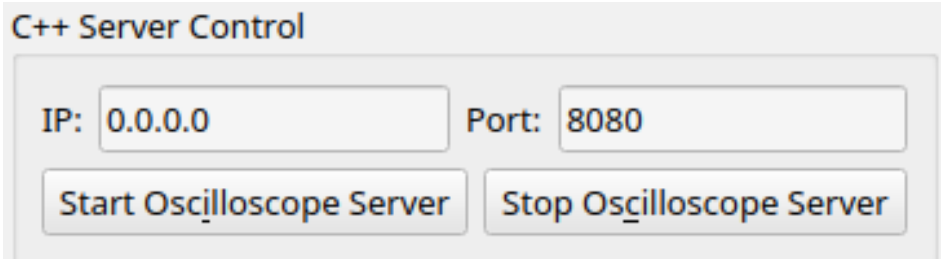

Figure D.1: Widget to set server endpoint in the oscilloscope

If the server cannot start, a popup will appear with the error.

Once this is done, the adapter can establish a connection to the oscilloscope to send outcome instances (Fig. D.2). The adapter can now connect to the oscilloscope server.

```
1> dqsd_otel_tcp_client:try_connect("127.0.0.1", 8080).
ok
dqsd_otel: Adapter connected to 127.0.0.1:8080
2>
```

Figure D.2: Establish connection from adapter to oscilloscope.

We now need to start the listener on the adapter (Fig. D.3).

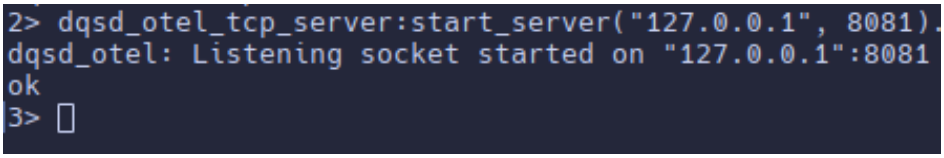

Figure D.3: Adapter starting listener for commands and parameters from the oscilloscope .

If an error may arrive during the start-up of the listener, it will be printed out.

Now, we can connect the oscilloscope to the adapter by setting the listener endpoint (Fig. D.4).

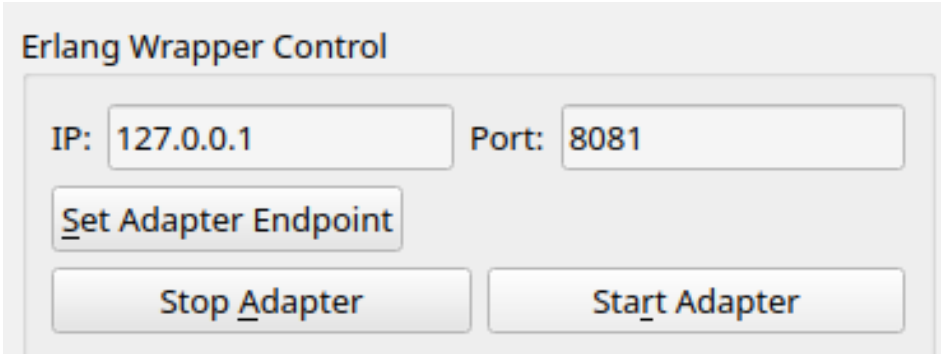

Figure D.4: Set Erlang listener endpoint in oscilloscope.

If the server cannot connect, an error will pop up. Once connected, the server can start and stop the adapter sending of the spans by clicking the two buttons below.

D.2 Sidebar: Outcome diagram and plots

D.2.1 Creating the system/outcome diagrams

You need to provide a textual description of your outcome diagram following the syntax which was defined previously (Section 4.3). You can create your system by clicking on the **Create or edit system** button (Fig. D.5). If the parser successfully parsed the text, your system will be created, and you can start setting the parameters for your probes.

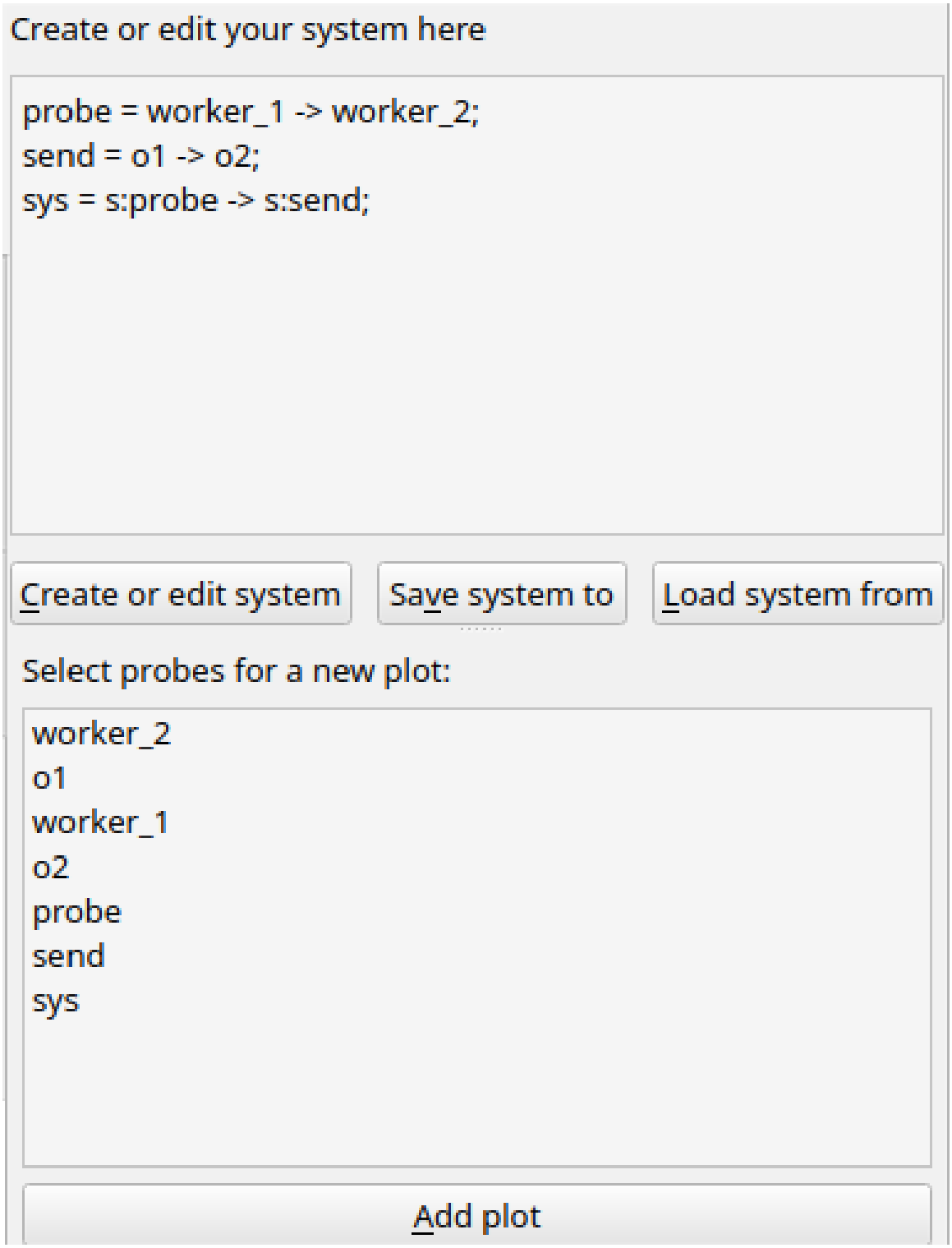

Figure D.5: After selecting create system, if the system definition is correct, the inserted probes will be available to be plotted.

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 (Fig. D.6). If there is a cycle, it will specify where it found the cycle.

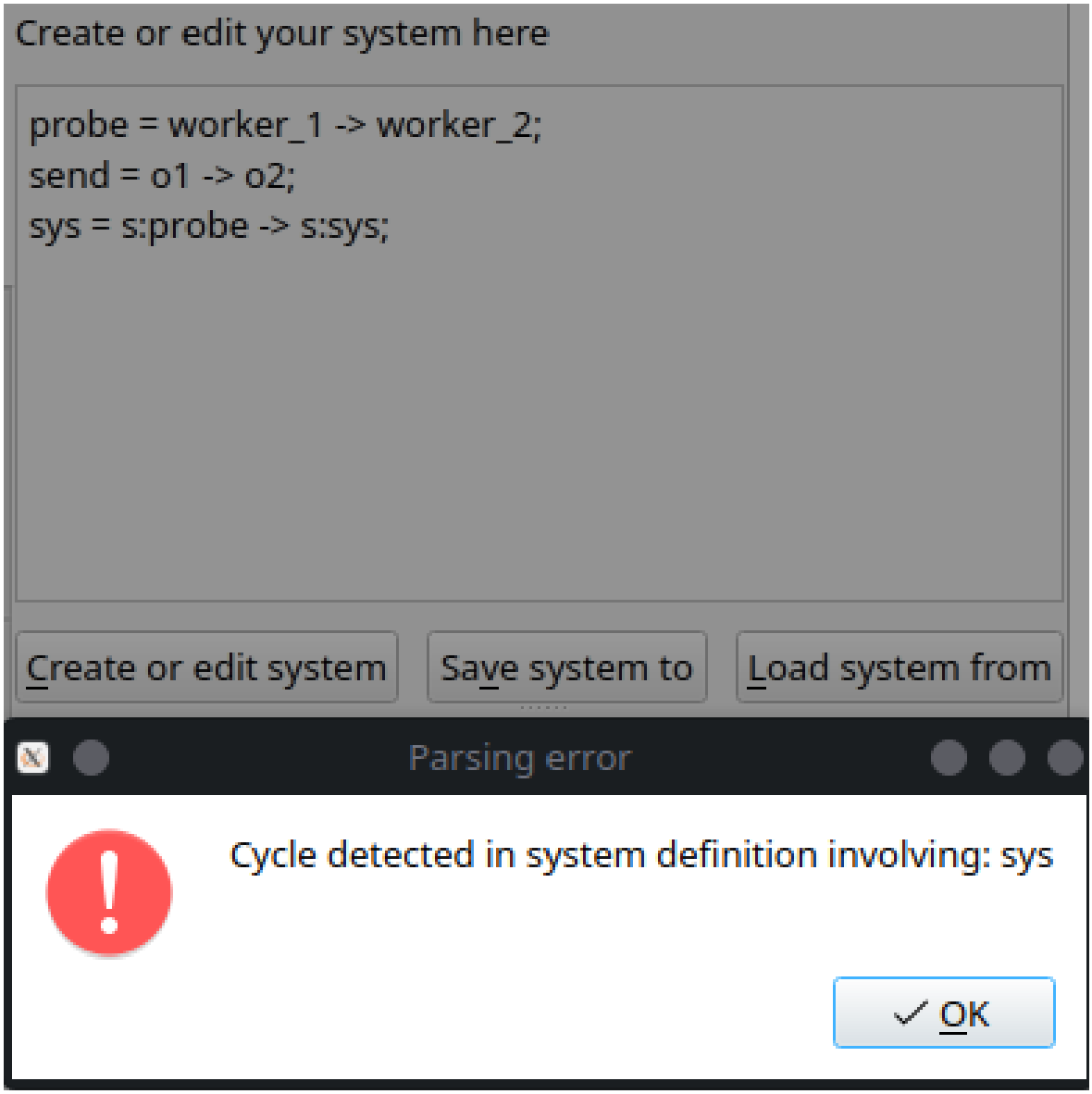

Figure D.6: Creation of a system with a cycle

D.2.2 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 (Fig. D.7).

D.2.3 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 (Fig. D.7). As for the extension, you can load any file extension.

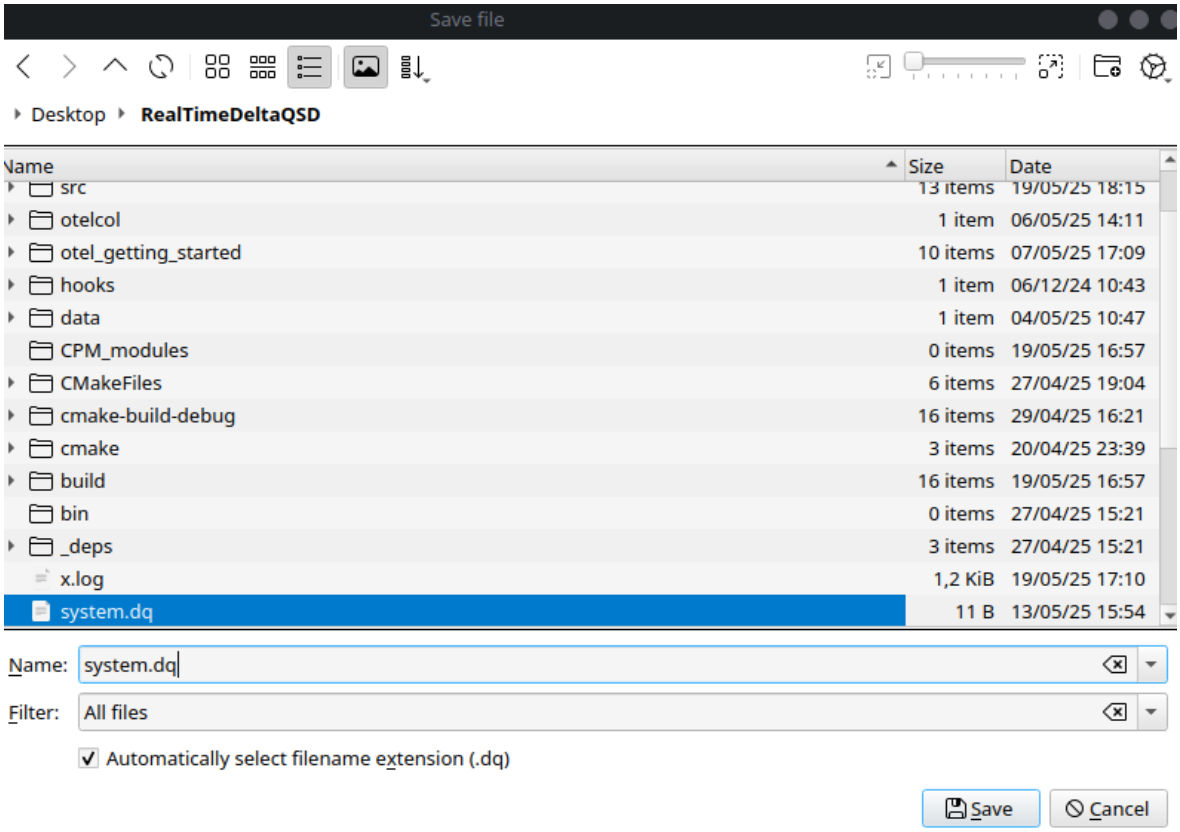

Figure D.7: Example popup where the user can select a file where the system definition is loaded.

D.2.4 Changing the sampling rate

The user can set the sampling rate via the slider in this tab (Fig. D.8):

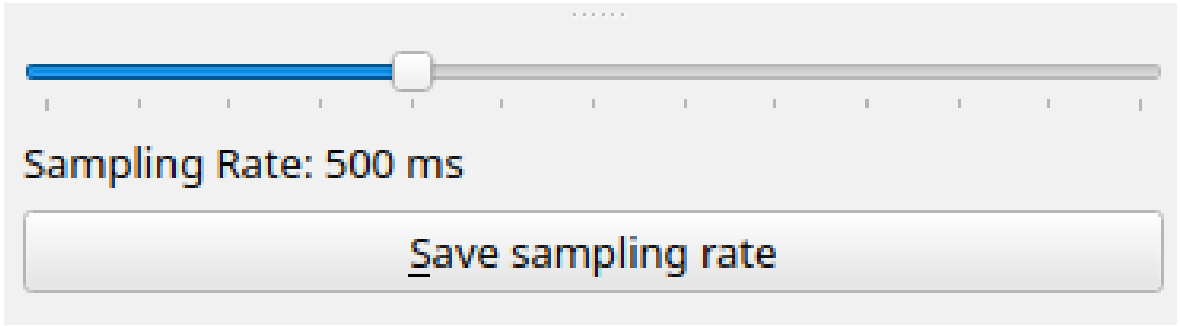

Figure D.8: Slider widget.

D.2.5 Managing the plots

Once you have your system defined you can start adding plots of the ΔQs 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 (Fig. D.9). Then by clicking the **"Add plot"** button, the selected probes will be added to the plot (Fig. D.10).

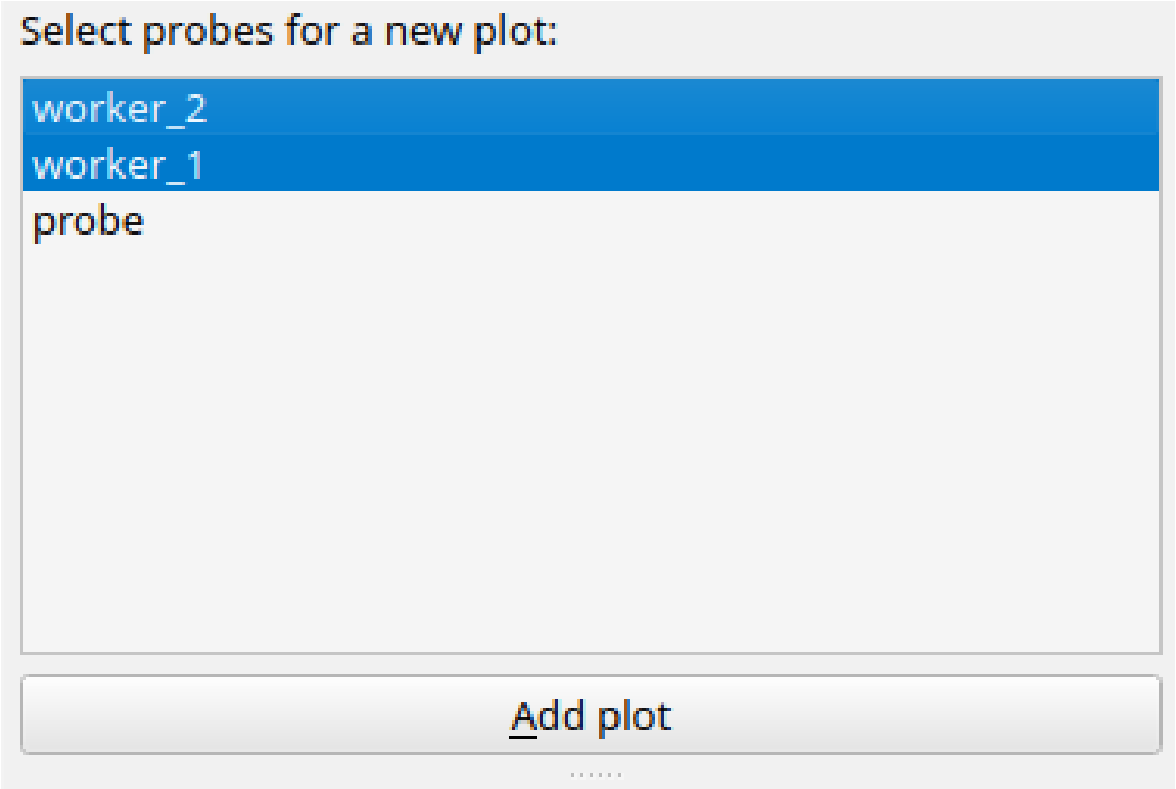

Figure D.9: Selecting the probes to add to a plot

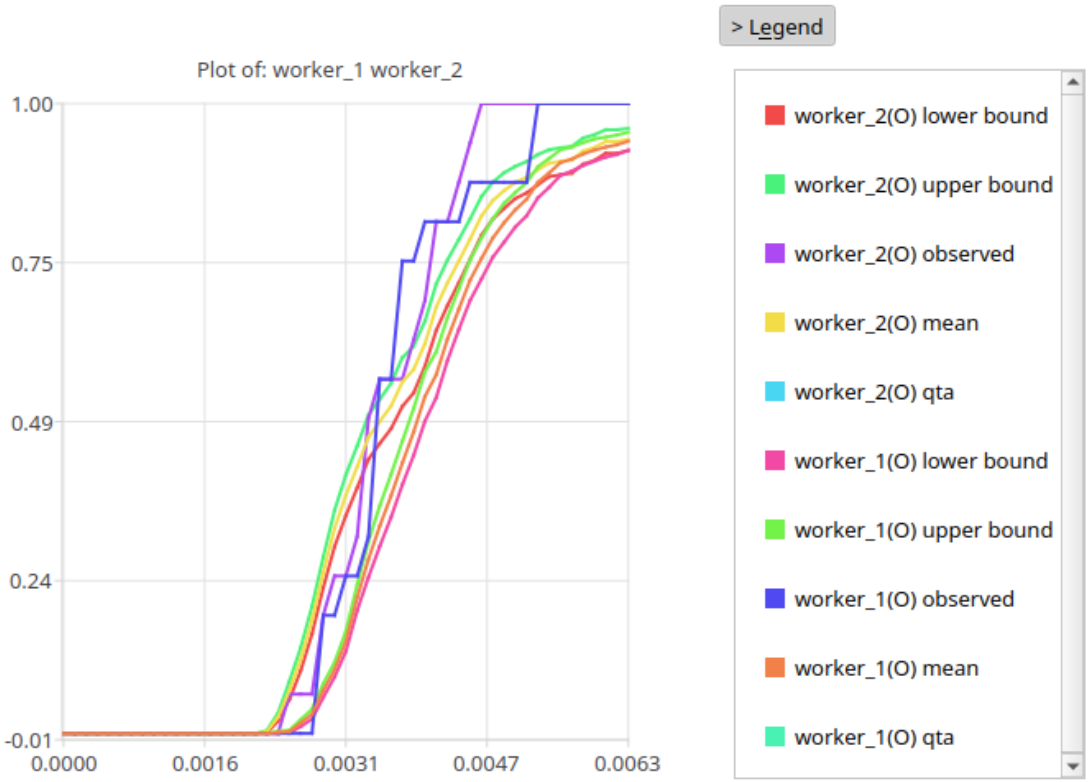

Figure D.10: After adding the probe's to a plot, if the adapter is already sending spans, the plot will show the real time ΔQs.

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 probes"** zone, by clicking **"Remove selected probes"** the components will be removed from the plot. Inversely, in the **"Available probes"** section, you can select the plots you want to add, and by clicking **"Add selected probes"** you can add the selected components to the selected plot (Fig. D.11).

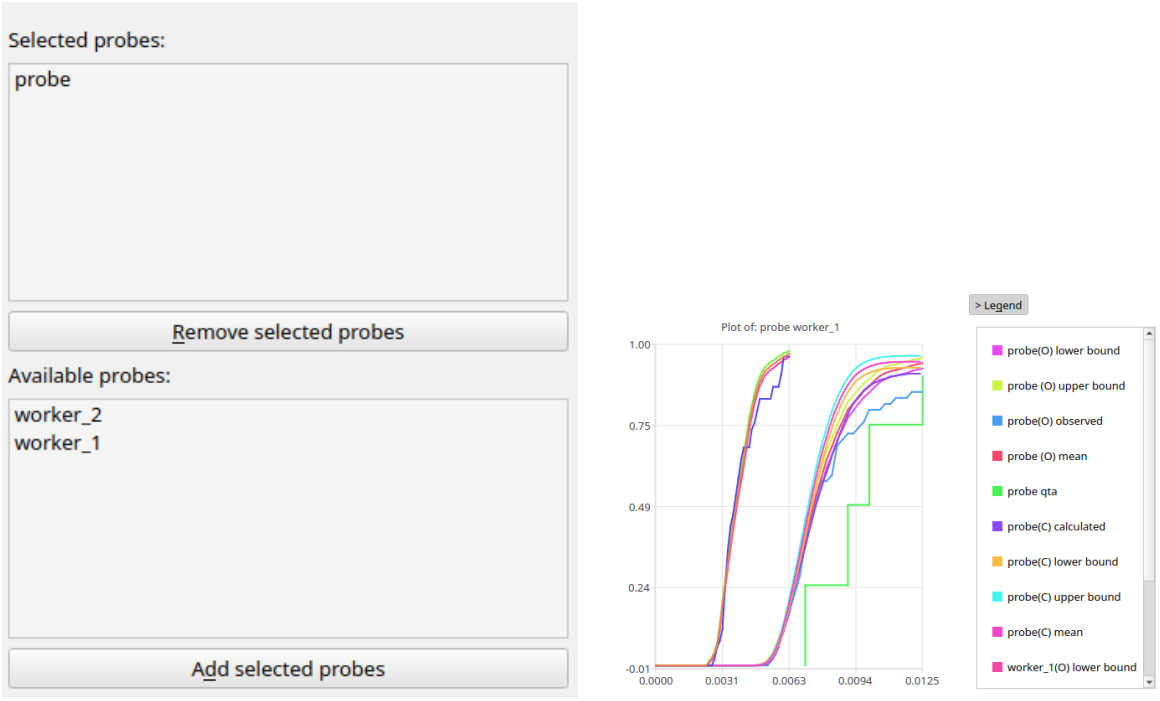

Figure D.11: To modify a plot, click on the probe you want to remove or add. Here, for example, worker_1's probe was added by selecting it below and clicking "Add selected probes".

D.2.6 Removing a plot

By left-clicking the plot, a popup appears, you can click **"Remove plot"**, this removes it from the selected plot (Fig. D.11).

D.3 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.

D.3.1 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. Fig. D.12 shows how to do this.

D.3.2 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. Fig. D.12 shows how to do this.

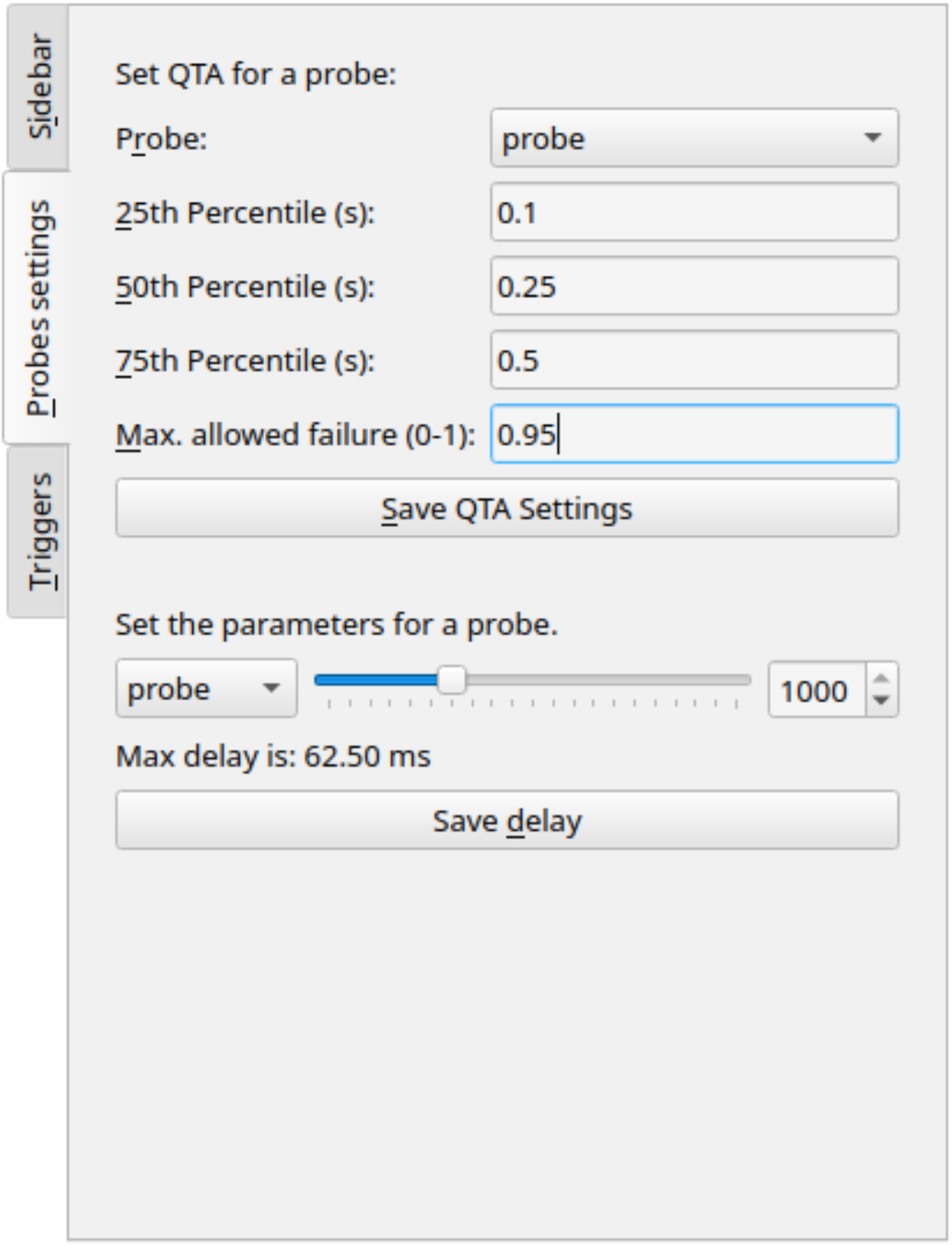

Figure D.12: Probes settings tab. Above: QTA settings. Below: Probe parameters settings.

D.4 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 ΔQs of all the probes before and after the trigger was triggered. You can discard the snapshot by left clicking on it and clicking *"Delete snapshot"*.

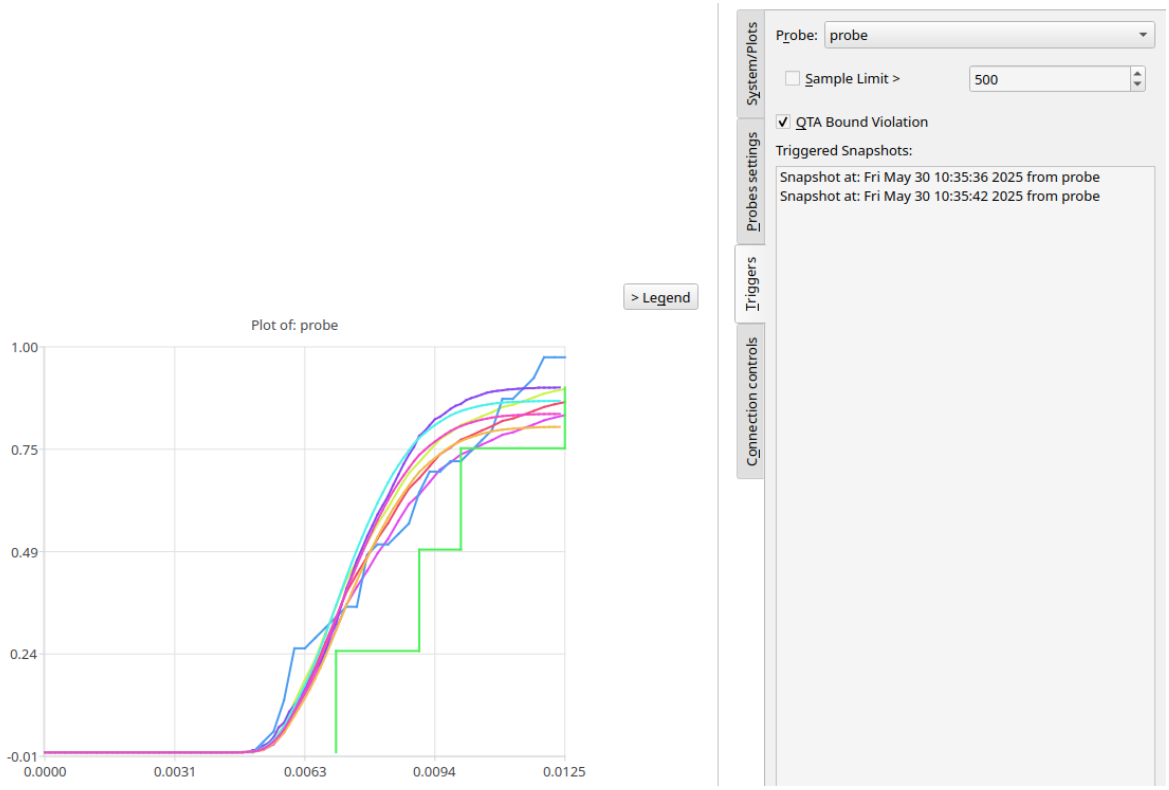

Figure D.13: Examples of the QTA being set for a probe "probe". Once a trigger is fired, the resulting snapshot from the fired trigger will be available under "Triggered snapshots".

D.5 Snapshots

The snapshots can be observed to look at the state of the system, before, during and after the trigger has been fired.

By clicking on the snapshot in the "triggered snapshots" section, a new window will popup where you can observe the system. A slider allows you to move backwards and forwards in time, observing the state of a probe in time.

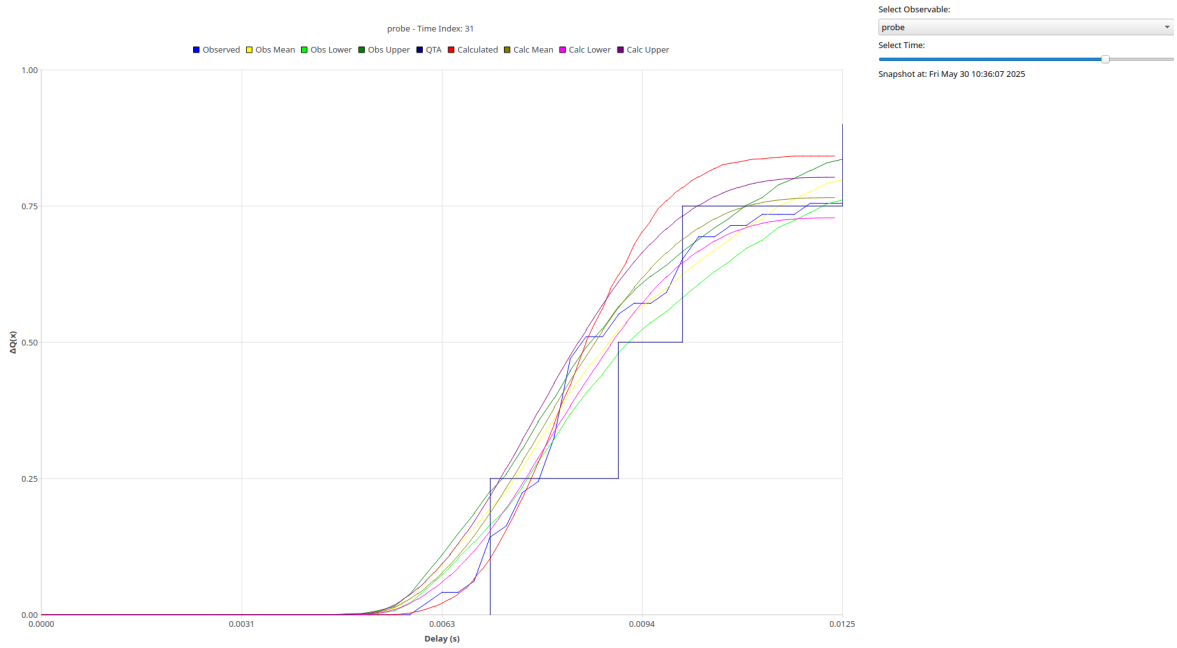

Figure D.14: Example of a snapshot, to the left, the graph at time t. To the right, the time of the graph on the left and a slider to advance backwards and forwards. As of right now, you can only select a probe at a time.

D.6 Instrumenting the Erlang application

D.6.1 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,
        ...
    dqsd_otel
    ]},
    ...
    ]}.

    {deps, [
    {opentelemetry, "~> 1.3"},
    {opentelemetry_api, "~> 1.2"},
    {opentelemetry_exporter, "~> 1.0"},
    {dqsd_otel, {git, "https://github.com/fnieri/dqsd_otel.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 was to change name, you can still find the project in https://github.com/fnieri/).

D.6.2 Starting spans

To start spans you need to call:

```
{ProbeCtx, Pid} = dqsd_otel:start_span(<<"probe">>, #{attributes =>
↪  [{attr, <<"my_attr_5o5s10">>}]})
or without attributes
{ProbeCtx, Pid} = dqsd_otel:start_span(<<"probe">>)
```

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.

D.6.3 With spans

To start with spans you need to call:

```
dqsd_otel:with_span(<<"worker_2">>, fun() -> ok, #{attributes =>
↪  [{attr, <<"my_attr">>}]})
or without attributes
dqsd_otel:with_span(<<"worker_2">>, fun() -> ok end)
```

The second attribute is the function you want to evaluate.

D.6.4 Ending spans

To end spans you started with `dqsd_otel:start_span`, you need to call:

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

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

D.6.5 Failing outcome instances

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

```
dqsd_otel: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.

Part III

Source Code Appendix

Appendix E

Grammar

This is the grammar for the ANTLR4 parser.

```
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;

// Parser Rules
start: definition* system? EOF;

definition: IDENTIFIER '=' component_chain ';';
system: 'system' '=' 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;
```

Appendix F

C++ Source Files

F.1 Root folder

F.1.1 main.cpp

This is the main file of the project.

```
1
2  #include "Application.h"
3  #include "dashboard/MainWindow.h"
4  #include "diagram/System.h"
5  #include "server/Server.h"
6  #include <QApplication>
7  #include <cstring>
8
9  #include <QApplication>
10 #include <QPalette>
11 #include <QStyleFactory>
12 #include <signal.h>
13 void setLightMode(QApplication &app)
14 {
15     // Taken from https://forum.qt.io/topic/104406/styling-the-
       qmdiarea-using-the-the-fusion-style-and-qpalette-dark-theme
16     // Use the Fusion style (consistent across platforms)
17     app.setStyle(QStyleFactory::create("Fusion"));
18
19     QPalette lightPalette;
20
21     lightPalette.setColor(QPalette::Window, QColor(255, 255, 255));
22     lightPalette.setColor(QPalette::WindowText, Qt::black);
23     lightPalette.setColor(QPalette::Base, QColor(245, 245, 245));
24     lightPalette.setColor(QPalette::AlternateBase, QColor(240, 240,
       240));
25     lightPalette.setColor(QPalette::ToolTipBase, Qt::white);
26     lightPalette.setColor(QPalette::ToolTipText, Qt::black);
27     lightPalette.setColor(QPalette::Text, Qt::black);
28     lightPalette.setColor(QPalette::Button, QColor(230, 230, 230));
29     lightPalette.setColor(QPalette::ButtonText, Qt::black);
30     lightPalette.setColor(QPalette::BrightText, Qt::red);
```

```
    lightPalette.setColor(QPalette::Link, QColor(0, 122, 204));

    lightPalette.setColor(QPalette::Highlight, QColor(0, 122, 204));
    lightPalette.setColor(QPalette::HighlightedText, Qt::white);

    app.setPalette(lightPalette);
}

int main(int argc, char *argv[])
{

    Server server(8080);
    signal(SIGPIPE, SIG_IGN); // Ignore SIGPIPE so when Erlang closes
    socket it will not crash
    Application &application = Application::getInstance();
    application.setServer(&server);
    System system = System();
    application.setSystem(system);
    QApplication app(argc, argv);
    setLightMode(app);
    MainWindow window;
    window.show();

    int result = app.exec();
    server.stop();
    return result;
}
```

F.1.2 Application.cpp

This is a singleton that handles the global state of the application.

```
#include "Application.h"

Application::Application()
{
}
Application &Application::getInstance()
{
    static Application instance;
    return instance;
}

std::shared_ptr<System> Application::getSystem()
{
    return system;
}

void Application::addObserver(std::function<void()> callback)
{
    observers.push_back(callback);
}

void Application::notifyObservers()
{
```

```
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> &
    a, const std::shared_ptr<Observable> &b)
112 {
113     if (!a || !b || a->getName() != b->getName())
114         return true;
115
116     // For Probes, check causalLinks. For Operators, check type,
    probabilities, children.
117     if (auto pa = std::dynamic_pointer_cast<Probe>(a)) {
118         auto pb = std::dynamic_pointer_cast<Probe>(b);
119         if (!pb)
120             return true;
121
122         auto ca = pa->getCausalLinks();
123         auto cb = pb->getCausalLinks();
124
125         if (ca.size() != cb.size())
126             return true;
127         for (size_t i = 0; i < ca.size(); ++i) {
128             if (ca[i]->getName() != cb[i]->getName())
129                 return true;
130         }
131     }
132
```

```
133     if (auto oa = std::dynamic_pointer_cast<Operator>(a)) {
134         auto ob = std::dynamic_pointer_cast<Operator>(b);
135         if (!ob)
136             return true;
137 
138         if (oa->getType() != ob->getType())
139             return true;
140 
141         auto ca = oa->getCausalLinks();
142         auto cb = ob->getCausalLinks();
143 
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 
153             } else {
154                 return true;
155             }
156         }
157         auto probsA = oa->getProbabilities();
158         auto probsB = ob->getProbabilities();
159         if (probsA != probsB)
160             return true;
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
    ().at(name)->getCausalLinks());
193         system->getObservables()[name] = newSystem.getProbes().at(name
    );
194     }
195     for (const auto &name : diff.addedProbes) {
196         system->getProbes()[name] = newSystem.getProbes().at(name);
197         system->getObservables()[name] = newSystem.getProbes().at(name
    );
198     }
199     for (const auto &name : diff.changedOperators) {
200         system->getOperators()[name] = newSystem.getOperators().at(
    name);
201         system->getObservables()[name] = newSystem.getOperators().at(
    name);
202     }
203     for (const auto &name : diff.addedOperators) {
204         system->getOperators()[name] = newSystem.getOperators().at(
    name);
205         system->getObservables()[name] = newSystem.getOperators().at(
    name);
206     }
207
208     for (const auto &name : diff.changedOutcomes)
209         system->getOutcomes()[name] = newSystem.getOutcomes().at(name)
    ;
210     for (const auto &name : diff.addedOutcomes) {
211         system->getOutcomes()[name] = newSystem.getOutcomes().at(name)
    ;
212         system->getObservables()[name] = newSystem.getOutcomes().at(
    name);
213     }
214     std::string defText = newSystem.getSystemDefinitionText();
215     system->setSystemDefinitionText(defText);
216
217     notifyObservers();
218 }
219
220 bool Application::startCppServer(const std::string &&ip, int port)
221 {
222     return server->startServer(ip, port);
223 }
224
225 void Application::stopCppServer()
226 {
227     server->stopServer();
228 }
229 void Application::setErlangEndpoint(const std::string &&ip, int port)
230 {
231     server->setErlangEndpoint(ip, port);
232 }
```

F.2 Dashboard

This folder contains the widgets that compose the dashboard.

F.2.1 ColorRegistry.cpp

This class stores all the colors for all the QtSeries, moreover, it generates HSV colors based on the algorithm taken from `https://martin.ankerl.com/2009/12/09/how-to-create-random-colors-programmatically/`.

```
#include "ColorRegistry.h"
#include <cmath>

std::unordered_map<std::string, QColor> ColorRegistry::colorMap;

QColor ColorRegistry::getColorFor(const std::string &name)
{
    if (colorMap.count(name))
        return colorMap[name];

    int index = colorMap.size();
    QColor color = generateDistinctColor(index);
    colorMap[name] = color;
    return color;
}

// Taken from
// https://martin.ankerl.com/2009/12/09/how-to-create-random-colors-programmatically/
QColor ColorRegistry::generateDistinctColor(int index)
{
    const double golden_ratio_conjugate = 137.508; // degrees
    double hue = std::fmod(index * golden_ratio_conjugate, 360.0);
    QColor color;
    color.setHsvF(hue / 360.0, 0.7, 0.95); // Saturation and Value tuned for brightness
    return color;
}
```

F.2.2 CustomLegendEntry.cpp

This class represents an entry in the plot legend.

```
#include "CustomLegendEntry.h"
#include <QHBoxLayout>
CustomLegendEntry::CustomLegendEntry(const QString &name, const QColor &color, QWidget *parent)
    : QWidget(parent)
{
    layout = new QHBoxLayout(this);
    colorBox = new QLabel;
    colorBox->setFixedSize(12, 12);
    colorBox->setStyleSheet(QString("background-color: %1").arg(color.name()));
```

```
    nameLabel = new QLabel(name);
    layout->addWidget(colorBox);
    layout->addWidget(nameLabel);
    layout->addStretch();
}
```

F.2.3 CustomLegendPanel.cpp

This widget represents the legend panel for a plot, with entries corresponding to each series.

```
#include "CustomLegendPanel.h"
#include "CustomLegendEntry.h"

CustomLegendPanel::CustomLegendPanel(QWidget *parent)
    : QWidget(parent)
{
    // Initialize scroll area and configure its behavior
    scrollArea = new QScrollArea(this);
    scrollArea->setWidgetResizable(true);
    scrollArea->setVerticalScrollBarPolicy(Qt::ScrollBarAsNeeded);
    scrollArea->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

    // Create content widget for the scroll area
    scrollContent = new QWidget();
    scrollArea->setWidget(scrollContent);

    // Set up layout for legend entries (aligned to top)
    legendLayout = new QVBoxLayout(scrollContent);
    legendLayout->setAlignment(Qt::AlignTop);

    // Set up main layout and add scroll area
    mainLayout = new QVBoxLayout(this);
    mainLayout->addWidget(scrollArea);
    setLayout(mainLayout);
}

/**
 * @brief Adds a new entry to the legend panel.
 * @param name The name for the entry.
 * @param color The color for the entry.
 */
void CustomLegendPanel::addEntry(const QString &name, const QColor &
    color)
{
    auto entry = new CustomLegendEntry(name, color, this);
    legendLayout->addWidget(entry);
    legendEntries[name] = entry;
}

/**
 * @brief Removes a specific entry from the legend by name.
 * @param name The name of the entry to remove.
 */
void CustomLegendPanel::removeEntry(const QString &name)
```

```
{
    if (legendEntries.count(name)) {
        QWidget *entry = legendEntries[name];
        legendLayout->removeWidget(entry);
        entry->deleteLater();
        legendEntries.erase(name);
    }
}

/**
 * @brief Clears all entries from the legend panel.
 */
void CustomLegendPanel::clear()
{
    for (auto it = legendEntries.begin(); it != legendEntries.end();)
    {
        QWidget *entry = it->second;
        legendLayout->removeWidget(entry);
        entry->deleteLater();
        it = legendEntries.erase(it);
    }
}
```

F.2.4 DQPlotController.cpp

This class is the controller of DeltaQPlot, based on the MVC design pattern.

```

#include "DQPlotController.h"
#include "../Application.h"
#include <QMetaObject>
#include <QVector>
#include <QtConcurrent>
#include <algorithm>
#include <cstdlib>
#include <qcontainerfwd.h>
#include <qlineseries.h>
using namespace std::chrono;
DQPlotController::DQPlotController(DeltaQPlot *plot, const std::vector
    <std::string> &selectedItems)
    : plot(plot)
{
    auto system = Application::getInstance().getSystem();
    for (const auto &name : selectedItems) {
        addComponent(name, (system->hasOutcome(name)));
    }
    setTitle();
}

DQPlotController::~DQPlotController()
{
    outcomes.clear();
    probes.clear();
    operators.clear();
}
```

```
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(
    name) != probes.end()) || (operators.find(name) != operators.end()
    ));
66 }
67
68 void DQPlotController::setTitle()
69 {
70     std::vector<std::string> existingItems = getComponents();
71     std::string title = "Plot of: ";
72     for (const auto &name : existingItems) {
73         title += name + " ";
74     }
75     plot->setTitle(QString::fromStdString(title));
76 }
77
78 void DQPlotController::editPlot(const std::vector<std::string> &
    selectedItems)
79 {
80     std::vector<std::string> existingItems = getComponents();
```

```
81
82      for (const auto &name : existingItems) {
83          if (std::find(selectedItems.begin(), selectedItems.end(), name
    ) == selectedItems.end()) {
84              removeComponent(name);
85          }
86      }
87      auto system = Application::getInstance().getSystem();
88
89      for (const auto &name : selectedItems) {
90          if (!containsComponent(name)) {
91              addComponent(name, system->hasOutcome(name));
92          }
93      }
94      setTitle();
95  }
96
97  void DQPlotController::addComponent(const std::string &name, bool
    isOutcome)
98  {
99      auto system = Application::getInstance().getSystem();
100     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.
    size());
111
112     for (const auto &kv : probes) {
113         components.push_back(kv.first);
114     }
115     for (const auto &kv : outcomes) {
116         components.push_back(kv.first);
117     }
118     for (const auto &kv : operators) {
119         components.push_back(kv.first);
120     }
121     return components;
122 }
123
124 QLineSeries *DQPlotController::createAndAddLineSeries(const std::
    string &legendName)
125 {
126     auto series = new QLineSeries();
127     plot->addSeries(series, legendName);
128     return series;
129 }
130
131 void DQPlotController::addOutcomeSeries(const std::string &name)
132 {
```

```
133     auto system = Application::getInstance().getSystem();
134
135     auto lowerBoundSeries = createAndAddLineSeries(name + "(O) lower
    bound");
136     auto upperBoundSeries = createAndAddLineSeries(name + "(O) upper
    bound");
137     auto outcomeSeries = createAndAddLineSeries(name + "(O) observed")
    ;
138     auto meanSeries = createAndAddLineSeries(name + "(O) mean");
139     auto qtaSeries = createAndAddLineSeries(name + "(O) qta");
140
141     OutcomeSeries series
142         = {.outcomeS = outcomeSeries, .lowerBoundS = lowerBoundSeries,
    .upperBoundS = upperBoundSeries, .meanS = meanSeries, .qtaS =
    qtaSeries};
143
144     outcomes[name] = {series, system->getOutcome(name)};
145 }
146
147 void DQPlotController::addExpressionSeries(const std::string &name,
    bool isProbe)
148 {
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,
        bool isProbe)
208 {
209     ExpressionSeries series;
210     if (isProbe)
211         series = probes[name].first;
212     else
213         series = operators[name].first;
214
215     plot->removeSeries(series.obsLowerBoundS);
216     delete series.obsLowerBoundS;
217     series.obsLowerBoundS = NULL;
218
219     plot->removeSeries(series.obsUpperBoundS);
220     delete series.obsUpperBoundS;
221     series.obsUpperBoundS = NULL;
222
223     plot->removeSeries(series.obsS);
224     delete series.obsS;
225     series.obsS = NULL;
226
227     plot->removeSeries(series.obsMeanS);
228     delete series.obsMeanS;
229     series.obsMeanS = NULL;
230
231     plot->removeSeries(series.qtaS);
232     delete series.qtaS;
233     series.qtaS = NULL;
```

```
234
235     plot->removeSeries(series.calcS);
236     delete series.calcS;
237     series.calcS = NULL;
238
239     plot->removeSeries(series.calcLowerBoundS);
240     delete series.calcLowerBoundS;
241     series.calcLowerBoundS = NULL;
242
243     plot->removeSeries(series.calcUpperBoundS);
244     delete series.calcUpperBoundS;
245     series.calcUpperBoundS = NULL;
246
247     plot->removeSeries(series.calcMeanS);
248     delete series.calcMeanS;
249     series.calcMeanS = NULL;
250
251     if (isProbe)
252         probes.erase(name);
253     else
254         operators.erase(name);
255 }
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                 plot->updateSeries(series.meanS, meanData);
359                 plot->updateSeries(series.qtaS, qtaData);
360 
361                 plot->setUpdatesEnabled(true);
362 
363                 //     auto guiEnd = high_resolution_clock::now();
364                 //       qDebug() << "GUI update took" << duration_cast
    <microseconds>(guiEnd - guiStart).count() << "";
365             },
366             Qt::QueuedConnection);
367     });
368     ret.waitForFinished();
369     return outcome->getMaxDelay();
370 }
371 
372 double DQPlotController::updateOperator(ExpressionSeries &series, std
    ::shared_ptr<Operator> &op, uint64_t timeLowerBound, uint64_t
    timeUpperBound)
373 {
374     updateExpression(series, op->getObservedDeltaQRepr(timeLowerBound,
     timeUpperBound), op->getCalculatedDeltaQRepr(timeLowerBound,
    timeUpperBound),
375         op->getQTA(), op->getMaxDelay());
376 
377     return op->getMaxDelay();
378 }
379 
380 double DQPlotController::updateProbe(ExpressionSeries &series, std::
    shared_ptr<Probe> &probe, uint64_t timeLowerBound, uint64_t
    timeUpperBound)
381 {
382     updateExpression(series, probe->getObservedDeltaQRepr(
```

```
    timeLowerBound, timeUpperBound), probe->getCalculatedDeltaQRepr(
    timeLowerBound, timeUpperBound),
383         probe->getQTA(), probe->getMaxDelay());
384     return probe->getMaxDelay();
385 }
386
387 void DQPlotController::updateExpression(ExpressionSeries &series,
    DeltaQRepr &&obsRepr, DeltaQRepr &&calcRepr, QTA &&qta, double
    maxDelay)
388 {
389     auto ret = QtConcurrent::run([=]() {
390         auto computeStart = high_resolution_clock::now();
391
392         DeltaQ obsDeltaQ = obsRepr.deltaQ;
393         std::vector<Bound> obsBounds = obsRepr.bounds;
394         DeltaQ calcDeltaQ = calcRepr.deltaQ;
395         std::vector<Bound> calcBounds = calcRepr.bounds;
396         int observedBins = obsDeltaQ.getBins();
397         double observedBinWidth = obsDeltaQ.getBinWidth();
398
399         // --- Prepare data ---
400         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             obsLowerBoundData.emplace_back(QPointF(x, obsBounds[i].
    lowerBound));
420             obsUpperBoundData.emplace_back(QPointF(x, obsBounds[i].
    upperBound));
421             obsMeanData.emplace_back(QPointF(x, obsBounds[i].mean));
422         }
423
424         QVector<QPointF> calcDeltaQData;
425         calcDeltaQData.emplace_back(QPointF(0, 0));
426         calcDeltaQData.reserve(observedBins);
427
428         QVector<QPointF> calcLowerBoundData;
429         calcLowerBoundData.emplace_back(QPointF(0, 0));
430         calcLowerBoundData.reserve(observedBins);
431
```

```
432         QVector<QPointF> calcUpperBoundData;
433         calcUpperBoundData.emplace_back(QPointF(0, 0));
434         calcUpperBoundData.reserve(observedBins);
435 
436         QVector<QPointF> calcMeanData;
437         calcMeanData.emplace_back(QPointF(0, 0));
438         calcMeanData.reserve(observedBins);
439 
440         QVector<QPointF> qtaData;
441 
442         // Prepare calculatedDeltaQ data
443         int calculatedBins = calcDeltaQ.getBins();
444         double calculatedBinWidth = calcDeltaQ.getBinWidth();
445         for (int i = 0; i < calculatedBins; ++i) {
446             double x = calculatedBinWidth * (i + 1);
447             calcDeltaQData.emplace_back(QPointF(x, calcDeltaQ.cdfAt(i)
    ));
448             calcLowerBoundData.emplace_back(QPointF(x, calcBounds[i].
    lowerBound));
449             calcUpperBoundData.emplace_back(QPointF(x, calcBounds[i].
    upperBound));
450             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 }
```

F.2.5 DQPlotList.cpp

This widget handles the adding and removing of probes series from a plot.

```
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  }
```

F.2.6 DelaySettingsWidget.cpp

This widget represent the slider widget to modify the parameters $dMax$, Δt and N.

```
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);
```

```cpp
    connect(binSpinBox, QOverload<int>::of(&QSpinBox::valueChanged), this, &DelaySettingsWidget::updateMaxDelay);
    connect(saveDelayButton, &QPushButton::clicked, this, &DelaySettingsWidget::onSaveDelayClicked);
    connect(observableComboBox, &QComboBox::currentTextChanged, this, &DelaySettingsWidget::loadObservableSettings);

    Application::getInstance().addObserver([this]() { this->populateComboBox(); });
}

/**
 * @brief Populates the combo box with available observables from the system.
 */
void DelaySettingsWidget::populateComboBox()
{
    auto system = Application::getInstance().getSystem();
    if (!system)
        return;

    observableComboBox->clear();
    for (const auto &[name, obs] : system->getObservables()) {
        if (obs)
            observableComboBox->addItem(QString::fromStdString(name));
    }
}

/**
 * @brief Loads delay settings for the currently selected observable.
 */
void DelaySettingsWidget::loadObservableSettings()
{
    auto system = Application::getInstance().getSystem();
    if (!system)
        return;

    QString observableName = observableComboBox->currentText();
    if (observableName.isEmpty())
        return;

    auto observable = system->getObservable(observableName.toStdString());

    auto exponent = observable->getDeltaTExp();
    auto bins = observable->getNBins();
    delaySlider->setValue(exponent);
    binSpinBox->setValue(bins);

    updateMaxDelay();
}

/**
 * @brief Computes the current maximum delay.
 * @return Maximum delay value based on bin count and delay exponent.
 */
```

```cpp
double DelaySettingsWidget::getMaxDelayMs() const
{
    int exponent = delaySlider->value();
    int bins = binSpinBox->value();
    return 1.0 * std::pow(2.0, exponent) * bins;
}

/**
 * @brief Updates the label that shows the computed max delay value.
 */
void DelaySettingsWidget::updateMaxDelay()
{
    double delay = getMaxDelayMs();
    maxDelayLabel->setText(QString("Max delay is: %1 ms").arg(delay, 0, 'f', 2));
}

/**
 * @brief Saves the current delay settings to the system and emits a change signal.
 */
void DelaySettingsWidget::onSaveDelayClicked()
{
    auto system = Application::getInstance().getSystem();
    if (!system)
        return;

    QString name = observableComboBox->currentText();
    if (name.isEmpty())
        return;

    int exponent = delaySlider->value();
    int bins = binSpinBox->value();
    std::string nameString = name.toStdString();
    system->setObservableParameters(nameString, exponent, bins);

    Q_EMIT delayParametersChanged();
}
```

F.2.7 DeltaQPlot.cpp

This widget is the widget containing a plot and its legend.

```cpp

#include "DeltaQPlot.h"
#include "ColorRegistry.h"
#include "CustomLegendPanel.h"
#include "DQPlotList.h"
#include <QChartView>
#include <QDebug>
#include <QHBoxLayout>
#include <QLineSeries>
#include <QMouseEvent>
#include <QRandomGenerator>
#include <QToolButton>
```

```cpp
#include <QVBoxLayout>
DeltaQPlot::DeltaQPlot(const std::vector<std::string> &selectedItems,
    QWidget *parent)
    : QWidget(parent)
{

    // Create legend panel and toggle
    legendPanel = new CustomLegendPanel(this);
    QToolButton *toggleButton = new QToolButton(this);
    toggleButton->setText("> Legend");
    toggleButton->setCheckable(true);
    toggleButton->setChecked(true);
    toggleButton->setToolButtonStyle(Qt::ToolButtonTextBesideIcon);

    connect(toggleButton, &QToolButton::toggled, legendPanel, &QWidget
    ::setVisible);
    connect(toggleButton, &QToolButton::toggled, [toggleButton](bool
    checked) { toggleButton->setText(checked ? "^ Legend" : "> Legend"
    ); });

    // Create chart and view
    chart = new QChart();
    chartView = new QChartView(chart, this);
    chartView->setRenderHint(QPainter::Antialiasing);
    chart->legend()->setVisible(false);

    axisX = new QValueAxis();
    axisY = new QValueAxis();
    axisY->setRange(-0.01, 1.0);
    axisX->setRange(0, 0.05);
    chart->addAxis(axisX, Qt::AlignBottom);
    chart->addAxis(axisY, Qt::AlignLeft);

    controller = new DQPlotController(this, selectedItems);
    plotList = new DQPlotList(controller, this);
    // Right-side layout: toggle + legend
    QVBoxLayout *rightLayout = new QVBoxLayout();
    rightLayout->addWidget(toggleButton);
    rightLayout->addWidget(legendPanel);
    rightLayout->addStretch();

    // Main layout: chart + right side
    QHBoxLayout *mainLayout = new QHBoxLayout(this);
    mainLayout->addWidget(chartView, 1);
    mainLayout->addLayout(rightLayout);
    mainLayout->setContentsMargins(0, 0, 0, 0);
    mainLayout->setSpacing(5);
    setLayout(mainLayout);
    chartView->setRenderHint(QPainter::Antialiasing);
}

DeltaQPlot::~DeltaQPlot()
{
    delete controller;
    controller = nullptr;
    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 &
        name)
83  {
84      chart->addSeries(series);
85      series->setName(QString::fromStdString(name));
86      series->attachAxis(axisX);
87      series->attachAxis(axisY);
88      QColor color = ColorRegistry::getColorFor(name);
89      legendPanel->addEntry(QString::fromStdString(name), color);
90      series->setColor(color);
91      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 }
```

F.2.8 MainWindow.cpp

This widget is the main window of the application, it has a tab to the side where the widgets to control the oscilloscope are. To the left, the panel where all plots are shown.

```
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 sampling rate changes
77      connect(sidebar, &Sidebar::onSamplingRateChanged, this, [this](int
     ms) {
78          qDebug() << "MainWindow received sampling rate:" << ms;
79          samplingRate = 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(samplingRate)).count();
115     uint64_t timeUpperBound = timeLowerBound + std::chrono::
    duration_cast<std::chrono::nanoseconds>(std::chrono::milliseconds(
    samplingRate)).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
    components before adding a plot.");
170         return;
171     }
172 
173     // Create new plot container
174     auto *plotWidget = new QWidget(this);
175     auto *plotWidgetLayout = new QVBoxLayout(plotWidget);
176     auto *deltaQPlot = new DeltaQPlot(selectedItems, this);
177 
178     // Set up connections and tracking
179     connect(deltaQPlot, &DeltaQPlot::plotSelected, this, &MainWindow::
    onPlotSelected);
180     plotContainers[deltaQPlot] = plotWidget;
181 
182     // Configure layout
183     plotWidgetLayout->addWidget(deltaQPlot);
184     plotWidget->setMaximumWidth(scrollArea->width() / MAX_P_ROW);
185     plotWidget->setMaximumHeight(scrollArea->height() / 2);
```

```
186
187     // Position in grid
188     int plotCount = plotContainers.size();
189     int row = (plotCount - 1) / MAX_P_ROW;
190     int col = (plotCount - 1) % MAX_P_COL;
191     plotLayout->addWidget(plotWidget, row, col);
192
193     // Update UI state
194     onPlotSelected(deltaQPlot);
195     sidebar->clearOnAdd();
196 }
197
198 /**
199  * @brief Handles window resize events to adjust plot sizes.
200  * @param event The resize event.
201  */
202 void MainWindow::resizeEvent(QResizeEvent *event)
203 {
204     QMainWindow::resizeEvent(event);
205     for (auto [plot, widget] : plotContainers.asKeyValueRange()) {
206         widget->setMaximumWidth(scrollArea->width() / MAX_P_COL);
207         widget->setMaximumHeight(scrollArea->height() / 2);
208     }
209 }
210
211 /**
212  * @brief Shows context menu for plot management.
213  * @param event The context menu event.
214  */
215 void MainWindow::contextMenuEvent(QContextMenuEvent *event)
216 {
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();
    ++it) {
224         if (it.value()->isAncestorOf(child)) {
225             selectedPlot = it.key();
226             break;
227         }
228     }
229
230     if (!selectedPlot)
231         return;
232
233     // Create and show context menu
234     QMenu contextMenu(this);
235     QAction *removeAction = contextMenu.addAction("Remove Plot");
236
237     QAction *selectedAction = contextMenu.exec(event->globalPos());
238
239     if (selectedAction == removeAction) {
240         onRemovePlot(selectedPlot);
```

```
241     }
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();
    ++it) {
260         int row = plotCount / MAX_P_ROW;
261         int col = plotCount % MAX_P_COL;
262         plotLayout->addWidget(it.value(), row, col);
263         plotLayout->setRowStretch(row, 1);
264         plotLayout->setColumnStretch(col, 1);
265         plotCount++;
266     }
267
268     sidebar->hideCurrentPlot();
269 }
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 }
```

F.2.9 NewPlotList.cpp

This widget is the widget to add a new plot to the plots panel.

```
1 /**
2  * @file NewPlotList.cpp
3  * @brief Implementation of the NewPlotList class, which provides a UI
       list for selecting observables to create a new plot.
4  */
```

```cpp

#include "NewPlotList.h"
#include "../Application.h"
#include <iostream>
#include <qlistwidget.h>

NewPlotList::NewPlotList(QWidget *parent)
    : QListWidget(parent)
{
    setSelectionMode(QAbstractItemView::MultiSelection);
    Application::getInstance().addObserver([this]() { this->reset(); });
}

/**
 * @brief Clears and repopulates the list with updated observables.
 *
 */
void NewPlotList::reset()
{
    this->blockSignals(true);
    while (count() != 0)
        delete takeItem(0);
    this->blockSignals(false);
    addItems();
}

/**
 * @brief Adds all observables from the system as items in the list.
 */
void NewPlotList::addItems()
{
    auto system = Application::getInstance().getSystem();
    auto observables = system->getObservables();

    for (const auto &obs : observables) {
        if (obs.second) {
            QListWidgetItem *item = new QListWidgetItem(QString::fromStdString(obs.first), this);
            item->setFlags(Qt::ItemIsSelectable | Qt::ItemIsEnabled);
        }
    }
}

/**
 * @brief Returns a list of selected observable names.
 * @return A vector of strings corresponding to selected item labels.
 */
std::vector<std::string> NewPlotList::getSelectedItems()
{
    std::vector<std::string> selectedItems;
    QList<QListWidgetItem *> selected = this->selectedItems();

    for (QListWidgetItem *item : selected) {
        selectedItems.push_back(item->text().toStdString());
    }
```

```
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  }
```

F.2.10 ObservableSettings.cpp

This widget is a tab that contains the settings for the probes (Setting a QTA, setting $dMax$).

```
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::
        loadObservableSettings);
19  }
```

F.2.11 SamplingRateWidget.cpp

This widget allows the sampling rate to be changed via a slider.

```
1   #include "SamplingRateWidget.h"
2
3   SamplingRateWidget::SamplingRateWidget(QWidget *parent)
4       : QWidget(parent)
5   {
6       // Available sampling rate options in milliseconds
7       samplingRates = {100, 200, 300, 400, 500, 600, 700, 800, 900,
        1000, 2000, 5000, 10000};
8
9       // Configure rate selection slider
10      slider = new QSlider(Qt::Horizontal);
11      slider->setMinimum(0);
```

```
12      slider->setMaximum(samplingRates.size() - 1);
13      slider->setValue(1);
14      slider->setTickInterval(1);
15      slider->setTickPosition(QSlider::TicksBelow);
16
17      valueLabel = new QLabel(QString("Sampling Rate: %1 ms").arg(
        samplingRates[0]));
18
19      saveButton = new QPushButton("Save sampling rate");
20
21      // Set up layout
22      auto *layout = new QVBoxLayout();
23      layout->addWidget(slider);
24      layout->addWidget(valueLabel);
25      layout->addWidget(saveButton);
26      setLayout(layout);
27
28      // Connect signals
29      connect(slider, &QSlider::valueChanged, this, &SamplingRateWidget
        ::onSliderValueChanged);
30      connect(saveButton, &QPushButton::clicked, this, &
        SamplingRateWidget::onSaveClicked);
31  }
32
33  /**
34   * @brief Updates the displayed rate when slider changes.
35   * @param value The current slider index (0-based).
36   *
37   * Converts slider index to actual milliseconds and updates the
        display label.
38   */
39  void SamplingRateWidget::onSliderValueChanged(int value)
40  {
41      int ms = samplingRates[value];
42      valueLabel->setText(QString("Sampling Rate: %1 ms").arg(ms));
43  }
44
45  /**
46   * @brief Emits the selected sampling rate when save is clicked.
47   *
48   * Gets the current slider value, converts it to milliseconds,
49   * and emits the onSamplingRateChanged signal.
50   */
51  void SamplingRateWidget::onSaveClicked()
52  {
53      int ms = samplingRates[slider->value()];
54      Q_EMIT onSamplingRateChanged(ms);
55  }
```

F.2.12 QTAInputWidget.cpp

This widget allows the QTA to be set for a probe

```
1  #include "QTAInputWidget.h"
2  #include "../Application.h"
```

```
#include <QDoubleValidator>
#include <QMessageBox>
#include <QPushButton>

QTAInputWidget::QTAInputWidget(QWidget *parent)
    : QWidget(parent)
{
    auto layout = new QFormLayout(this);

    qtaLabel = new QLabel(this);
    qtaLabel->setText("Set QTA for a probe:");
    layout->addRow(qtaLabel);
    observableComboBox = new QComboBox(this);
    layout->addRow("Probe:", observableComboBox);

    perc25Edit = new QLineEdit(this);
    perc25Edit->setPlaceholderText("Seconds (s)");

    perc50Edit = new QLineEdit(this);
    perc50Edit->setPlaceholderText("Seconds (s)");

    perc75Edit = new QLineEdit(this);
    perc75Edit->setPlaceholderText("Seconds (s)");

    cdfMaxEdit = new QLineEdit(this);
    cdfMaxEdit->setPlaceholderText("Value between 0 and 1");

    layout->addRow("25th Percentile (s):", perc25Edit);
    layout->addRow("50th Percentile (s):", perc50Edit);
    layout->addRow("75th Percentile (s):", perc75Edit);
    layout->addRow("Max. allowed failure (0-1):", cdfMaxEdit);

    connect(observableComboBox, &QComboBox::currentTextChanged, this,
   &QTAInputWidget::loadObservableSettings);

    Application::getInstance().addObserver([this]() { this->
   populateComboBox(); });
    saveButton = new QPushButton("Save QTA Settings", this);
    layout->addRow(saveButton);
    connect(saveButton, &QPushButton::clicked, this, &QTAInputWidget::
   onSaveButtonClicked);

    setLayout(layout);

    Application::getInstance().addObserver([this]() { this->
   populateComboBox(); });
}

void QTAInputWidget::populateComboBox()
{
    auto system = Application::getInstance().getSystem();
    if (!system)
        return;

    observableComboBox->clear();
    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()
    , getCdfMax(), true);
91          observable->setQTA(newQTA);
92      } catch (std::exception &e) {
93          QMessageBox::warning(this, "Error", e.what());
94      }
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 }
```

F.2.13 Sidebar.cpp

This widget is a tab where the user can handle the system, add/remove plots and change the sampling rate.

```
1  #include "Sidebar.h"
2
3  #include "NewPlotList.h"
4  #include "SamplingRateWidget.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      // Sampling rate section
50      samplingRateWidget = new SamplingRateWidget(this);
51      mainSplitter->addWidget(samplingRateWidget);
52
53      connect(samplingRateWidget, &SamplingRateWidget::
        onSamplingRateChanged, this, &Sidebar::handleSamplingRateChanged);
54
55      // Current plot section (initially hidden)
56      currentPlotWidget = new QWidget(this);
57      currentPlotLayout = new QVBoxLayout(currentPlotWidget);
58      currentPlotLabel = new QLabel("Modify current plot:", this);
59      currentPlotLabel->setSizePolicy(QSizePolicy::Preferred,
        QSizePolicy::Fixed);
60      currentPlotLabel->hide();
61
62      currentPlotLayout->addWidget(currentPlotLabel);
63      mainSplitter->addWidget(currentPlotWidget);
64
65      layout->addWidget(mainSplitter);
66  }
67
68  /**
69   * @brief Sets the current plot list widget to display.
70   * @param plotList The DQPlotList widget to show in the current plot
       section.
71   */
72  void Sidebar::setCurrentPlotList(DQPlotList *plotList)
73  {
74      if (currentPlotList == plotList) {
75          return;
76      }
77
78      if (currentPlotList) {
79          layout->removeWidget(currentPlotList);
80          currentPlotList->hide();
81      }
82
83      if (plotList) {
84          currentPlotList = plotList;
85          layout->addWidget(currentPlotList);
86          currentPlotList->show();
87          currentPlotLabel->show();
88      }
```

```
89 }
90
91 /**
92  * @brief Handles sampling rate change events from the
      SamplingRateWidget.
93  * @param ms The new sampling rate in milliseconds.
94  */
95 void Sidebar::handleSamplingRateChanged(int ms)
96 {
97     Q_EMIT onSamplingRateChanged(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
      probes before adding a plot.");
130         return;
131     }
132
133     Q_EMIT addPlotClicked();
134 }
```

F.2.14 SnapshotViewerWindow.cpp

This is a window to observe a snapshot from the triggers tab.

```
1 /**
2  * @file SnapshotViewerWindow.cpp
```

```
 * @brief Implementation of the SnapshotViewerWindow class.
 */

#include "SnapshotViewerWindow.h"

#include <QComboBox>
#include <QHBoxLayout>
#include <QLabel>
#include <QSlider>
#include <QVBoxLayout>
#include <QtCharts/QChartView>
#include <QtCharts/QLineSeries>
#include <QtCharts/QValueAxis>
#include <QtConcurrent>
#include <qnamespace.h>

/**
 * @brief Constructs a new SnapshotViewerWindow.
 */
SnapshotViewerWindow::SnapshotViewerWindow(std::vector<Snapshot> &
    snapshotList, QWidget *parent)
    : QWidget(parent)
    , observableSelector(new QComboBox(this))
    , timeSlider(new QSlider(Qt::Horizontal, this))
    , timeLabel(new QLabel(this))
    , chartView(new QChartView(new QChart(), this))
{
    auto *mainLayout = new QHBoxLayout(this);

    // Chart area
    chartView->setRenderHint(QPainter::Antialiasing);
    mainLayout->addWidget(chartView, 3); // 3/4 of space

    // Controls
    auto *controlLayout = new QVBoxLayout(this);
    controlLayout->addWidget(new QLabel("Select Observable:"));
    controlLayout->addWidget(observableSelector);

    controlLayout->addWidget(new QLabel("Select Time:"));
    controlLayout->addWidget(timeSlider);
    controlLayout->addWidget(timeLabel);
    controlLayout->addStretch();
    mainLayout->addLayout(controlLayout, 1); // 1/4 of space

    connect(observableSelector, &QComboBox::currentTextChanged, this,
    &SnapshotViewerWindow::onObservableChanged);
    connect(timeSlider, &QSlider::valueChanged, this, &
    SnapshotViewerWindow::onTimeSliderChanged);

    setSnapshots(snapshotList);
}

/**
 * @brief Sets the snapshot data and updates the UI.
 */
void SnapshotViewerWindow::setSnapshots(std::vector<Snapshot> &
```

```
    snapshotList)
{
    snapshots.clear();
    observableSelector->clear();

    for (auto &s : snapshotList) {
        snapshots[s.getName()] = s;
        observableSelector->addItem(QString::fromStdString(s.getName()
    ));
    }

    if (!snapshots.empty()) {
        observableSelector->setCurrentIndex(0);
        onObservableChanged(observableSelector->currentText());
    }
}

/**
 * @brief Handles the change of observable by updating the slider
    range and triggering a plot update.
 */
void SnapshotViewerWindow::onObservableChanged(const QString &name)
{
    currentObservable = name.toStdString();

    const auto &snapshot = snapshots.at(currentObservable);
    int count = static_cast<int>(snapshot.getObservedSize());
    timeSlider->setRange(0, std::max(0, count - 1));
    timeSlider->setValue(0);
    onTimeSliderChanged(0);
}

/**
 * @brief Handles changes in the time slider by updating the chart.
 */
void SnapshotViewerWindow::onTimeSliderChanged(int value)
{
    if (snapshots.empty() || !snapshots.count(currentObservable))
        return;

    const auto &snapshot = snapshots.at(currentObservable);

    if (snapshot.getObservedSize() == 0 || value >= static_cast<int>(
    snapshot.getObservedSize()))
        return;

    const auto obs = snapshot.getObservedDeltaQs()[value];
    const auto calc = snapshot.getCalculatedSize() > value ? std::
    optional<DeltaQRepr>(snapshot.getCalculatedDeltaQs()[value]) : std
    ::nullopt;
    const auto qta = snapshot.getQTAs()[value];
    auto time = obs.time;

    qint64 msTime = time / 1000000;
    QDateTime timestamp = QDateTime::fromMSecsSinceEpoch(msTime);
    timeLabel->setText(QString("Snapshot at: %1").arg(timestamp.
```

```
    toString()));

    auto ret = QtConcurrent::run([=]() {
        int bins = obs.deltaQ.getBins();
        double binWidth = obs.deltaQ.getBinWidth();

        QVector<QPointF> obsMean, obsLower, obsUpper, obsCdf, qtaData;
        obsCdf.reserve(bins + 1);
        obsMean.reserve(bins + 1);
        obsLower.reserve(bins + 1);
        obsUpper.reserve(bins + 1);

        obsCdf.append(QPointF(0, 0));
        obsMean.append(QPointF(0, 0));
        obsLower.append(QPointF(0, 0));
        obsUpper.append(QPointF(0, 0));

        for (int i = 0; i < bins; ++i) {
            double x = binWidth * (i + 1);
            obsCdf.append(QPointF(x, obs.deltaQ.cdfAt(i)));
            obsLower.append(QPointF(x, obs.bounds[i].lowerBound));
            obsUpper.append(QPointF(x, obs.bounds[i].upperBound));
            obsMean.append(QPointF(x, obs.bounds[i].mean));
        }

        if (qta.defined) {
            double maxDelay = bins * binWidth;
            qtaData = {
                {qta.perc_25, 0          },
                {qta.perc_25, 0.25       },
                {qta.perc_50, 0.25       },
                {qta.perc_50, 0.5        },
                {qta.perc_75, 0.5        },
                {qta.perc_75, 0.75       },
                {maxDelay,    0.75       },
                {maxDelay,    qta.cdfMax}
            };
        }

        QVector<QPointF> calcCdf, calcMean, calcLower, calcUpper;
        if (calc) {
            int cbins = calc->deltaQ.getBins();
            double cbinWidth = calc->deltaQ.getBinWidth();
            calcCdf.reserve(cbins + 1);
            calcMean.reserve(cbins + 1);
            calcLower.reserve(cbins + 1);
            calcUpper.reserve(cbins + 1);

            calcCdf.append(QPointF(0, 0));
            calcMean.append(QPointF(0, 0));
            calcLower.append(QPointF(0, 0));
            calcUpper.append(QPointF(0, 0));

            for (int i = 0; i < cbins; ++i) {
                double x = cbinWidth * (i + 1);
                calcCdf.append(QPointF(x, calc->deltaQ.cdfAt(i)));
```

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

```
                series->attachAxis(axisY);
            }
        });
    });
}
```

F.2.15 StubControlWidget.cpp

This widget allows to open the server on the IP and Port defined by the user and to connect to the adapter on the IP and port specified by the user.

```
#include "StubControlWidget.h"

StubControlWidget::StubControlWidget(QWidget *parent)
    : QWidget(parent)
{
    mainLayout = new QVBoxLayout(this);

    // Erlang Control Group
    QGroupBox *erlangGroup = new QGroupBox("Erlang Wrapper Control",
    this);
    QVBoxLayout *erlangMainLayout = new QVBoxLayout(erlangGroup);

    // First row: IP and Port
    QHBoxLayout *erlangIpPortLayout = new QHBoxLayout();
    erlangIpPortLayout->addWidget(new QLabel("IP:"));
    erlangReceiverIpEdit = new QLineEdit("127.0.0.1", this);
    erlangIpPortLayout->addWidget(erlangReceiverIpEdit);
    erlangIpPortLayout->addWidget(new QLabel("Port:"));
    erlangReceiverPortEdit = new QLineEdit("8081", this);
    erlangIpPortLayout->addWidget(erlangReceiverPortEdit);
    erlangMainLayout->addLayout(erlangIpPortLayout);

    // Second row: Set Endpoint Button
    setErlangEndpointButton = new QPushButton("Set Adapter Endpoint",
    this);
    erlangMainLayout->addWidget(setErlangEndpointButton, 0, Qt::
    AlignLeft);

    // Third row: Start/Stop Wrapper
    QHBoxLayout *erlangButtonsLayout = new QHBoxLayout();
    stopErlangButton = new QPushButton("Stop Adapter", this);
    startErlangButton = new QPushButton("Start Adapter", this);
    erlangButtonsLayout->addWidget(stopErlangButton);
    erlangButtonsLayout->addWidget(startErlangButton);
    erlangMainLayout->addLayout(erlangButtonsLayout);

    // Server Control Group
    QGroupBox *serverGroup = new QGroupBox("C++ Server Control", this)
    ;
    QVBoxLayout *serverMainLayout = new QVBoxLayout(serverGroup);

    // First row: IP and Port
    QHBoxLayout *serverIpPortLayout = new QHBoxLayout();
    serverIpPortLayout->addWidget(new QLabel("IP:"));
```

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

```
89      }
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(),
        port);
105 }
```

F.2.16 SystemCreationWidget.cpp

This widget allows the creation/update of a system, loading an already existing one or saving one.

```
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 }
```

F.2.17 TriggersTab.cpp

This tab holds the widgets to set/remove triggers and view fired ones.

```cpp
#include "TriggersTab.h"
#include "SnapshotViewerWindow.h"
#include <QDateTime>
#include <QLabel>
#include <QString>
#include <QTimer>
#include <cstdlib>
#include <sstream>
#include <string>
#include <unordered_set>

TriggersTab::TriggersTab(QWidget *parent)
    : QWidget(parent)
{
    // Main layout setup
    mainLayout = new QVBoxLayout(this);
    formLayout = new QFormLayout();

    // Observable selection dropdown
    observableComboBox = new QComboBox(this);
    connect(observableComboBox, &QComboBox::currentTextChanged, this,
   &TriggersTab::onObservableChanged);
    formLayout->addRow("Probe:", observableComboBox);

    // Sample limit controls
    sampleLimitCheckBox = new QCheckBox("Sample Limit >", this);
    sampleLimitSpinBox = new QSpinBox(this);
    sampleLimitSpinBox->setRange(1, 100000);
    sampleLimitSpinBox->setValue(sampleLimitThreshold);

    sampleLimitLayout = new QHBoxLayout();
    sampleLimitLayout->addWidget(sampleLimitCheckBox);
    sampleLimitLayout->addWidget(sampleLimitSpinBox);

    sampleLimitWidget = new QWidget(this);
    sampleLimitWidget->setLayout(sampleLimitLayout);
    formLayout->addRow(sampleLimitWidget);

    // QTA bounds violation checkbox
    qtaBoundsCheckBox = new QCheckBox("QTA Bound Violation", this);
    formLayout->addRow(qtaBoundsCheckBox);

    // Connect signals
    connect(sampleLimitCheckBox, &QCheckBox::checkStateChanged, this,
   &TriggersTab::onTriggerChanged);
    connect(sampleLimitSpinBox, QOverload<int>::of(&QSpinBox::
   valueChanged), this, &TriggersTab::onTriggerChanged);
    connect(qtaBoundsCheckBox, &QCheckBox::checkStateChanged, this, &
   TriggersTab::onTriggerChanged);

    mainLayout->addLayout(formLayout);

```

```
49      // Triggered events list
50      triggeredList = new QListWidget(this);
51      triggeredList->setSelectionMode(QAbstractItemView::SingleSelection
        );
52      triggeredList->setMinimumHeight(150);
53      connect(triggeredList, &QListWidget::itemClicked, this, &
        TriggersTab::onTriggeredItemClicked);
54
55      mainLayout->addWidget(new QLabel("Triggered Snapshots:"));
56      mainLayout->addWidget(triggeredList);
57
58      // Set up system observer
59      Application::getInstance().addObserver([this]() { this->
        populateObservables(); });
60
61      populateObservables();
62  }
63
64  TriggersTab::~TriggersTab()
65  {
66      delete mainLayout;
67  }
68
69  /**
70   * @brief Populates the observable dropdown with available probes and
        outcomes.
71   */
72  void TriggersTab::populateObservables()
73  {
74      try {
75          auto system = Application::getInstance().getSystem();
76          observableComboBox->clear();
77          for (const auto &[name, obs] : system->getObservables()) {
78              if (obs) {
79                  observableComboBox->addItem(QString::fromStdString(
        name));
80              }
81          }
82      } catch (std::exception &) {
83          return;
84      }
85  }
86
87  /**
88   * @brief Handles observable selection changes.
89   * @param name The newly selected observable name (unused, signal
        requires parameter).
90   */
91  void TriggersTab::onObservableChanged(const QString &)
92  {
93      updateCheckboxStates();
94  }
95
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(
    sampleLimitSpinBox->value()),
110                 [this, name](const DeltaQ &, const QTA &, std::
    uint64_t time) {
111                     this->captureSnapshots(time, name);
112                 },
113                 true,
114                 sampleLimitSpinBox->value());
115         } else {
116             observable->removeTrigger(TriggerType::SampleLimit);
117         }
118 
119         // Update QTA bounds trigger
120         if (qtaBoundsCheckBox->isChecked()) {
121             observable->addTrigger(
122                 TriggerType::QTAViolation,
123                 TriggerDefs::Conditions::QTABounds(),
124                 [this, name](const DeltaQ &, const QTA &, std::
    uint64_t time) {
125                     this->captureSnapshots(time, name);
126                 },
127                 true,
128                 std::nullopt);
129         } else {
130             observable->removeTrigger(TriggerType::QTAViolation);
131         }
132     } catch (std::exception &) {
133         return;
134     }
135 }
136 
137 /**
138  * @brief Captures snapshots when triggers are activated.
139  * @param time The timestamp of the trigger event.
140  * @param name The name of the observable that triggered.
141  */
142 void TriggersTab::captureSnapshots(std::uint64_t time, const std::
    string &name)
143 {
144     auto system = Application::getInstance().getSystem();
145     if (!system->isRecording()) {
146         system->setRecording(true);
147         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
    ::SampleLimit));
200         qtaBoundsCheckBox->setChecked(activeTypes.count(TriggerType::
    QTAViolation));
```

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

```
    <Snapshot> &>(snapshotList));
255         viewer->setAttribute(Qt::WA_DeleteOnClose);
256         viewer->resize(800, 600);
257         viewer->show();
258     }
259 }
```

F.3 Outcome diagram

The "diagram" folder contains everything related to outcome diagrams. Due to time related issues, there are some issues with the names. We will explain what each class represents, but it differs from the definitions which are explained in the thesis.

F.3.1 Observable.cpp

The observable class represents a generic "observable" element of the outcome diagram, it is the base class for probes, outcome and operators. In this class one can calculate the observed ΔQ, store the outcome instances (samples), set the parameters, set a QTA, add/remove triggers and get a snapshot. It is what we described throughout the whole paper as a probe.

```
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
       lowerTime, std::uint64_t upperTime)
22 {
23     std::lock_guard<std::mutex> lock(samplesMutex);
24     if (!sorted) {
25         std::sort(samples.begin(), samples.end(), [](const Sample &a,
       const Sample &b) { return a.endTime < b.endTime; });
26         sorted = true;
27     }
28
29     std::vector<Sample> selectedSamples;
```

```
30
31      auto lower = std::lower_bound(samples.begin(), samples.end(),
    lowerTime, [](const Sample &s, long long time) { return s.endTime
    < time; });
32
33      auto upper = std::upper_bound(samples.begin(), samples.end(),
    upperTime, [](long long time, const Sample &s) { return time < s.
    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      updateSnapshot(timeLowerBound, deltaQ);
54
55      triggerManager.evaluate(deltaQ, qta, timeLowerBound);
56      return deltaQ;
57  }
58
59  void Observable::updateSnapshot(uint64_t timeLowerBound, DeltaQ &
    deltaQ)
60  {
61      if (!(deltaQ == DeltaQ())) {
62          observedInterval.addDeltaQ(deltaQ);
63          confidenceIntervalHistory.push_back(deltaQ);
64
65          if (confidenceIntervalHistory.size() > MAX_DQ) {
66              observedInterval.removeDeltaQ(confidenceIntervalHistory.
    front());
67              confidenceIntervalHistory.pop_front();
68          }
69      }
70      observableSnapshot.addObservedDeltaQ(timeLowerBound, deltaQ,
    observedInterval.getBounds());
71      observableSnapshot.addQTA(timeLowerBound, qta);
72      if (!recording && (confidenceIntervalHistory.size() > MAX_DQ)) {
73          observableSnapshot.resizeTo(MAX_DQ);
74      }
```

```
75 }
76
77 DeltaQ Observable::getObservedDeltaQ(uint64_t timeLowerBound, uint64_t
       timeUpperBound)
78 {
79     std::lock_guard<std::mutex> lock(observedMutex);
80     auto deltaQRepr = observableSnapshot.getObservedDeltaQAtTime(
       timeLowerBound);
81     if (!deltaQRepr.has_value()) {
82         calculateObservedDeltaQ(timeLowerBound, timeUpperBound);
83         deltaQRepr = observableSnapshot.getObservedDeltaQAtTime(
       timeLowerBound);
84     }
85     return deltaQRepr.value().deltaQ;
86 }
87
88 DeltaQRepr Observable::getObservedDeltaQRepr(uint64_t timeLowerBound,
       uint64_t timeUpperBound)
89 {
90     std::lock_guard<std::mutex> lock(observedMutex);
91     auto deltaQRepr = observableSnapshot.getObservedDeltaQAtTime(
       timeLowerBound);
92     if (!deltaQRepr.has_value()) {
93         calculateObservedDeltaQ(timeLowerBound, timeUpperBound);
94         deltaQRepr = observableSnapshot.getObservedDeltaQAtTime(
       timeLowerBound);
95     }
96     return deltaQRepr.value();
97 }
98
99 void Observable::setQTA(const QTA &newQTA)
100 {
101     if (newQTA.perc_25 > maxDelay || newQTA.perc_50 > maxDelay ||
       newQTA.perc_75 > maxDelay)
102         throw std::invalid_argument("Percentages should not be bigger
       than maximum delay " + std::to_string(maxDelay));
103     qta = newQTA;
104 }
105
106 void Observable::addTrigger(TriggerType type, TriggerDefs::Condition
       condition, TriggerDefs::Action action, bool enabled, std::optional
       <int> sampleLimitVal)
107 {
108     auto &tm = triggerManager;
109     tm.addTrigger(type, condition, action, enabled);
110     auto &trigger = tm.getTriggersByType(type).back();
111     trigger.sampleLimitValue = sampleLimitVal;
112 }
113
114 void Observable::removeTrigger(TriggerType type)
115 {
116     triggerManager.removeTriggersByType(type);
117 }
118
119 void Observable::setRecording(bool isRecording)
120 {
```

```
121     if (recording && !isRecording) {
122         recording = isRecording;
123         observableSnapshot.resizeTo(30); // FIXME magic numb
124     }
125     recording = isRecording;
126 }
127 
128 Snapshot Observable::getSnapshot()
129 {
130     return observableSnapshot;
131 }
132 
133 double Observable::setNewParameters(int newExp, int newNBins)
134 {
135     std::lock_guard<decltype(paramMutex)> lock(paramMutex);
136 
137     bool binsChanged = (newNBins != nBins);
138     bool expChanged = (newExp != deltaTExp);
139 
140     nBins = newNBins;
141     deltaTExp = newExp;
142 
143     maxDelay = DELTA_T_BASE * std::pow(2, deltaTExp) * nBins;
144 
145     if (qta.perc_25 > maxDelay || qta.perc_50 > maxDelay || qta.
    perc_75 > maxDelay) {
146         qta = QTA::create(0, 0, 0, qta.cdfMax, false);
147     }
148 
149     if (binsChanged || expChanged) {
150         observedInterval.setNumBins(nBins);
151         confidenceIntervalHistory.clear();
152     }
153 
154     return maxDelay;
155 }
156 
157 std::string Observable::getName() const &
158 {
159     return name;
160 }
```

F.3.2 Operator.cpp

This class represent a generic operator, it can be either an FTF, ATF or PC. It allows calculating the "calculated ΔQ".

```
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>
```

```cpp
#include <limits>
#define OP_EPSILON std::numeric_limits<double>::epsilon()

Operator::Operator(const std::string &name, OperatorType type)
    : Observable(name)
    , type(type)
    , calculatedInterval(50)
{
}

Operator::~Operator() = default;

void Operator::setProbabilities(const std::vector<double> &probs)
{
    if (type != OperatorType::PRB) {
        throw std::invalid_argument("Only probabilistic operators accept probabilities");
    }

    double result = std::reduce(probs.begin(), probs.end());

    if (std::fabs(1 - result) > OP_EPSILON)
        throw std::logic_error("Result should approximate to 1");

    probabilities = probs;
}

DeltaQ Operator::calculateCalculatedDeltaQ(uint64_t timeLowerBound, uint64_t timeUpperBound)
{
    if (observableSnapshot.getCalculatedDeltaQAtTime(timeLowerBound).has_value()) {
        return observableSnapshot.getCalculatedDeltaQAtTime(timeLowerBound).value().deltaQ;
    }
    std::vector<DeltaQ> deltaQs;

    // Get DeltaQ for all children
    for (auto &childrenLinks : causalLinks) {
        std::vector<DeltaQ> childrenDeltaQs;

        // For each children in causal link, get their DeltaQ
        for (auto &component : childrenLinks) {
            childrenDeltaQs.push_back(component->getObservedDeltaQ(timeLowerBound, timeUpperBound));
        }
        // Get the convolution of the components in a children
        deltaQs.push_back(convolveN(childrenDeltaQs));
    }
    DeltaQ result;

    // Choose appropriate operation
    if (type == OperatorType::FTF)
        result = firstToFinish(deltaQs);
    else if (type == OperatorType::PRB)
        result = probabilisticChoice(probabilities, deltaQs);
```

```
60      else
61          result = allToFinish(deltaQs);
62
63      int calculatedBins = result.getBins();
64
65      if (calculatedBins != calculatedInterval.getBins()) {
66          calculatedInterval.setNumBins(calculatedBins);
67          calculatedDeltaQHistory.clear();
68      }
69
70      calculatedInterval.addDeltaQ(result);
71      calculatedDeltaQHistory.push_back(result);
72
73      if (calculatedDeltaQHistory.size() > MAX_DQ) {
74          calculatedInterval.removeDeltaQ(calculatedDeltaQHistory.front
    ());
75          calculatedDeltaQHistory.pop_front();
76      }
77
78      observableSnapshot.addCalculatedDeltaQ(timeLowerBound, result,
    calculatedInterval.getBounds());
79
80      if (!recording && calculatedDeltaQHistory.size() > MAX_DQ) {
81          observableSnapshot.removeOldestCalculatedDeltaQ();
82      }
83      return result;
84  }
85
86  DeltaQRepr Operator::getCalculatedDeltaQRepr(uint64_t timeLowerBound,
    uint64_t timeUpperBound)
87  {
88      std::lock_guard<std::mutex> lock(calcMutex);
89      auto deltaQRepr = observableSnapshot.getCalculatedDeltaQAtTime(
    timeLowerBound);
90      if (!deltaQRepr.has_value()) {
91          calculateCalculatedDeltaQ(timeLowerBound, timeUpperBound);
92          deltaQRepr = observableSnapshot.getCalculatedDeltaQAtTime(
    timeLowerBound);
93      }
94      return deltaQRepr.value();
95  }
```

F.3.3 Outcome.cpp

This class represents a simple outcome.

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

F.3.4 Probe.cpp

This class represents what we described as "sub-outcome diagram". As the operator class, it allows calculating the "calculated ΔQ".

```
#include "Probe.h"
#include "../maths/ConfidenceInterval.h"
#include "../maths/DeltaQOperations.h"
#include <iostream>
#include <memory>
#include <mutex>
Probe::Probe(const std::string &name)
    : Observable(name)
    , calculatedInterval(50)
{
}

Probe::Probe(const std::string &name, std::vector<std::shared_ptr<
    Observable>> causalLinks)
    : Observable(name)
    , causalLinks(causalLinks)
    , calculatedInterval(50)
{
}

Probe::~Probe() = default;

DeltaQ Probe::calculateCalculatedDeltaQ(uint64_t timeLowerBound,
    uint64_t timeUpperBound)
{
    if (observableSnapshot.getCalculatedDeltaQAtTime(timeLowerBound).
    has_value()) {
        return observableSnapshot.getCalculatedDeltaQAtTime(
    timeLowerBound).value().deltaQ;
    }
    std::vector<DeltaQ> deltaQs;
    for (const auto &component : causalLinks) {

        deltaQs.push_back(component->getObservedDeltaQ(timeLowerBound,
    timeUpperBound));
    }

    DeltaQ result = convolveN(deltaQs);
    int calculatedBins = result.getBins();

    if (calculatedBins != calculatedInterval.getBins()) {
        calculatedInterval.setNumBins(calculatedBins);
        calculatedDeltaQHistory.clear();
    }

    calculatedInterval.addDeltaQ(result);
    calculatedDeltaQHistory.push_back(result);

    if (calculatedDeltaQHistory.size() > MAX_DQ) {
        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,
    uint64_t timeUpperBound)
72  {
73      std::lock_guard<std::mutex> lock(calcMutex);
74      auto deltaQRepr = observableSnapshot.getCalculatedDeltaQAtTime(
    timeLowerBound);
75      if (!deltaQRepr.has_value()) {
76          calculateCalculatedDeltaQ(timeLowerBound, timeUpperBound);
77          deltaQRepr = observableSnapshot.getCalculatedDeltaQAtTime(
    timeLowerBound);
78      }
79      return deltaQRepr.value();
80  }
```

F.3.5 System.cpp

This class represents the system, the whole outcome diagram. It coordinates the various parts of the outcome diagram.

```
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::
       shared_ptr<Outcome>> outcomesMap)
12 {
13     outcomes = outcomesMap;
14     for (auto &[name, outcome] : outcomes) {
15         observables[name] = outcome;
16     }
17 }
18
19 void System::setProbes(std::unordered_map<std::string, std::shared_ptr
       <Probe>> probesMap)
20 {
21     probes = probesMap;
22     for (auto &[name, probe] : probes) {
23         observables[name] = probe;
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<
       Observable>> &System::getObservables()
75 {
76     return observables;
77 }
78
79 bool System::hasOutcome(const std::string &name)
80 {
81     return outcomes.find(name) != outcomes.end();
82 }
83
84 bool System::hasProbe(const std::string &name)
85 {
86     return probes.find(name) != probes.end();
87 }
88
89 bool System::hasOperator(const std::string &name)
90 {
91     return operators.find(name) != operators.end();
92 }
93
94 std::shared_ptr<Operator> System::getOperator(const std::string &name)
95 {
96     return operators[name];
97 }
98
99 std::shared_ptr<Probe> System::getProbe(const std::string &name)
100 {
101     return probes[name];
102 }
103
104 std::vector<std::string> System::getAllComponentsName()
105 {
106     std::vector<std::string> names;
107     names.reserve(observables.size());
108
109     for (auto &[name, obs] : observables) {
110         if (obs)
```

```
111             names.emplace_back(name);
112     }
113
114     return names;
115 }
116
117 void System::setSystemDefinitionText(std::string &text)
118 {
119     systemDefinitionText = text;
120 }
121
122 std::string System::getSystemDefinitionText()
123 {
124     return systemDefinitionText;
125 }
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 }
```

F.4 ΔQ (maths)

The "maths" folder represents all the classes related to ΔQ, where mathematical operations are being done (hence the "maths" name).

F.4.1 ConfidenceInterval.cpp

This class represents the confidence bounds described earlier.

```
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
14     : numBins(numBins)
15     , cdfSum(numBins, 0.0)
16     , cdfSumSquares(numBins, 0.0)
17     , cdfSampleCounts(numBins, 0)
18     , bounds(numBins)
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.
    getBins() << " expected" << numBins << "\n";
30         return;
31     }
32     for (size_t i = 0; i < cdf.size(); ++i) {
33         const double cdfValue = cdf[i];
34
35         cdfSum[i] += cdfValue;
36         cdfSumSquares[i] += cdfValue * cdfValue;
37         cdfSampleCounts[i] += 1;
38     }
39
40     updateConfidenceInterval();
```

```
}

void ConfidenceInterval::removeDeltaQ(const DeltaQ &deltaQ)
{
    if (deltaQ == DeltaQ()) {
        return;
    }

    const auto &cdf = deltaQ.getCdfValues();
    if (cdf.size() != numBins) {
        return; // Returning a previous DeltaQ which had different
    bins
    }

    for (size_t i = 0; i < cdf.size(); ++i) {
        const double cdfValue = cdf[i];

        cdfSum[i] -= cdfValue;
        cdfSumSquares[i] -= cdfValue * cdfValue;

        if (cdfSampleCounts[i] == 0) {
            return; // Removing more than needed
        }

        cdfSampleCounts[i] -= 1;
    }

    updateConfidenceInterval();
}

void ConfidenceInterval::updateConfidenceInterval()
{
    for (size_t i = 0; i < bounds.size(); ++i) {
        unsigned int n = cdfSampleCounts[i];
        if (cdfSampleCounts[i] == 0) {
            bounds[i].lowerBound = 0.0;
            bounds[i].upperBound = 0.0;
            continue;
        }

        double mean = cdfSum[i] / n;
        double meanSquare = cdfSumSquares[i] / n;
        double variance = meanSquare - (mean * mean);
        double stddev = std::sqrt(std::max(variance, 0.0));
        double marginOfError = z * stddev / std::sqrt(n);
        bounds[i].lowerBound = std::max(0.0, mean - marginOfError);
        bounds[i].upperBound = std::min(1.0, mean + marginOfError);
        bounds[i].mean = mean;
    }
}

void ConfidenceInterval::reset()
{
    bounds = std::vector<Bound>();
    bounds.resize(numBins);
    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 }
```

F.4.2 DeltaQ.cpp

This class represents a ΔQ. It supports calculating a ΔQ given multiple samples, calculating the quartiles of a ΔQ, it supports various arithmetical transformations.

```
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(),
    result.pdfValues.begin(), [constant](double value) { return value
    * constant; });
224 
225     result.cdfValues = deltaQ.cdfValues;
226     std::transform(result.cdfValues.begin(), result.cdfValues.end(),
    result.cdfValues.begin(), [constant](double value) { return value
    * constant; });
227 
228     return result;
229 }
230 
231 template <typename BinaryOperation>
232 DeltaQ applyBinaryOperation(const DeltaQ &lhs, const DeltaQ &rhs,
    BinaryOperation op)
233 {
```

```
234     const DeltaQ &highestDeltaQ = (lhs > rhs) ? lhs : rhs;
235     const DeltaQ &otherDeltaQ = (lhs > rhs) ? rhs : lhs;
236 
237     std::vector<double> resultingCdf;
238     resultingCdf.reserve(highestDeltaQ.getBins());
239 
240     for (size_t i = 0; i < highestDeltaQ.getBins(); i++) {
241         double result = op(highestDeltaQ.cdfAt(i), otherDeltaQ.cdfAt(i
    ));
242         resultingCdf.push_back(result);
243     }
244 
245     return {highestDeltaQ.getBinWidth(), resultingCdf, false};
246 }
247 
248 DeltaQ operator*(double constant, const DeltaQ &deltaQ)
249 {
250     return deltaQ * constant;
251 }
252 
253 DeltaQ operator+(const DeltaQ &lhs, const DeltaQ &rhs)
254 {
255     return applyBinaryOperation(lhs, rhs, std::plus<>());
256 }
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 << ",
    " << std::setprecision(7) << pdfValues[i] << ")";
277         if (i < pdfValues.size() - 1) {
278             oss << ", ";
279         }
280     }
281 
282     oss << ">";
283     return oss.str();
284 }
```

F.4.3 DeltaQOperations.cpp

This file contains the definition of all the operations that can be done on a ΔQ or on ΔQs, specified in the implementation chapter. Convolution (naive, FFT), FTF, ATF, PC operators, rebinning.

```cpp
#include "DeltaQOperations.h"
#include "DeltaQ.h"
#include <fftw3.h>
#include <iostream>
#include <math.h>
#include <mutex>
#include <vector>
DeltaQ rebin(const DeltaQ &source, double targetBinWidth)
{
    double originalBinWidth = source.getBinWidth();

    if (std::abs(originalBinWidth - targetBinWidth) < 1e-9) {
        return source; // Already same bin width
    }

    if (targetBinWidth < originalBinWidth) {
        throw std::invalid_argument("Target bin width must be greater than or equal to original.");
    }

    int factor = static_cast<int>(std::round(targetBinWidth / originalBinWidth));
    const auto &originalPdf = source.getPdfValues();

    int newNumBins = static_cast<int>(std::ceil(originalPdf.size() / static_cast<double>(factor)));
    std::vector<double> newPdf(newNumBins, 0.0);

    for (size_t i = 0; i < originalPdf.size(); ++i) {
        newPdf[i / factor] += originalPdf[i];
    }

    return DeltaQ(targetBinWidth, newPdf, true);
}

DeltaQ convolve(const DeltaQ &lhs, const DeltaQ &rhs)
{
    double commonBinWidth = std::max(lhs.getBinWidth(), rhs.getBinWidth());

    if (lhs == DeltaQ()) {
        return rhs;
    }
    if (rhs == DeltaQ()) {
        return lhs;
    }

    DeltaQ lhsRebinned = rebin(lhs, commonBinWidth);
    DeltaQ rhsRebinned = rebin(rhs, commonBinWidth);

```

```
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
125         resultTime.resize(convSize);
126
127         fftw_destroy_plan(planLhs);
128         fftw_destroy_plan(planRhs);
129         fftw_destroy_plan(planInv);
130         fftw_free(lhsFreq);
131         fftw_free(rhsFreq);
132         fftw_free(resultFreq);
133         return {lhsRebinned.getBinWidth(), resultTime, true};
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,
    const std::vector<DeltaQ> &deltaQs)
```

```
{
    std::vector<DeltaQ> nonEmpty;
    std::vector<double> effectiveProbs;
    double commonBinWidth = 0.0;

    for (size_t i = 0; i < deltaQs.size(); ++i) {
        if (deltaQs[i] == DeltaQ()) {
            continue;
        }
        nonEmpty.push_back(deltaQs[i]);
        effectiveProbs.push_back(probabilities[i]);
        commonBinWidth = std::max(commonBinWidth, deltaQs[i].
    getBinWidth());
    }

    if (nonEmpty.empty()) {
        return DeltaQ();
    }

    for (auto &dq : nonEmpty) {
        dq = rebin(dq, commonBinWidth);
    }

    std::vector<DeltaQ> scaledDeltaQs;
    for (size_t i = 0; i < nonEmpty.size(); ++i) {
        scaledDeltaQs.push_back(nonEmpty[i] * effectiveProbs[i]);
    }

    DeltaQ result = scaledDeltaQs[0];
    for (size_t i = 1; i < scaledDeltaQs.size(); ++i) {
        result = result + scaledDeltaQs[i];
    }

    return result;
}

DeltaQ firstToFinish(const std::vector<DeltaQ> &deltaQs)
{
    std::vector<DeltaQ> nonEmpty;
    double commonBinWidth = 0.0;

    for (const auto &dq : deltaQs) {
        if (dq == DeltaQ()) {
            continue;
        }
        nonEmpty.push_back(dq);
        commonBinWidth = std::max(commonBinWidth, dq.getBinWidth());
    }

    if (nonEmpty.empty()) {
        return DeltaQ();
    }

    for (auto &dq : nonEmpty) {
        dq = rebin(dq, commonBinWidth);
    }
```

```
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 }
```

F.4.4 Snapshot.cpp

This class represents a single snapshot of a probe. It contains the QTA, observable ΔQ and calculated ΔQ at time t.

```
#include "Snapshot.h"

void Snapshot::addObservedDeltaQ(std::uint64_t time, const DeltaQ &
    deltaQ, const std::vector<Bound> &bounds)
{
    DeltaQRepr repr = {time, deltaQ, bounds};
    observedDeltaQs[time] = repr;
}

void Snapshot::removeOldestObservedDeltaQ()
{
    if (getObservedSize() == 0)
        return;
    observedDeltaQs.erase(observedDeltaQs.begin());
}

void Snapshot::addCalculatedDeltaQ(std::uint64_t time, const DeltaQ &
    deltaQ, const std::vector<Bound> &bounds)
{
    DeltaQRepr repr = {time, deltaQ, bounds};
    calculatedDeltaQs[time] = repr;
}

DeltaQ Snapshot::getOldestCalculatedDeltaQ() const
{
    return calculatedDeltaQs.begin()->second.deltaQ;
}

DeltaQ Snapshot::getOldestObservedDeltaQ() const
{
    return observedDeltaQs.begin()->second.deltaQ;
}

void Snapshot::removeOldestCalculatedDeltaQ()
{
    if (getCalculatedSize() == 0)
        return;

    calculatedDeltaQs.erase(calculatedDeltaQs.begin());
}

void Snapshot::resizeTo(size_t newSize)
{
    int observedSize = getObservedSize();
    int calculatedSize = getCalculatedSize();

    // Don't shrink if not needed
    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 }
```

F.4.5 TriggerManager.cpp

This class is the manager of triggers for a probe. It can add/remove/evaluate triggers.

```
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 {
```

```
    triggers_.emplace_back(type, std::move(condition), std::move(action), enabled);
}

std::vector<TriggerManager::Trigger> TriggerManager::getTriggersByType(TriggerType type)
{
    std::vector<Trigger> result;
    for (auto &trigger : triggers_) {
        if (trigger.type == type) {
            result.push_back(trigger);
        }
    }
    return result;
}

void TriggerManager::evaluate(const DeltaQ &dq, const QTA &qta, std::uint64_t time) const
{
    for (const auto &trigger : triggers_) {
        if (trigger.enabled && trigger.condition(dq, qta)) {
            trigger.action(dq, qta, time);
        }
    }
}

void TriggerManager::removeTriggersByType(TriggerType type)
{
    triggers_.erase(std::remove_if(triggers_.begin(), triggers_.end(),
    [type](const auto &trigger) { return trigger.type == type; }), triggers_.end());
}

void TriggerManager::clearAllTriggers()
{
    triggers_.clear();
}

void TriggerManager::setTriggersEnabled(TriggerType type, bool enabled)
{
    for (auto &trigger : triggers_) {
        if (trigger.type == type) {
            trigger.enabled = enabled;
        }
    }
}
std::vector<TriggerManager::Trigger> TriggerManager::getAllTriggers() const
{
    std::vector<Trigger> result;
    for (const auto &trigger : triggers_) {
        result.push_back(trigger);
    }
    return result;
}
```

F.4.6 Triggers.cpp

This class contains the conditions of the triggers selected by the user. The trigger manager evaluates the conditions at runtime. The Actions namespace is WIP.

```
#include "Triggers.h"

namespace TriggerDefs
{
namespace Conditions
{
    Condition SampleLimit(int maxSamples)
    {
        return [maxSamples](const DeltaQ &dq, const QTA &) { return dq.getTotalSamples() > maxSamples; };
    }

    Condition QTABounds()
    {
        return [](const DeltaQ &dq, const QTA &qta) {
            const QTA &dqQta = dq.getQTA();
            return dqQta.perc_25 > qta.perc_25 || dqQta.perc_50 > qta.perc_50 || dqQta.perc_75 > qta.perc_75 || dqQta.cdfMax < qta.cdfMax;
        };
    }

    Condition FailureRate(double threshold)
    {
        return [threshold](const DeltaQ &dq, const QTA &) { return dq.getQTA().cdfMax < threshold; };
    }
}

namespace Actions
{
    Action LogToConsole(const std::string &message)
    {
        return [message](const DeltaQ &, const QTA &, std::uint64_t) { std::cout << "TRIGGER: " << message << "\n"; };
    }

    Action notify()
    {
        return [](const DeltaQ &dq, const QTA &qta, std::uint64_t) {
    };
    }

    Action SaveSnapshot(const std::string &filename)
    {
        return [](const DeltaQ &dq, const QTA &, std::uint64_t) { };
    }
}
```

```
}
```

F.5 parser

F.5.1 SystemBuilder.cpp

This class builds a new outcome diagram (system class) given an AST built when parsing.

```
#include "SystemBuilder.h"
#include <memory>
#include <stdexcept>

#include <locale>
System SystemBuilderVisitor::getSystem() const
{
    return system;
}

void SystemBuilderVisitor::checkForCycles() const
{
    std::set<std::string> visited;
    std::set<std::string> recursionStack;

    for (const auto &[node, _] : dependencies) {
        if (hasCycle(node, visited, recursionStack)) {
            throw std::invalid_argument("Cycle detected in system definition involving: " + node);
        }
    }
}

bool SystemBuilderVisitor::hasCycle(const std::string &node, std::set<std::string> &visited, std::set<std::string> &recursionStack) const
{
    if (recursionStack.find(node) != recursionStack.end()) {
        return true;
    }
    if (visited.find(node) != visited.end()) {
        return false;
    }

    visited.insert(node);
    recursionStack.insert(node);

    if (dependencies.find(node) != dependencies.end()) {
        for (const auto &neighbor : dependencies.at(node)) {
            if (hasCycle(neighbor, visited, recursionStack)) {
                return true;
            }
        }
    }

```

```
43      recursionStack.erase(node);
44      return false;
45  }
46
47  std::any SystemBuilderVisitor::visitStart(parser::DQGrammarParser::
        StartContext *context)
48  {
49      for (const auto definition : context->definition()) {
50          visitDefinition(definition);
51      }
52
53      if (context->system()) {
54          visitSystem(context->system());
55      }
56
57      system.setOutcomes(outcomes);
58      system.setProbes(probes);
59      system.setOperators(operators);
60      /*
61      for (const auto &[name, link] : definitionLinks) {
62          std::cout << name << " [ ";
63          for (auto &name2 : link) {
64              std::cout << name2 << " ";
65          }
66          std::cout << "]\n";
67      }
68
69      for (const auto &[name, op] : operatorLinks) {
70          std::cout << name << " [ ";
71          for (auto link : op) {
72              std::cout << " [";
73              for (auto lill : link) {
74                  std::cout << lill << " ";
75              }
76              std::cout << "]";
77          }
78          std::cout << "]\n";
79      }
80      */
81      checkForCycles();
82      return nullptr;
83  }
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     allNames.insert(name);
153     OperatorType type;
154     if (typeStr == "a")
155         type = OperatorType::ATF;
156     else if (typeStr == "f")
157         type = OperatorType::FTF;
158     else if (typeStr == "p")
159         type = OperatorType::PRB;
160     else
161         throw std::invalid_argument("Unknown operator type: " +
    typeStr);
162 
163     const auto op = std::make_shared<Operator>(name, type);
164     if (type == OperatorType::PRB && context->probability_list()) {
165         const auto probabilities = std::any_cast<std::vector<double>>(
    visitProbability_list(context->probability_list()));
166         op->setProbabilities(probabilities);
167     } else if (type == OperatorType::PRB && !context->probability_list
    ()) {
168         throw std::invalid_argument("A probabilistic operator must
    have probabilities");
169     } else if (type != OperatorType::PRB && context->probability_list
    ()) {
170         throw std::invalid_argument("A non probabilistic operator
    cannot have probabilities");
171     }
172 
173     std::vector<std::vector<std::shared_ptr<Observable>>>
    operatorPtrLinks;
174 
175     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;

    for (auto chainCtx : context->component_chain()) {
        auto chain = std::any_cast<std::vector<std::shared_ptr<
    Observable>>>(visitComponent_chain(chainCtx));
        componentsChains.push_back(chain);
    }

    return componentsChains;
}

std::any SystemBuilderVisitor::visitComponent_chain(parser::
    DQGrammarParser::Component_chainContext *context)
{
    std::vector<std::shared_ptr<Observable>> components;

    for (auto compCtx : context->component()) {
        auto component = std::any_cast<std::shared_ptr<Observable>>(
    visitComponent(compCtx));
        components.push_back(component);
    }
    return components;
}

std::any SystemBuilderVisitor::visitOutcome(parser::DQGrammarParser::
    OutcomeContext *context)
{
    std::string name = context->IDENTIFIER()->getText();

    if (outcomes.find(name) != outcomes.end()) {
        return std::dynamic_pointer_cast<Observable>(outcomes[name]);
    }
    if (allNames.find(name) != allNames.end()) {
        throw std::invalid_argument("Duplicate name detected: " + name
    );
    }
    allNames.insert(name);

    auto outcome = std::make_shared<Outcome>(name);
    outcomes[name] = outcome;
    return std::dynamic_pointer_cast<Observable>(outcome);
}
```

F.5.2 SystemParserInterface.cpp

This files contains functions to be called by the dashboard to avoid communicating directly to ANTLR. It throws errors which are caught by the caller if the parsing was unsuccessful.

```
#include "SystemParserInterface.h"
#include "SystemErrorListener.h"
#include <exception>
#include <fstream>
#include <iostream>
```

```cpp
#include <sstream>
#include <stdexcept>

/**
 * @brief Parses a system definition from a file.
 * @param filename Path to the file containing system definition.
 * @return Optional containing the parsed System if successful, nullopt on error.
 * @throws std::invalid_argument if parsing fails.
 */
std::optional<System> SystemParserInterface::parseFile(const std::string &filename)
{
    std::ifstream file(filename);
    if (!file) {
        std::cerr << "Error: Could not open file: " << filename << std::endl;
        return std::nullopt;
    }

    // Read entire file content
    std::stringstream buffer;
    buffer << file.rdbuf();
    std::string content = buffer.str();

    antlr4::ANTLRInputStream input(content);
    try {
        return parseInternal(input);
    } catch (std::exception &e) {
        throw std::invalid_argument(e.what());
    }
}

/**
 * @brief Parses a system definition from a string.
 * @param inputStr String containing system definition.
 * @return Optional containing the parsed System if successful, nullopt on error.
 * @throws std::invalid_argument if parsing fails.
 */
std::optional<System> SystemParserInterface::parseString(const std::string &inputStr)
{
    antlr4::ANTLRInputStream input(inputStr);
    try {
        return parseInternal(input);
    } catch (std::exception &e) {
        throw std::invalid_argument(e.what());
    }
}

/**
 * @brief Internal parsing implementation using ANTLR.
 * @param input ANTLR input stream containing system definition.
 * @return Optional containing the parsed System if successful, nullopt on error.
```

```
 * @throws std::invalid_argument if parsing fails.
 */
std::optional<System> SystemParserInterface::parseInternal(antlr4::
    ANTLRInputStream &input)
{
    // Initialize lexer and parser
    parser::DQGrammarLexer lexer(&input);
    antlr4::CommonTokenStream tokens(&lexer);
    parser::DQGrammarParser parser(&tokens);

    // Configure error handling
    SystemErrorListener errorListener;
    lexer.removeErrorListeners();
    parser.removeErrorListeners();
    lexer.addErrorListener(&errorListener);
    parser.addErrorListener(&errorListener);

    try {
        // Parse and build system
        auto tree = parser.start();
        SystemBuilderVisitor visitor;
        visitor.visitStart(tree);
        return visitor.getSystem();
    } catch (std::exception &e) {
        throw std::invalid_argument(e.what());
    }
}
```

F.6 server

F.6.1 Server.cpp

This class represents the server which receives and sends messages from Erlang.

```
#include "Server.h"
#include "../Application.h"
#include <arpa/inet.h>
#include <cstdint>
#include <cstring>
#include <fcntl.h>
#include <iostream>
#include <regex>
#include <signal.h>
#include <sys/socket.h>
#include <unistd.h>

#define TIMEOUT "to"
#define EXEC_OK "ok"
#define FAIL "fa"

// Inspired from https://beej.us/guide/bgnet/html//index.html#client-
    server-background

/**
```

```
 * @brief Constructs the Server and registers system observer.
 * @param port The TCP port to listen on.
 */
Server::Server(int port)
    : port(port)
    , server_fd(0)
    , new_socket(0)
    , server_started(false)
{
    Application::getInstance().addObserver([this]() { this->updateSystem(); });

    // Start worker thread (this can run independently)
    workerThread = std::thread([this]() {
        while (!shutdownWorker) {
            std::unique_lock lock(queueMutex);
            queueCond.wait(lock, [this] { return !sampleQueue.empty() || shutdownWorker; });

            while (!sampleQueue.empty()) {
                auto [name, sample] = sampleQueue.front();
                sampleQueue.pop();
                lock.unlock();

                if (system) {
                    system->addSample(name, sample);
                }

                lock.lock();
            }
        }
    });
}

/**
 * @brief Destructor cleans up sockets and joins threads.
 */
Server::~Server()
{
    std::lock_guard<std::mutex> lock(erlangMutex);
    if (erlang_socket > 0) {
        close(erlang_socket);
        erlang_socket = -1;
    }
    close(new_socket);
    close(server_fd);
    if (serverThread.joinable())
        serverThread.join();
}
/**
 * @brief Updates the system reference from Application.
 */
void Server::updateSystem()
{
    system = Application::getInstance().getSystem();
}
```

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

```
123         return;
124     }
125 
126     std::cout << "Server running on " << server_ip << ":" << port <<
    std::endl;
127     running = true;
128     server_started = true;
129 
130     while (running) {
131         int addrlen = sizeof(address);
132         client_socket = accept(server_fd, (struct sockaddr *)&address,
    (socklen_t *)&addrlen);
133 
134         if (client_socket < 0) {
135             if (errno == EWOULDBLOCK || errno == EAGAIN) {
136                 std::this_thread::sleep_for(std::chrono::milliseconds
    (100));
137                 cleanupThreads();
138                 continue;
139             }
140             if (!running)
141                 break; // Server was stopped
142             perror("Accept failed");
143             continue;
144         }
145 
146         std::lock_guard<std::mutex> lock(clientsMutex);
147         clientThreads.emplace_back(&Server::handleClient, this,
    client_socket);
148     }
149 
150     // Cleanup
151     close(server_fd);
152     server_fd = 0;
153     cleanupThreads();
154     server_started = false;
155 }
156 
157 /**
158  * @brief Starts the server on specified IP and port.
159  * @param ip The IP address to bind to (default: "0.0.0.0" for all
    interfaces)
160  * @param port The port to listen on
161  * @return true if server started successfully
162  */
163 bool Server::startServer(const std::string &ip, int port)
164 {
165     if (server_started) {
166         std::cerr << "Server already running. Stop it first." << std::
    endl;
167         return false;
168     }
169 
170     this->server_ip = ip;
171     this->port = port;
172 
```

```
173     serverThread = std::thread(&Server::run, this);
174 
175     // Wait a bit to see if server started successfully
176     std::this_thread::sleep_for(std::chrono::milliseconds(100));
177     return server_started;
178 }
179 
180 /**
181  * @brief Stops the server and closes all sockets.
182  */
183 void Server::stopServer()
184 {
185     if (!server_started) {
186         std::cout << "Server not running." << std::endl;
187         return;
188     }
189 
190     running = false;
191     server_started = false;
192 
193     // Close server socket to break accept loop
194     if (server_fd > 0) {
195         close(server_fd);
196         server_fd = 0;
197     }
198 
199     // Join server thread
200     if (serverThread.joinable()) {
201         serverThread.join();
202     }
203 
204     // Cleanup client threads
205     cleanupThreads();
206 
207     std::cout << "Server stopped." << std::endl;
208 }
209 
210 /**
211  * @brief Sets the Erlang endpoint for connections.
212  * @param ip Erlang server IP address
213  * @param port Erlang server port
214  * @return true if endpoint set successfully
215  */
216 bool Server::setErlangEndpoint(const std::string &ip, int port)
217 {
218     std::lock_guard<std::mutex> lock(erlangMutex);
219 
220     // Close existing connection if any
221     if (erlang_socket > 0) {
222         close(erlang_socket);
223         erlang_socket = -1;
224     }
225 
226     erlang_ip = ip;
227     erlang_port = port;
228 
```

```
229     std::cout << "Erlang endpoint set to " << ip << ":" << port << std
    ::endl;
230     return true;
231 }
232 
233 /**
234  * @brief Connects to the Erlang process.
235  * @return true if connection succeeded.
236  */
237 bool Server::connectToErlang()
238 {
239     std::lock_guard<std::mutex> lock(erlangMutex);
240 
241     if (erlang_socket > 0)
242         return true; // Already connected
243 
244     erlang_socket = socket(AF_INET, SOCK_STREAM, 0);
245     if (erlang_socket < 0) {
246         perror("Erlang socket creation failed");
247         return false;
248     }
249 
250     sockaddr_in erlang_addr {};
251     erlang_addr.sin_family = AF_INET;
252     erlang_addr.sin_port = htons(erlang_port);
253 
254     if (inet_pton(AF_INET, erlang_ip.c_str(), &erlang_addr.sin_addr)
    <= 0) {
255         std::cerr << "Invalid Erlang IP: " << erlang_ip << std::endl;
256         close(erlang_socket);
257         erlang_socket = -1;
258         return false;
259     }
260 
261     if (connect(erlang_socket, (struct sockaddr *)&erlang_addr, sizeof
    (erlang_addr)) < 0) {
262         perror("Failed to connect to Erlang");
263         close(erlang_socket);
264         erlang_socket = -1;
265         return false;
266     }
267 
268     std::cout << "Connected to Erlang on " << erlang_ip << ":" <<
    erlang_port << std::endl;
269     return true;
270 }
271 
272 /**
273  * @brief Sends a command to the Erlang process.
274  * @param command The command string to send.
275  */
276 void Server::sendToErlang(const std::string &command)
277 {
278     signal(SIGPIPE, SIG_IGN); // Ignore SIGPIPE to prevent crashes on
    disconnect
279 
```

```
280     if (!connectToErlang()) {
281         std::cerr << "Unable to send to Erlang: not connected.\n";
282         return;
283     }
284 
285     std::lock_guard<std::mutex> lock(erlangMutex);
286 
287     std::string msgWithNewline = command + "\n";
288     ssize_t sent = send(erlang_socket, msgWithNewline.c_str(),
    msgWithNewline.size(), 0);
289 
290     if (sent == -1) {
291         perror("send failed");
292 
293         if (errno == EPIPE) {
294             std::cerr << "Broken pipe: Erlang side likely disconnected
    ." << std::endl;
295             close(erlang_socket);
296             erlang_socket = -1;
297         }
298 
299         return;
300     }
301     std::cout << "Sent to Erlang: " << command << std::endl;
302 }
303 
304 /**
305  * @brief Handles communication with a client.
306  * @param clientSocket The client socket file descriptor.
307  */
308 void Server::handleClient(int clientSocket)
309 {
310     std::string buffer;
311     char tempBuf[4096];
312 
313     while (running) {
314         int valread = read(clientSocket, tempBuf, sizeof(tempBuf));
315         if (valread <= 0) {
316             if (valread == 0 || errno == ECONNRESET)
317                 break;
318             if (errno == EWOULDBLOCK || errno == EAGAIN) {
319                 std::this_thread::sleep_for(std::chrono::milliseconds
    (10));
320                 continue;
321             }
322             perror("Read failed");
323             break;
324         }
325 
326         buffer.append(tempBuf, valread);
327 
328         size_t pos;
329         size_t offset = 0;
330         while ((pos = buffer.find('\n', offset)) != std::string::npos)
     {
331             std::string_view message(buffer.data() + offset, pos -
```

```
    offset);
332             parseErlangMessage(message.data(), message.size());
333             offset = pos + 1;
334         }
335         buffer.erase(0, offset);
336     }
337 
338     close(clientSocket);
339 }
340 
341 /**
342  * @brief Cleans up finished client threads.
343  */
344 void Server::cleanupThreads()
345 {
346     std::lock_guard<std::mutex> lock(clientsMutex);
347     auto it = clientThreads.begin();
348     while (it != clientThreads.end()) {
349         if (it->joinable()) {
350             it->join();
351             it = clientThreads.erase(it);
352         } else {
353             ++it;
354         }
355     }
356 }
357 
358 /**
359  * @brief Stops the server and worker threads.
360  */
361 void Server::stop()
362 {
363     running = false;
364     {
365         std::lock_guard lock(queueMutex);
366         shutdownWorker = true;
367     }
368     queueCond.notify_all();
369     if (workerThread.joinable())
370         workerThread.join();
371 }
372 
373 /**
374  * @brief Parses messages from Erlang and adds samples to queue.
375  * @param buffer The message buffer.
376  * @param len Length of the message.
377  */
378 void Server::parseErlangMessage(const char *buffer, int len)
379 {
380     if (buffer == nullptr || len <= 0 || len >= 1024) {
381         std::cerr << "error" << std::endl;
382         return;
383     }
384 
385     std::string message(buffer, len);
386 
```

```cpp
    // Parse message components
    size_t nPos = message.find("n:");
    size_t bPos = message.find(";b:");
    size_t ePos = message.find(";e:");
    size_t sPos = message.find(";s:");

    if (nPos != 0 || bPos == std::string::npos || ePos == std::string::npos || sPos == std::string::npos) {
        std::cerr << "Failed to parse message: " << message << std::endl;
        return;
    }

    // Extract message fields
    std::string name = message.substr(2, bPos - 2);
    std::string bStr = message.substr(bPos + 3, ePos - (bPos + 3));
    std::string eStr = message.substr(ePos + 3, sPos - (ePos + 3));
    std::string statusStr = message.substr(sPos + 3);

    // Convert to sample data
    uint64_t startTime = std::stoull(bStr);
    uint64_t endTime = std::stoull(eStr);
    Sample sample;
    Status status = Status::SUCCESS;

    if (statusStr == TIMEOUT || statusStr == FAIL) {
        status = (statusStr == TIMEOUT) ? Status::TIMEDOUT : Status::FAILED;
    } else if (statusStr != EXEC_OK) {
        std::cerr << "Unknown status: " << statusStr << std::endl;
        return;
    }

    double long elapsed = (endTime - startTime) / 1'000'000'000.0L;
    sample = {startTime, endTime, elapsed, status};

    // Add to processing queue
    {
        std::lock_guard lock(queueMutex);
        sampleQueue.emplace(name, sample);
    }
    queueCond.notify_one();
}
```

Appendix G

C++ Header Files

G.1 Root

G.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
       to notify
31     Server *server = nullptr;
32     bool componentsDiffer(const std::shared_ptr<Observable> &a, const
      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  };
```

G.2 dashboard

G.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 };
```

G.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,
        QWidget *parent = nullptr);
18  };
```

G.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
        entries by name
39      QVBoxLayout *mainLayout;                     ///< Main layout of
        the panel
40      QVBoxLayout *legendLayout;                   ///< Layout containing
        the legend entries
```

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

G.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
```

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

```
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
```

G.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  };
```

G.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
       visualization of probes over time.
24  */
25 class DeltaQPlot : public QWidget
26 {
27     Q_OBJECT
28
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,
       QWidget *parent = nullptr);
36
37     /**
38      * @brief Destructor.
39      */
40     ~DeltaQPlot();
41
42     /**
43      * @brief Adds a QLineSeries to the chart.
44      * @param series Pointer to the line series.
45      * @param name Name of the series.
46      */
47     void addSeries(QLineSeries *series, const std::string &name);
48
49     /**
50      * @brief Updates the plot data using provided time range and bin
       width.
51      */
52     void update(uint64_t timeLowerBound, uint64_t timeUpperBound);
53
54     /**
55      * @brief Removes a series from the chart.
56      * @param series Series to remove.
57      */
58     void removeSeries(QAbstractSeries *series);
59
60     /**
61      * @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> &
    data);
76
77      void updateXRange(double xRange);
78      /**
79       * @brief Returns the associated plot list.
80       */
81      DQPlotList *getPlotList();
82
83      void setTitle(QString &&);
84
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
```

G.2.8 MainWindow.h

```cpp
#pragma once
#include "../diagram/System.h"
#include "DeltaQPlot.h"
#include "NewPlotList.h"
#include "ObservableSettings.h"
#include "Sidebar.h"
#include "StubControlWidget.h"
#include "TriggersTab.h"

#include <QHBoxLayout>
#include <QListWidget>
#include <QMainWindow>
#include <QPushButton>
#include <QTimer>
#include <QVBoxLayout>
#include <qboxlayout.h>
#include <qwidget.h>

/**
 * @class MainWindow
 * @brief The main application window containing plots and control panels.
 */
class MainWindow : public QMainWindow
{
    Q_OBJECT

    QHBoxLayout *mainLayout; ///< Main horizontal layout

    QScrollArea *scrollArea; ///< Scroll area for plots
    QGridLayout *plotLayout; ///< Grid layout for plot arrangement
    QWidget *plotContainer; ///< Container widget for plots

    QWidget *centralWidget; ///< Central widget for main layout

    QThread *timerThread; ///< Thread for update timer
    QTimer *updateTimer; ///< Timer for periodic updates

    QWidget *sideContainer; ///< Container for side panels
    QVBoxLayout *sideLayout; ///< Layout for side panels
    QTabWidget *sideTabWidget; ///< Tab widget for side panels
    TriggersTab *triggersTab; ///< Triggers configuration panel
    Sidebar *sidebar; ///< Main sidebar control panel
    ObservableSettings *observableSettings; ///< Observable settings panel

    StubControlWidget *stubWidget; ///< Stub control widget (placeholder)
    QPushButton *addPlotButton; ///< Button to add new plots

    QMap<DeltaQPlot *, QWidget *> plotContainers; ///< Map of plots to their containers
    uint64_t timeLowerBound; ///< Lower time bound for data updates

```

```
51      std::mutex plotDelMutex; ///< Mutex for plot deletion safety
52      std::mutex updateMutex; ///< Mutex for update operations
53
54      int samplingRate {200}; ///< Current sampling 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 };
```

G.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
       observables from the system.
36      */
37     void reset();
38
39 private:
40     /**
41      * @brief Adds observable items to the list from the current
       system.
42      */
43     void addItems();
44 };
45
46 #endif // NEW_PLOT_LIST_H
```

G.2.10 ObservableSettings.h

```
1  #ifndef OBS_SETTINGS_H
2  #define OBS_SETTINGS_H
3
4  #include <qlabel.h>
```

```
#pragma once
#include <QWidget>

#include "DelaySettingsWidget.h"
#include "QTAInputWidget.h"

class ObservableSettings : public QWidget
{
    Q_OBJECT

    QVBoxLayout *layout;

    QLabel *delayLabel;
    DelaySettingsWidget *delaySettingsWidget;

    QLabel *qtaLabel;
    QTAInputWidget *qtaInputWidget;

public:
    explicit ObservableSettings(QWidget *parent = nullptr);
};

#endif
```

G.2.11 SamplingRateWidget.h

```
#pragma once

#include <QLabel>
#include <QPushButton>
#include <QSlider>
#include <QVBoxLayout>
#include <QWidget>

/**
 * @class SamplingRateWidget
 * @brief A widget for saving sampling  rate intervals.
 *
 * Provides a slider interface to select from predefined sampling
   rates
 * (in milliseconds) and emits the selected rate when saved.
 */
class SamplingRateWidget : public QWidget
{
    Q_OBJECT

public:
    explicit SamplingRateWidget(QWidget *parent = nullptr);

private Q_SLOTS:
    /**
     * @brief Handles slider value changes to update the displayed
   rate.
     * @param value The current slider index (0-based).
     */
```

```
    void onSliderValueChanged(int value);

    /**
     * @brief Handles save button clicks to emit the selected rate.
     */
    void onSaveClicked();

Q_SIGNALS:
    /**
     * @brief Emitted when a new sampling rate is saved.
     * @param milliseconds The selected sampling rate in milliseconds.
     */
    void onSamplingRateChanged(int milliseconds);

private:
    QSlider *slider; ///< Slider for selecting sampling rate
    QLabel *valueLabel; ///< Displays the current sampling rate
    QPushButton *saveButton; ///< Button to save the selected rate
    QVector<int> samplingRates; ///< Available sampling rate options (
    ms)
};
```

G.2.12 QTAInputWidget.h

```
#ifndef QTAINPUTWIDGET_H
#define QTAINPUTWIDGET_H

#include <QWidget>
#include <QLineEdit>
#include <QComboBox>
#include <QFormLayout>
#include <QLabel>
#include <QPushButton>

/**
 * @class QTAInputWidget
 * @brief A Qt widget for configuring QTAs for observables.
 */
class QTAInputWidget : public QWidget
{
    Q_OBJECT

public:
    explicit QTAInputWidget(QWidget *parent = nullptr);

    /**
     * @brief Gets the 25th percentile value (in seconds).
     * @return The value entered in the 25th percentile field.
     */
    double getPerc25() const;

    /**
     * @brief Gets the 50th percentile (median) value (in seconds).
     * @return The value entered in the 50th percentile field.
     */
```

```
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
    observables.
55       * @note Called automatically when the system updates.
56       */
57      void populateComboBox();
58
59      /**
60       * @brief Loads QTA settings for the selected observable into the
    UI fields.
61       * @note Triggered when the dropdown selection changes.
62       */
63      void loadObservableSettings();
64
65      /**
66       * @brief Saves the current QTA settings to the system.
67       * @note Called when the "Save" button is clicked.
68       * @throws std::exception if validation fails (e.g., invalid CDF
    value).
69       */
70      void onSaveButtonClicked();
71
72  private:
73      QComboBox *observableComboBox;   ///< Dropdown to select an
    observable (probe/outcome).
74      QLineEdit *perc25Edit;           ///< Input field for the 25th
    percentile (seconds).
75      QLineEdit *perc50Edit;           ///< Input field for the 50th
    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
```

G.2.13 Sidebar.h

```
1   #ifndef SIDEBAR_H
2   #define SIDEBAR_H
3
4   #include "DQPlotList.h"
5   #include "NewPlotList.h"
6   #include "SamplingRateWidget.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   */
23  class Sidebar : public QWidget
24  {
25      Q_OBJECT
26
27      QVBoxLayout *newPlotListLayout; ///< Layout for new plot selection components
28      QWidget *newPlotListWidget; ///< Container widget for new plot controls
29      QLabel *newPlotLabel; ///< Label for new plot section
30      NewPlotList *newPlotList; ///< List widget for selecting probes for new plots
31      QPushButton *addNewPlotButton; ///< Button to create new plot
32
33      QWidget *currentPlotWidget; ///< Container widget for current plot controls
34      QVBoxLayout *currentPlotLayout; ///< Layout for current plot components
35      QLabel *currentPlotLabel; ///< Label for current plot section
36      DQPlotList *currentPlotList = nullptr; ///< List widget for managing current plot's probes
37
38      QSplitter *mainSplitter; ///< Main splitter organizing sections vertically
39      QVBoxLayout *layout; ///< Main layout of the sidebar
```

```
40
41      SystemCreationWidget *systemCreationWidget; ///< Widget for system
        creation/configuration
42      SamplingRateWidget *samplingRateWidget; ///< Widget for adjusting
       sampling rate
43
44  Q_SIGNALS:
45      /**
46       * @brief Emitted when the "Add plot" button is clicked.
47       */
48      void addPlotClicked();
49
50      /**
51       * @brief Emitted when sampling rate is changed.
52       * @param milliseconds The new sampling rate in milliseconds.
53       */
54      void onSamplingRateChanged(int milliseconds);
55
56  private Q_SLOTS:
57      /**
58       * @brief Handles "Add plot" button click event.
59       */
60      void onAddPlotClicked();
61
62      /**
63       * @brief Handles sampling rate change events.
64       * @param ms The new sampling rate in milliseconds.
65       */
66      void handleSamplingRateChanged(int ms);
67
68  public:
69      /**
70       * @brief Constructs a Sidebar widget.
71       * @param parent The parent widget (optional).
72       */
73      explicit Sidebar(QWidget *parent = nullptr);
74
75      /**
76       * @brief Sets the current plot list widget.
77       * @param currentPlotList The DQPlotList widget to display.
78       */
79      void setCurrentPlotList(DQPlotList *currentPlotList);
80
81      /**
82       * @brief Hides the current plot management section.
83       */
84      void hideCurrentPlot();
85
86      /**
87       * @brief Gets the new plot list widget.
88       * @return Pointer to the NewPlotList widget.
89       */
90      NewPlotList *getPlotList() const
91      {
92          return newPlotList;
93      }
```

```
94
95      /**
96       * @brief Clears new plot selection after plot creation.
97       */
98      void clearOnAdd();
99  };
100
101 #endif
```

G.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
        triggers.
14   */
15  class SnapshotViewerWindow : public QWidget
16  {
17      Q_OBJECT
18
19  public:
20      explicit SnapshotViewerWindow(std::vector<Snapshot> &snapshotList,
        QWidget *parent = nullptr);
21
22      /**
23       * @brief Sets the snapshots to display in the viewer.
24       * @param snapshotList A vector of Snapshots.
25       */
26      void setSnapshots(std::vector<Snapshot> &snapshotList);
27
28  private 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
    and time index.
44      */
45     void updatePlot();
46 
47     QChartView *chartView;                      ///< Chart view for
    plotting the snapshot data.
48     QComboBox *observableSelector;              ///< Dropdown for
    selecting an observable.
49     QSlider *timeSlider;                        ///< Slider for
    selecting time index.
50     QLabel *timeLabel;                          ///< Label showing the
    current time.
51 
52     std::map<std::string, Snapshot> snapshots;///< Map of observable
    name to snapshot.
53     std::string currentObservable;              ///< Currently selected
    observable.
54 };
```

G.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  };
```

G.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
        system definitions.
14   */
15  class SystemCreationWidget : public QWidget
16  {
17      Q_OBJECT
18
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
    text.
68     QPushButton *updateSystemButton;    ///< Button to update system.
69     QPushButton *saveSystemButton;      ///< Button to save system.
70     QPushButton *loadSystemButton;      ///< Button to load system.
71     QLabel *systemLabel;                ///< Label describing the
    editor.
72
73     QVBoxLayout *mainLayout;            ///< Layout for the main
    components.
74     QHBoxLayout *buttonLayout;          ///< Layout for the buttons.
75 };
76
77 #endif // SYSTEMCREATIONWIDGET_H
```

G.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>
```

```cpp
#include <QVBoxLayout>
#include <QWidget>

#include "../Application.h"
#include "src/maths/TriggerManager.h"

/**
 * @class TriggersTab
 * @brief Widget for managing and monitoring trigger conditions on observables.
 *
 * Provides UI for:
 * - Setting up trigger conditions (sample limits, QTA violations)
 * - Displaying triggered events
 * - Viewing snapshots of triggered states
 */
class TriggersTab : public QWidget
{
    Q_OBJECT

public:
    explicit TriggersTab(QWidget *parent = nullptr);

    ~TriggersTab();

    /**
     * @brief Adds a triggered message to the display list.
     * @param msg The message to display.
     */
    void addTriggeredMessage(const QString &msg);

private Q_SLOTS:
    /**
     * @brief Handles observable selection changes.
     * @param name The newly selected observable name.
     */
    void onObservableChanged(const QString &name);

    /**
     * @brief Handles trigger condition changes.
     */
    void onTriggerChanged();

    /**
     * @brief Handles triggered item clicks to show snapshots.
     * @param item The clicked list item.
     */
    void onTriggeredItemClicked(QListWidgetItem *item);

private:
    /**
     * @brief Captures snapshots when triggers are activated.
     * @param time The timestamp of the trigger event.
     * @param name The name of the observable that triggered.
     */
```

```
63      void captureSnapshots(std::uint64_t time, const std::string &name)
        ;
64
65      /**
66       * @brief Populates the observable dropdown list.
67       */
68      void populateObservables();
69
70      /**
71       * @brief Updates checkbox states based on current triggers.
72       */
73      void updateCheckboxStates();
74
75      QVBoxLayout *mainLayout;              ///< Main vertical layout
76      QFormLayout *formLayout;              ///< Form layout for controls
77      QComboBox *observableComboBox;        ///< Dropdown for observable
        selection
78
79      QWidget *sampleLimitWidget;           ///< Container for sample
        limit controls
80      QHBoxLayout *sampleLimitLayout;       ///< Layout for sample limit
        controls
81      QCheckBox *sampleLimitCheckBox;       ///< Checkbox to enable
        sample limit trigger
82      QSpinBox *sampleLimitSpinBox;         ///< Spinbox for sample limit
        threshold
83
84      QCheckBox *qtaBoundsCheckBox;         ///< Checkbox for QTA bounds
        violation trigger
85
86      QListWidget *triggeredList;           ///< List widget for
        triggered events
87
88      /**
89       * @brief Gets the currently selected observable.
90       * @return Shared pointer to the current observable.
91       * @throws std::runtime_error if system or observable doesn't
        exist.
92       */
93      std::shared_ptr<Observable> getCurrentObservable();
94
95      static constexpr int sampleLimitThreshold = 500;      ///< Default
        sample limit threshold
96      static constexpr double failureRateThreshold = 0.95;  ///< Failure
        rate threshold constant
97  };
```

G.3 diagram

G.3.1 Observable.h

```
1   #pragma once
2
3   #include "../maths/Snapshot.h"
```

```cpp
#include "Sample.h"
#include "src/maths/TriggerManager.h"
#include <deque>
#include <math.h>
#include <mutex>

#define DELTA_T_BASE 0.001
#define MAX_DQ 30
class Observable
{
protected:
    std::string name;
    std::deque<Sample> samples;
    mutable bool sorted;

    std::deque<DeltaQ> confidenceIntervalHistory;
    double maxDelay {0.05};
    int deltaTExp {0}; // Exponent for dynamic binning
    int nBins {50}; // Number of bins

    TriggerManager triggerManager;

    ConfidenceInterval observedInterval;
    QTA qta;

    Snapshot observableSnapshot;

    std::mutex observedMutex;
    std::mutex samplesMutex;
    std::mutex paramMutex;

    bool recording = false;

private:
    void updateSnapshot(uint64_t timeLowerBound, DeltaQ &deltaQ);

public:
    /**
     * @brief Constructor an observable with its name
     */
    Observable(const std::string &name);

    virtual ~Observable() { };
    /**
     * Add a sample (outcome instance) to an observable
     */
    void addSample(const Sample &sample);
    /**
     * @brief Get all sample with endTime timeLowerBound - timeUpperBound
     * @return The sample in range timeLowerBound - timeUpperBound
     */
    std::vector<Sample> getSamplesInRange(std::uint64_t timeLowerBound, std::uint64_t timeUpperBound);
    /**
```

```
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 evalute
123      * @param action action to perform on trigger fired
124      * @param enabled
125      * @param sampleLimit sampleLimit sample limit for sample limit
    trigger
126      */
127     void addTrigger(TriggerType type, TriggerDefs::Condition condition
    , TriggerDefs::Action action, bool enabled, std::optional<int>
    sampleLimit);
128     /**
129      * @brief Remove observable trigger
130      */
131     void removeTrigger(TriggerType type);
132 
133     /**
134      * @brief Get snapshot of observable
135      */
136     Snapshot getSnapshot();
137     /**
138      * @brief Get observable name
139      * @return its name
140      */
141     [[nodiscard]] std::string getName() const &;
142 };
```

G.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
       DeltaQSD paradigm
9   */
10 class Operator : public Observable
11 {
12     OperatorType type;
13 
14     std::vector<double> probabilities; ///< The probabilities of the
    components inside the operator, only available for probabilistic
```

```
    operator
15
16      std::vector<std::vector<std::shared_ptr<Observable>>> causalLinks;
        ///< The causal links for each children
17
18      ConfidenceInterval calculatedInterval;
19
20      std::mutex calcMutex;
21
22      std::deque<DeltaQ> calculatedDeltaQHistory; ///< History for
    confidence intervals
23
24  public:
25      Operator(const std::string &name, OperatorType);
26
27      ~Operator();
28      /**
29       * @brief calculate a Calculated deltaQ with bounds timeLowerBound
    , timeUpperBound
30       * @param timeLowerBound
31       * @param timeUpperBound
32       * @return calculated DeltaQ
33       */
34      DeltaQ calculateCalculatedDeltaQ(uint64_t, uint64_t);
35
36      /**
37       * @brief Get the representation of a calculated DeltaQ for
    plotting
38       * @return the representation of a calculated DeltaQ
39       */
40      DeltaQRepr getCalculatedDeltaQRepr(uint64_t, uint64_t);
41
42      void setProbabilities(const std::vector<double> &);
43
44      /**
45       * @brief Get the probabilities of a probabilistic operator
46       * @return The probabilities
47      */
48      std::vector<double> getProbabilities()
49      {
50          return probabilities;
51      }
52
53      /**
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<
    Observable>>> links)
60      {
61          causalLinks = links;
62      }
63
```

```
64      std::vector<std::vector<std::shared_ptr<Observable>>>
        getCausalLinks()
65      {
66          return causalLinks;
67      }
68
69      OperatorType getType()
70      {
71          return type;
72      }
73  };
```

G.3.3 OperatorType.h

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

G.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 };
```

G.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
    , timeUpperBound
36      * @param timeLowerBound
37      * @param timeUpperBound
38      * @return calculated DeltaQ
39      */
40     DeltaQ calculateCalculatedDeltaQ(uint64_t timeLowerBound, uint64_t
     timeUpperBound);
41
42     /**
43      * @brief Get the representation of a calculated DeltaQ for
    plotting
44      * @return the representation of a calculated DeltaQ
45      */
46     DeltaQRepr getCalculatedDeltaQRepr(uint64_t, uint64_t);
47
48     std::vector<Bound> getBounds() const;
49
50     std::vector<Bound> getObservedBounds() const;
51
52     std::vector<Bound> getCalculatedBounds() const;
53
54
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 };
```

G.3.6 Sample.h

This struct represents an outcome instance.

```
#pragma once

#include <cstdint>

enum Status { SUCCESS, TIMEDOUT, FAILED };

/**
 * @struct Sample represent an outcome instance
 */
struct Sample {
    std::uint64_t startTime;
    std::uint64_t endTime;
    double long elapsedTime;
    Status status;
};
```

G.3.7 System.h

```
/**
 * @author Francesco Nieri
 * @date 26/10/2024
 * Class representing a DeltaQ system
 */
#pragma once

#include "../maths/DeltaQ.h"
#include "Observable.h"
#include "Operator.h"
#include "Outcome.h"
#include "Probe.h"
#include <memory>
#include <unordered_map>

class System
{
    std::unordered_map<std::string, std::shared_ptr<Outcome>> outcomes
    {}; ///< All outcome
    std::unordered_map<std::string, std::shared_ptr<Operator>>
   operators {}; ///< All operators
    std::unordered_map<std::string, std::shared_ptr<Probe>> probes {};
    /// < All probes
    std::unordered_map<std::string, std::shared_ptr<Observable>>
   observables {}; ///< The above grouped together

    std::string systemDefinitionText; ///< The definition of the
   system

    bool recordingTrigger = false;
    std::map<uint64_t, std::vector<Snapshot>> snapshots;

public:
```

```
29      System() = default;
30
31      [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<
        Outcome>> &getOutcomes();
32
33      [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<
        Probe>> &getProbes();
34
35      [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<
        Operator>> &getOperators();
36
37      [[nodiscard]] std::unordered_map<std::string, std::shared_ptr<
        Observable>> &getObservables();
38
39      void setOutcomes(std::unordered_map<std::string, std::shared_ptr<
        Outcome>> outcomesMap);
40
41      void setOperators(std::unordered_map<std::string, std::shared_ptr<
        Operator>> operatorsMap);
42
43      void setProbes(std::unordered_map<std::string, std::shared_ptr<
        Probe>> probesMap);
44
45      bool hasOutcome(const std::string &name);
46
47      bool hasOperator(const std::string &name);
48
49      std::shared_ptr<Outcome> getOutcome(const std::string &outcomeName
        );
50
51      std::shared_ptr<Operator> getOperator(const std::string &);
52
53      bool hasProbe(const std::string &name);
54
55      std::shared_ptr<Probe> getProbe(const std::string &name);
56
57      std::shared_ptr<Observable> getObservable(const std::string &
        observableName);
58
59      void setSystemDefinitionText(std::string &text);
60
61      std::string getSystemDefinitionText();
62
63      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
        time t
79       */
80      void getObservablesSnapshotAt(std::uint64_t);
81
82      /**
83       * Get the snapshots of all observables for a trigger at time t
84       */
85      std::map<std::uint64_t, std::vector<Snapshot>> getAllSnapshots();
86
87      /**
88       * @brief Get the name of all components
89       */
90      std::vector<std::string> getAllComponentsName();
91
92      /**
93       * @deprecated This may be used in the future
94       * Calculate the resulting DeltaQ for the whole system
95       */
96      DeltaQ calculateDeltaQ();
97  };
```

G.4 ΔQ (maths)

G.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
```

```
    void updateConfidenceInterval();

public:
    /**
     * @brief Default constructor, set to 0 bins
     */
    ConfidenceInterval();
    /**
     * @brief Constructor for ConfidenceInterval with number of bins
     * @param numBins number of bins
     */
    ConfidenceInterval(int numBins);

    void setNumBins(int newNumBins);
    /**
     * @brief Add DeltaQ to intervals
     * @param DeltaQ DeltaQ to add
     */
    void addDeltaQ(const DeltaQ &);
    /**
     * @brief Remove DeltaQ from intervals
     * @param DeltaQ DeltaQ to remove
     */
    void removeDeltaQ(const DeltaQ &);

    /**
     * @brief Get current confidence bounds
     */
    std::vector<Bound> getBounds() const;
    /**
     * @brief Get number of bins
     * @return Number of bins
     */
    unsigned int getBins();
    /**
     * @brief Zero the confidence intervals
     */
    void reset();
};
```

G.4.2 DeltaQ.h

```
/**
 * @author: Francesco Nieri
 * @date 26/10/2024
 * Class representing a DeltaQ
 */

#ifndef DELTAQ_H
#define DELTAQ_H

#pragma once

#include "../diagram/Sample.h"
#include <array>
```

```cpp
#include <ostream>
#include <vector>

#include "QTA.h"

class DeltaQ
{
    double binWidth;
    std::vector<double> pdfValues;
    std::vector<double> cdfValues;
    int bins {0};

    QTA qta;
    unsigned int totalSamples {0};

    /**
     * Calculate PDF and CDF values given samples from an outcome
     */
    void calculateDeltaQ(std::vector<Sample> &samples);
    /**
     * Calculate CDF given PDF values
     */
    void calculateCDF();
    /**
     * Calculate PDF from a given CDF
     */
    void calculatePDF();

public:
    DeltaQ() = default;
    DeltaQ(double binWidth);
    DeltaQ(double binWidth, const std::vector<double> &values, bool
    isPdf);
    DeltaQ(double binWidth, std::vector<Sample> &);
    DeltaQ(double binWidth, std::vector<Sample> &, int);
    /**
     * Getters
     */
    [[nodiscard]] const std::vector<double> &getPdfValues() const;
    [[nodiscard]] const std::vector<double> &getCdfValues() const;
    [[nodiscard]] double getBinWidth() const;
    [[nodiscard]] int getBins() const;
    [[nodiscard]] double pdfAt(int x) const;
    [[nodiscard]] double cdfAt(int x) const;
    [[nodiscard]] const unsigned int getTotalSamples() const;
    [[nodiscard]] QTA getQTA() const;
    void calculateQuartiles(std::vector<Sample> &);
    void setBinWidth(double newWidth);
    /**
     * Operator Overloads
     */
    friend DeltaQ operator*(const DeltaQ &deltaQ, double constant);
    friend DeltaQ operator*(double constant, const DeltaQ &deltaQ);
    friend DeltaQ operator*(const DeltaQ &lhs, const DeltaQ &rhs);
    friend DeltaQ operator+(const DeltaQ &lhs, const DeltaQ &rhs);
    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
```

G.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 ΔQ_{LTF(A,B)} ΔQ_A * ΔQ_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   * DeltaQ_{FTF(A, B)} = DeltaQ_A + DeltaQ_B - DeltaQ_A * DeltaQ_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
       event
34   * O_A occurs with probability p/(p+q)
35   * O_B occurs with probability q/(p+q)
36   * Therefore:
```

```
 * ΔQ_{PC(A,B)} = p/(p+q) ΔQ_A + q/(p+q) ΔQ_B
 */
DeltaQ probabilisticChoice(const std::vector<double> &probabilities,
    const std::vector<DeltaQ> &deltaQs);

DeltaQ rebin(const DeltaQ &source, double targetBinWidth);

/**
 * Choose the highest size from a list of DeltaQs
 */
int chooseLongestDeltaQSize(const std::vector<DeltaQ> &deltaQs);
```

G.4.4 DeltaQRepr.h

```

#pragma once

#include "ConfidenceInterval.h"
#include "DeltaQ.h"
#include <cstdint>
#include <vector>

/**
 * Class storing a DeltaQ representation for graphical plotting
 */

struct DeltaQRepr {
    std::uint64_t time;
    DeltaQ deltaQ;
    std::vector<Bound> bounds;

    DeltaQRepr()
        : time(0)
        , deltaQ()
        , bounds()
    {
    }

    // DeltaQRepr copy constructor
    DeltaQRepr(const DeltaQRepr &other)
        : time(other.time)
        , deltaQ(other.deltaQ)
        , bounds(other.bounds)
    {
    }

    // DeltaQRepr move semantics
    DeltaQRepr(DeltaQRepr &&other) noexcept
        : time(other.time)
        , deltaQ(std::move(other.deltaQ))
        , bounds(std::move(other.bounds))
    {
    }

    // 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  };
```

G.4.5 QTA.h

This class represent a sample QTA.

```
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,
    bool isDefined)
23      {
24          if (!(p25 <= p50 && p50 <= p75)) {
25              throw std::invalid_argument("Percentiles must be ordered:
    perc_25 < perc_50 < perc_75.");
26          }
27          if (cdf < 0.0 || cdf - 1.0 > QTA_EPSILON) {
28              throw std::invalid_argument("cdfMax must be between 0 and
    1 (exclusive lower bound, inclusive upper).");
29          }
30
31          return QTA {p25, p50, p75, cdf, isDefined};
32      }
33  };
34
35  #endif // QTA_H
```

G.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
     values along with QTAs
12   */
13  class Snapshot
14  {
15      std::string observableName; ///< Name of the observable being
    tracked.
16      std::map<uint64_t, DeltaQRepr> observedDeltaQs; ///< Map of
    observed DeltaQ values at times t.
17      std::map<uint64_t, DeltaQRepr> calculatedDeltaQs; ///< Map of
    calculated DeltaQ values at times t
18      std::map<uint64_t, QTA> QTAs; ///< Map of QTAs at time t
19
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
    observed.
28       * @param deltaQ The DeltaQ value to store.
29       * @param bounds Confidence interval bounds for the DeltaQ.
30       */
```

```
31      void addObservedDeltaQ(std::uint64_t timestamp, const DeltaQ &
        deltaQ, const std::vector<Bound> &bounds);
32
33      /**
34       * @brief Retrieves the oldest observed DeltaQ.
35       * @return The oldest DeltaQRepr (based on timestamp order).
36       */
37      DeltaQ getOldestObservedDeltaQ() const;
38
39      /// @brief Removes the oldest observed DeltaQ entry (FIFO order).
40      void removeOldestObservedDeltaQ();
41
42      // --- Calculated DeltaQ Methods ---
43      /**
44       * @brief Adds a calculated DeltaQ to the snapshot.
45       * @param timestamp The lower time bound at which the DeltaQ was
        calculated.
46       * @param deltaQ The DeltaQ value to store.
47       * @param bounds Confidence interval bounds for the DeltaQ.
48       */
49      void addCalculatedDeltaQ(std::uint64_t timestamp, const DeltaQ &
        deltaQ, const std::vector<Bound> &bounds);
50
51      /**
52       * @brief Retrieves the oldest calculated DeltaQ.
53       * @return The oldest DeltaQRepr (based on timestamp order).
54       */
55      DeltaQ getOldestCalculatedDeltaQ() const;
56
57      /// @brief Removes the oldest calculated DeltaQ entry (FIFO order)
        .
58      void removeOldestCalculatedDeltaQ();
59
60      // --- QTA Methods ---
61      /**
62       * @brief Adds a QTA to the snapshot.
63       * @param timestamp The time associated with the QTA.
64       * @param qta The QTA value to store.
65       */
66      void addQTA(std::uint64_t timestamp, const QTA &qta);
67
68      // --- Size Management ---
69      /// @return The number of observed DeltaQs stored.
70      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
        size (removes oldest entries).
77       * @param size The maximum number of entries to retain.
78       */
79      void resizeTo(size_t size);
80
81      // --- Name Management ---
```

```
82      /**
83       * @brief Sets the name of the observable.
84       * @param name New name for the observable.
85       */
86      void setName(const std::string &name);
87
88      /// @return The current observable name.
89      std::string getName() &;
90
91      // --- Data Retrieval ---
92      /// @return A vector of all observed DeltaQReprs (sorted by
    timestamp).
93      std::vector<DeltaQRepr> getObservedDeltaQs() const &;
94
95      /// @return A vector of all calculated DeltaQReprs (sorted by
    timestamp).
96      std::vector<DeltaQRepr> getCalculatedDeltaQs() const &;
97
98      /**
99       * @brief Retrieves an observed DeltaQ at a specific timestamp.
100      * @param timestamp The time to query.
101      * @return The DeltaQRepr if found, or `std::nullopt` otherwise.
102      */
103     std::optional<DeltaQRepr> getObservedDeltaQAtTime(std::uint64_t
    timestamp);
104
105     /**
106      * @brief Retrieves a calculated DeltaQ at a specific timestamp.
107      * @param timestamp The time to query.
108      * @return The DeltaQRepr if found, or `std::nullopt` otherwise.
109      */
110     std::optional<DeltaQRepr> getCalculatedDeltaQAtTime(std::uint64_t
    timestamp);
111
112     /// @return A vector of all QTAs (sorted by timestamp).
113     std::vector<QTA> getQTAs() const &;
114 };
```

G.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  };
```

G.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
```

G.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  }
```

G.5 parser

G.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
       tree.
11   */
12  class SystemBuilderVisitor : public parser::DQGrammarVisitor
13  {
14  private:
15      std::unordered_map<std::string, std::shared_ptr<Outcome>> outcomes
       ;   ///< Map of outcome names to Outcome objects.
16      std::unordered_map<std::string, std::shared_ptr<Operator>>
       operators; ///< Map of operator names to Operator objects.
17      std::unordered_map<std::string, std::shared_ptr<Probe>> probes;
           ///< Map of probe names to Probe objects.
18
19      std::vector<std::string> definedProbes; ///< List of defined probe
        names.
20      System system; ///< The final system constructed by the visitor.
21
22      std::string currentlyBuildingProbe; ///< Tracks the probe
       currently being built for dependency management.
23      std::map<std::string, std::vector<std::string>> dependencies; ///<
        Graph of probe dependencies.
24
25      std::unordered_map<std::string, std::vector<std::string>>
       definitionLinks; ///< For debugging: links between definitions.
26      std::unordered_map<std::string, std::vector<std::vector<std::
       string>>> operatorLinks; ///< For debugging: operator chains.
27      std::vector<std::string> systemLinks; ///< Top-level system
       observable links.
28      std::unordered_set<std::string> allNames; ///< Tracks all used
       names to detect duplicates.
29
30      /**
31       * @brief Checks the dependency graph for cycles.
32       * @throws std::invalid_argument if a cycle is detected.
33       */
34      void checkForCycles() const;
35
```

```
36      /**
37       * @brief Recursive utility to detect cycles in a graph.
38       * @param node Current node.
39       * @param visited Set of visited nodes.
40       * @param recursionStack Stack of nodes in the current DFS path.
41       * @return True if a cycle is found.
42       */
43      bool hasCycle(const std::string& node,
44                    std::set<std::string>& visited,
45                    std::set<std::string>& recursionStack) const;
46
47  public:
48      /**
49       * @brief Returns the constructed System.
50       * @return The system object.
51       */
52      System getSystem() const;
53
54      // Visitor overrides from DQGrammarVisitor:
55      std::any visitStart(parser::DQGrammarParser::StartContext *context
    ) override;
56      std::any visitDefinition(parser::DQGrammarParser::
    DefinitionContext *context) override;
57      std::any visitSystem(parser::DQGrammarParser::SystemContext *
    context) override;
58      std::any visitComponent(parser::DQGrammarParser::ComponentContext
    *context) override;
59      std::any visitBehaviorComponent(parser::DQGrammarParser::
    BehaviorComponentContext *context) override;
60      std::any visitProbeComponent(parser::DQGrammarParser::
    ProbeComponentContext *context) override;
61      std::any visitProbability_list(parser::DQGrammarParser::
    Probability_listContext *context) override;
62      std::any visitComponent_list(parser::DQGrammarParser::
    Component_listContext *context) override;
63      std::any visitOutcome(parser::DQGrammarParser::OutcomeContext *
    context) override;
64      std::any visitComponent_chain(parser::DQGrammarParser::
    Component_chainContext *context) override;
65  };
66
67  #endif // SYSTEMBUILDER_H
```

G.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 *
    offendingSymbol, size_t line, size_t charPositionInLine, const std
```

```
    ::string &msg,
10          std::exception_ptr e) override
11      {
12          throw std::runtime_error("Syntax error at line " + std::
    to_string(line) + ":" + std::to_string(charPositionInLine) + " - "
     + msg);
13      }
14  };
```

G.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
       files or strings.
15   */
16  class SystemParserInterface
17  {
18  public:
19      /**
20       * @brief Parses a system definition from a file.
21       * @param filename Path to the file containing system definition.
22       * @return Optional containing the parsed System if successful,
    nullopt otherwise.
23       */
24      static std::optional<System> parseFile(const std::string &filename
    );
25
26      /**
27       * @brief Parses a system definition from a string.
28       * @param input String containing system definition.
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  };
```

G.6 server

G.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 =
    8080);
39
40      void stopServer();
41
42      bool setErlangEndpoint(const std::string &ip, int port);
43
44      bool isServerRunning() const
```

```
45      {
46          return server_started;
47      }
48
49      /**
50       * @brief Stops the server and worker threads.
51       */
52      void stop();
53
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
        monitored
80
81      std::vector<std::thread> clientThreads; ///< Active client handler
        threads
82      std::mutex clientsMutex; ///< Mutex for client threads access
83      std::atomic<bool> running {false}; ///< Server running state flag
84
85      /**
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 H

Build Configuration Files

H.1 src/CMakeLists.txt

```
add_library(${PREFIX}_application
    Application.cpp
    Application.h
)


target_link_libraries(${PREFIX}_application
    PRIVATE
    ${PREFIX}_server
)

# Make sure other libraries can find Application headers
target_include_directories(${PREFIX}_application
    PUBLIC ${CMAKE_SOURCE_DIR}
)

add_subdirectory(dashboard)
add_subdirectory(diagram)
add_subdirectory(maths)
add_subdirectory(server)
add_subdirectory(parser)

add_executable(RealTimeDeltaQSD main.cpp)

target_include_directories(RealTimeDeltaQSD PUBLIC ${CMAKE_SOURCE_DIR
    })

target_link_libraries(RealTimeDeltaQSD ${PREFIX}_server ${PREFIX}
    _diagram ${PREFIX}_dashboard ${PREFIX}_parser)
```

H.2 dashboard/CMakeLists.txt

```
add_library(${PREFIX}_dashboard
    ColorRegistry.cpp
```

```
    ColorRegistry.h
    CustomLegendEntry.h
    CustomLegendEntry.cpp
    CustomLegendPanel.h
    CustomLegendPanel.cpp
    DelaySettingsWidget.h
    DelaySettingsWidget.cpp
    DQPlotController.h
    DQPlotController.cpp
    DQPlotList.h
    DQPlotList.cpp
    DeltaQPlot.h
    DeltaQPlot.cpp
    MainWindow.h
    MainWindow.cpp
    NewPlotList.h
    NewPlotList.cpp
    ObservableSettings.h
    ObservableSettings.cpp
    SamplingRateWidget.h
    SamplingRateWidget.cpp
    Sidebar.h
    Sidebar.cpp
    SnapshotViewerWindow.h
    SnapshotViewerWindow.cpp
    StubControlWidget.h
    StubControlWidget.cpp
    SystemCreationWidget.h
    SystemCreationWidget.cpp
    QTAInputWidget.cpp
    QTAInputWidget.h
    TriggersTab.cpp
    TriggersTab.h
)


find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS Core Gyu Widgets
   Charts Graphs)
find_package(Qt6 REQUIRED COMPONENTS Core Gui Charts Widgets Graphs)

target_link_libraries(${PREFIX}_dashboard
        PUBLIC Qt6::Core Qt6::Gui Qt6::Widgets Qt6::Charts Qt6::Graphs
        ${PREFIX}_parser
    )

target_include_directories(${PREFIX}_dashboard
        PUBLIC
        ${CMAKE_SOURCE_DIR}
        ${PREFIX}_parser
)
```

H.3 diagram/CMakeLists.txt

```
add_library(${PREFIX}_diagram
```

```
    Outcome.h
    Outcome.cpp
    Operator.h
    Operator.cpp
    Probe.h
    Probe.cpp
    System.h
    System.cpp
    Sample.h
    Observable.h
    Observable.cpp
)

target_link_libraries(${PREFIX}_diagram
    PUBLIC
        ${PREFIX}_maths
        ${PREFIX}_application
)


target_include_directories(${PREFIX}_diagram
        PUBLIC
        ${CMAKE_SOURCE_DIR}
)

add_executable(diagram
        main.cpp
)

target_link_libraries(diagram
        PRIVATE
        ${PREFIX}_diagram
)
```

H.4 maths/CMakeLists.txt

```
add_library(${PREFIX}_maths
    ConfidenceInterval.h
    ConfidenceInterval.cpp
    DeltaQ.h
    DeltaQ.cpp
    DeltaQRepr.h
    DeltaQOperations.h
    DeltaQOperations.cpp
    QTA.h
    Triggers.h
    Triggers.cpp
    TriggerTypes.h
    TriggerManager.cpp
    TriggerManager.h
    Snapshot.h
    Snapshot.cpp
)

```

```
target_include_directories(${PREFIX}_maths
        PUBLIC
        ${CMAKE_SOURCE_DIR}
)

target_link_libraries(${PREFIX}_maths
    PRIVATE
    fftw3
)

add_executable(maths
        main.cpp
)


target_link_libraries(maths
        PRIVATE
        ${PREFIX}_maths
)
```

H.5 parser/CMakeLists.txt

```

find_package(antlr4-runtime REQUIRED CONFIG)

if(NOT antlr4-runtime_FOUND)
    # Manually specify paths (adjust according to your installation)
    set(ANTLR4_INCLUDE_DIR "/usr/local/include")
    set(ANTLR4_LIB_DIR "/usr/local/lib")

    find_library(ANTLR4_RUNTIME_LIB antlr4-runtime PATHS ${
   ANTLR4_LIB_DIR})
    if(NOT ANTLR4_RUNTIME_LIB)
        message(FATAL_ERROR "ANTLR4 runtime library not found")
    endif()

    add_library(antlr4-runtime SHARED IMPORTED)
    set_target_properties(antlr4-runtime PROPERTIES
        IMPORTED_LOCATION ${ANTLR4_RUNTIME_LIB}
        INTERFACE_INCLUDE_DIRECTORIES ${ANTLR4_INCLUDE_DIR}
    )
endif()

set(ANTLR_GENERATED_DIR ${CMAKE_CURRENT_BINARY_DIR}/generated)
file(MAKE_DIRECTORY ${ANTLR_GENERATED_DIR})

# Generate ANTLR files
find_program(ANTLR_EXECUTABLE antlr4)

execute_process(
    COMMAND ${ANTLR_EXECUTABLE}
        -Dlanguage=Cpp
        -visitor -no-listener
        -o ${ANTLR_GENERATED_DIR}
```

```
        -package parser
        ${CMAKE_CURRENT_SOURCE_DIR}/DQGrammar.g4
    WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
    OUTPUT_VARIABLE ANTLR_OUTPUT
    ERROR_VARIABLE ANTLR_ERROR
    RESULT_VARIABLE ANTLR_RESULT
)
if(NOT ANTLR_RESULT EQUAL 0)
    message(FATAL_ERROR "ANTLR generation failed: ${ANTLR_ERROR}")
endif()

message(STATUS "ANTLR generated: ${ANTLR_OUTPUT}")

add_library(${PREFIX}_parser
    ${ANTLR_GENERATED_DIR}/DQGrammarLexer.h
    ${ANTLR_GENERATED_DIR}/DQGrammarLexer.cpp
    ${ANTLR_GENERATED_DIR}/DQGrammarParser.h
    ${ANTLR_GENERATED_DIR}/DQGrammarParser.cpp
    ${ANTLR_GENERATED_DIR}/DQGrammarBaseVisitor.cpp
    ${ANTLR_GENERATED_DIR}/DQGrammarVisitor.cpp
    ${ANTLR_GENERATED_DIR}/DQGrammarBaseVisitor.h
    ${ANTLR_GENERATED_DIR}/DQGrammarVisitor.h
     SystemBuilder.h
     SystemBuilder.cpp
     SystemErrorListener.h
     SystemParserInterface.h
     SystemParserInterface.cpp
)
target_link_libraries(${PREFIX}_parser
    PUBLIC antlr4-runtime
)

target_include_directories(${PREFIX}_parser
    PUBLIC
    ${CMAKE_SOURCE_DIR}
    ${ANTLR_GENERATED_DIR}
    ${ANTLR4_INCLUDE_DIR}
)

add_executable(parser
    main.cpp
)

target_link_libraries(parser
    PRIVATE
    ${PREFIX}_parser
    antlr4-runtime
 ${PREFIX}_diagram
)
```

H.6 server/CMakeLists.txt

```
add_library(${PREFIX}_server
    Server.cpp
```

```
    Server.h
)

target_link_libraries(${PREFIX}_server
    PUBLIC
        ${PREFIX}_diagram
    )

target_include_directories(${PREFIX}_server
    PUBLIC
    ${CMAKE_SOURCE_DIR}
)
```

Appendix I

Erlang Source Files

I.1 Root

I.1.1 dqsd_otel.erl

The ΔQ adapter, it can start, fail, end spans and start and end with_spans, communicates to the TCP client to send outcome instances to the oscilloscope.

```
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
       /1, with_span/2, with_span/3, span_process/3]).
7  -export([start/2, stop/1]).
8  -export([init_ets/0]).
9  -export([set_stub_running/1]).
10 -export([handle_c_message/1]).
11
12 -include_lib("opentelemetry_api/include/otel_tracer.hrl").
13
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,
       uses a custom timeout defined by the user in the oscilloscope,
18 %%
19 %% Features:
20 %% - Span lifecycle management (start/end/fail/timeout)
21 %% - Dynamic timeouts for spans
22 %% - Supports toggling stub behavior at runtime
23 %%
24 %% Usage:
25 %%   {Ctx, Pid} = dqsd_otel:start_span(<<"my_span">>).
26 %%   dqsd_otel:end_span(Ctx, Pid).
27 %%   dqsd_otel:fail_span(Pid)
28
29
```

```erlang
%%%=======================
%%% Application Callbacks
%%%=======================


start(_Type, _Args) ->
    dqsd_otel_sup:start_link().

stop(_State) ->
    ok.

init_ets() ->
    ets:new(timeout_registry, [named_table, public, set]),
    ets:new(otel_state, [named_table, public, set]),
    ets:insert(otel_state, {stub_running, false}),
    {ok, self()}.


%%%=======================
%%% For testing purposes
%%% =======================

set_stub_running(Bool) when is_boolean(Bool) ->
    ets:insert(otel_state, {stub_running, Bool}),
    io:format("Stub running set to: ~p~n", [Bool]),
    ok.

%%%=======================
%%% Public API
%%%=======================
%% @doc Starts the otel_wrapper application and all dependencies.
-spec start() -> {ok, [atom()]} | {error, term()}.
start() ->
    application:ensure_all_started(dqsd_otel).

%% @doc Starts a span with the given name, if the stub is running.
%% Returns a tuple of SpanContext and the internal span process PID or
    `ignore`.
-spec start_span(binary()) -> {opentelemetry:span_ctx(), pid() |
   ignore}.
start_span(Name) ->
    SpanCtx = ?start_span(Name),
    case ets:lookup(otel_state, stub_running) of
        [{_, true}] ->
            case ets:lookup(timeout_registry, Name) of
                [{_, T}] ->
                    StartTime = erlang:system_time(nanosecond),
                    Pid = spawn(?MODULE, span_process, [Name,
   StartTime, T]),
                    {SpanCtx, Pid};
                [] ->
                    {SpanCtx, ignore}
            end;
        _ ->
            {SpanCtx, ignore}
    end.
```

```
83
84  %% @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,
        StartTime, T]),
94                      {SpanCtx, Pid};
95                  [] ->
96                      {SpanCtx, ignore}
97              end;
98          _ ->
99              {SpanCtx, ignore}
100     end.
101
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
113 %% @doc Fail the span and reports it to the oscilloscope, unless stub
        is disabled or Pid is `ignore`.
114 -spec fail_span( pid() | ignore) -> ok | term().
115 fail_span(Pid) ->
116     case Pid of
117         ignore -> ok;
118     _ when is_pid(Pid) ->
119         Pid ! {<<"fail_span">>, erlang:system_time(nanosecond)}
120     end.
121
122
123
124 %% @doc Executes Fun inside a span with attributes.
125 -spec with_span(binary(), fun(() -> any())) -> any().
126 with_span(Name, Fun) ->
127     ?with_span(Name, #{},
128         fun(_SpanCtx) ->
129             Pid = start_with_span(Name),
130             Result = Fun(),
131             end_with_span(Pid),
132             Result
133         end).
```

```erlang

%% @doc Executes Fun inside a span with attributes.
-spec with_span(binary(), fun(() -> any()), map()) -> any().
with_span(Name, Fun, Attrs) when is_map(Attrs), is_function(Fun, 0) ->
    ?with_span(Name, Attrs,
        fun(_SpanCtx) ->
            Pid = start_with_span(Name),
            Result = Fun(),
            end_with_span(Pid),
            Result
        end).

start_with_span(Name) ->
    case ets:lookup(otel_state, stub_running) of
        [{_, true}] ->
            case ets:lookup(timeout_registry, Name) of
                [{_, T}] ->
                    StartTime = erlang:system_time(nanosecond),
                    Pid = spawn(?MODULE, span_process, [Name, StartTime, T]),
                    Pid;
                [] ->
                    ignore
            end;
        _ ->
            ignore
    end.

end_with_span(Pid) ->
        case Pid of
            ignore -> ok;
        _ when is_pid(Pid) ->
            Pid ! {<<"end_span">>, erlang:system_time(nanosecond)}
    end.



%%%=======================
%%% Span Worker
%%%=======================

span_process(NameBin, StartTime, Timeout) ->
    Deadline = StartTime + (Timeout * 1000000),
    Timer = erlang:send_after(Timeout, self(), {<<"timeout">>, Deadline}),
    receive
        {<<"fail_span">>, EndTime} ->
            io:format("failure"),
            erlang:cancel_timer(Timer),
            send_span(NameBin, StartTime, EndTime, <<"fa">>);
        {<<"end_span">>, EndTime} ->
            erlang:cancel_timer(Timer),
            send_span(NameBin, StartTime, EndTime, <<"ok">>);
        {<<"timeout">>, Deadline} ->
            send_span(NameBin, StartTime, Deadline, <<"to">>)
    end.
```

```
188
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", [
    Name, Timeout]);
201                 _ ->
202                     io:format("dqsd_otel: Invalid timeout: ~p~n", [
    TimeoutBin])
203             end;
204         [<<"start_stub">>] ->
205             ets:insert(otel_state, {stub_running, true}),
206             io:format("dqsd_otel: Stub enabled~n");
207         [<<"stop_stub">>] ->
208             ets:insert(otel_state, {stub_running, false}),
209             io:format("dqsd_otel: Stub stopped~n");
210         _ ->
211             io:format("dqsd_otel: Unknown command: ~p~n", [Bin])
212     end.
213
214 %%%=======================
215 %%% Sending Span Data to C
216 %%%=======================
217
218 send_span(NameBin, Start, End, StatusBin) ->
219     Data = io_lib:format("n:~s;b:~p;e:~p;s:~s~n", [
220         NameBin,
221         Start,
222         End,
223         StatusBin
224     ]),
225     dqsd_otel_tcp_client:send_span(lists:flatten(Data)).
```

I.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.
```

I.1.3 dqsd_otel_sup.erl

The supervisor of the adapter. It start the TCP server, client and adapter.

```
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}}.
```

I.1.4 dqsd_otel_tcp_client.erl

The TCP client, it can send outcome instances to the oscilloscope.

```
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,
      terminate/2]).
7       -export([disconnect/0, try_connect/2]).
```

```
8
9       -define(SERVER, ?MODULE).
10
11      -record(state, {
12          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
    connection.
25      -spec try_connect(string() | binary(), integer()) -> ok.
26      try_connect(IP, Port) ->
27          gen_server:cast(?SERVER, {try_connect, IP, Port}).
28
29      -spec disconnect() -> ok.
30      disconnect() ->
31          gen_server:cast(?SERVER, disconnect).
32
33      %%% gen_server callbacks
34
35      init([]) ->
36          State = #state{socket = undefined, logged_disconnected = false
    },
37          {ok, State}.
38
39      handle_cast({send, _Data}, State = #state{socket = undefined,
    logged_disconnected = false}) ->
40          io:format("dqsd_otel: No socket. Dropping subsequent spans.~n"
    ),
41          {noreply, State#state{logged_disconnected = true}};
42
43      handle_cast({send, _Data}, State = #state{socket = undefined}) ->
44          %% Already logged, suppress further logs
45          {noreply, State};
46
47      handle_cast({send, Data}, State = #state{socket = Socket}) ->
48          case gen_tcp:send(Socket, Data) of
49              ok ->
50                  {noreply, State};
51              {error, Reason} ->
52                  io:format("dqsd_otel: TCP send failed: ~p~n", [Reason]
    ),
53                  NewState = State#state{socket = undefined,
    logged_disconnected = true},
54                  {noreply, NewState}
55          end;
56
57      handle_cast({try_connect, IP, Port}, State) ->
```

```
58          IPStr = case IP of
59              Bin when is_binary(Bin) -> binary_to_list(Bin);
60              Str when is_list(Str)   -> Str
61          end,
62          case gen_tcp:connect(IPStr, Port, [binary, {active, false}])
    of
63              {ok, Socket} ->
64                  io:format("dqsd_otel: Adapter 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("dqsd_otel: 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("dqsd_otel: Socket already disconnected.~n"),
78          {noreply, State};
79
80      handle_cast(disconnect, State = #state{socket = Socket}) ->
81          io:format("dqsd_otel: 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.
```

I.1.5 dqsd_otel_tcp_server.erl

The TCP server accepts messages from the oscilloscope and forwards them to the adapter to set the various settings.

```
1 %%%-------------------------------------------------------------------
2 %%% @doc
3 %%% TCP server for receiving and processing messages from an
    oscilloscope.
```

```erlang
%%% Allows starting and stopping a TCP server on a given IP and port.
%%%
%%% When a line-delimited binary message is received, it is passed to
%%% `dqsd_otel:handle_c_message/1` for further processing.
%%%
%%%-------------------------------------------------------------------

-module(dqsd_otel_tcp_server).
-behaviour(gen_server).

%%% API
-export([start_link/0, start_server/2, stop_server/0]).

%%% gen_server callbacks
-export([init/1, handle_call/3, handle_cast/2, handle_info/2]).

-record(state, {
    socket,    %% Listening socket
    acceptor   %% PID of acceptor process
}).

-spec start_link() -> {ok, pid()} | ignore | {error, term()}.
start_link() ->
    gen_server:start_link({local, ?MODULE}, ?MODULE, [], []).

%%--------------------------------------------------------------------
%% @doc Starts the TCP listener on the given IP and Port.
%% Spawns an accept loop to handle incoming connections.
%%
%% IP can be a string, binary, or tuple (e.g., "127.0.0.1" or {127,0,0,1}).
%% Port is an integer.
%%--------------------------------------------------------------------

-spec start_server(string() | binary() | tuple(), integer()) -> ok | {error, term()}.
start_server(IP, Port) ->
    gen_server:call(?MODULE, {start, IP, Port}).

%%--------------------------------------------------------------------
%% @doc Stops the TCP server and closes the listening socket.
%% Also shuts down the acceptor process.
%%
%% @spec stop_server() -> ok | {error, not_running}
%%--------------------------------------------------------------------
stop_server() ->
    gen_server:call(?MODULE, stop).

%%--------------------------------------------------------------------
%% @private
%% gen_server init callback.
%% Initializes the state without an open socket or acceptor.
%%--------------------------------------------------------------------
init([]) ->
    {ok, #state{}}.

```

```erlang
%%--------------------------------------------------------------------
%% @private
%% Handles the start request. Binds to the given IP and port.
%% On success, starts the accept loop.
%%--------------------------------------------------------------------
handle_call({start, IP, Port}, _From, State) ->
    Options = [binary, {packet, line}, {active, false}, {reuseaddr, true}, {ip, parse_ip(IP)}],
    case gen_tcp:listen(Port, Options) of
        {ok, Socket} ->
            Acceptor = spawn(fun() -> accept_loop(Socket) end),
            io:format("dqsd_otel: Listening socket started on ~p:~p~n", [IP, Port]),
            {reply, ok, State#state{socket = Socket, acceptor = Acceptor}};
        {error, Reason} ->
            io:format("dqsd_otel: Could not start listening socket on ~p:~p~n", [IP, Port]),
            {reply, {error, Reason}, State}
    end;

%%--------------------------------------------------------------------
%% @private
%% Handles stop request. Stops acceptor and closes socket.
%% Returns error if server is not running.
%%--------------------------------------------------------------------
handle_call(stop, _From, #state{socket = undefined, acceptor = undefined} = State) ->
    io:format("dqsd_otel: Server not running.~n"),
    {reply, {error, not_running}, State};


handle_call(stop, _From, #state{socket = Socket, acceptor = Acceptor}) ->
    catch exit(Acceptor, shutdown),
    gen_tcp:close(Socket),
    io:format("dqsd_otel: Stopped listening from oscilloscope~n"),
    {reply, ok, #state{socket = undefined, acceptor = undefined}}.

handle_cast(_, State) -> {noreply, State}.
handle_info(_, State) -> {noreply, State}.

%%--------------------------------------------------------------------
%% @private
%% Accept loop that runs in a separate process.
%% Accepts TCP connections and spawns a handler for each one.
%%--------------------------------------------------------------------
accept_loop(ListenSocket) ->
    {ok, Socket} = gen_tcp:accept(ListenSocket),
    spawn(fun() -> handle_client(Socket) end),
    accept_loop(ListenSocket).

%%--------------------------------------------------------------------
%% @private
%% Handles a client socket connection. Reads line-by-line.
%% Forwards trimmed binary lines to `dqsd_otel:handle_c_message/1`.
```

```
%%--------------------------------------------------------------------
handle_client(Socket) ->
    case gen_tcp:recv(Socket, 0) of
        {ok, Line} ->
            Trimmed = binary:replace(Line, <<"\n">>, <<>>, [global]),
            dqsd_otel:handle_c_message(Trimmed),
            handle_client(Socket);
        {error, closed} ->
            gen_tcp:close(Socket)
    end.

%%--------------------------------------------------------------------
%% @private
%% Parses an IP address from string, binary, or already-parsed tuple.
%%--------------------------------------------------------------------
parse_ip("127.0.0.1") -> {127,0,0,1};
parse_ip("0.0.0.0") -> {0,0,0,0};
parse_ip(IP) when is_list(IP) ->
    {ok, Addr} = inet:parse_address(IP), Addr;
parse_ip(IP) -> IP.
```

Appendix J

Erlang Application Files

J.1 Root

J.1.1 dqsd_otel.app.src

The app.src file of the adapter.

```
{application, dqsd_otel,
    [{description, "An OpenTelemetry Wrapper for the ΔQ Oscilloscope"},
     {vsn, "git"},
     {modules, [dqsd_otel, dqsd_otel_sup, dqsd_otel_tcp_client, dqsd_otel_tcp_server]},
     {registered, [dqsd_otel_sup]},
     {applications, [kernel, stdlib, opentelemetry_exporter, opentelemetry, opentelemetry_api]},
     {mod, {dqsd_otel, []}},
     {env, []},
     {mantainers,["Francesco Nieri"]},
     {licenses,["Apache 2.0"]},
     {links,[{"https://github.com/fnieri/dqsd_otel"}]}]}.
```

Appendix K

Synthetic Applications

K.1 Synthetic applications

K.1.1 M/M/1/K queue application

```
-module(mm1k).
-export([start/2, start/3, send/1, worker_loop/4, stop/1,
    worker_buffer/5, set_lambda/1]).

-include_lib("opentelemetry_api/include/otel_tracer.hrl").

start(_Type, _Args) ->
    {ok, _} = application:ensure_all_started(opentelemetry),
    {ok, Pid} = example_sup:start_link(),
    {ok, Pid}.

set_lambda(NewX) when is_number(NewX), NewX > 0 ->
    ets:insert(send_rate, {lambda, NewX}),
    {ok, NewX}.

start(X, Y, K) ->
    Worker1Buffer = spawn_opt(fun() -> worker_buffer(worker_1,
    undefined, K, [], false) end, [{scheduler, 1}]),
    Worker2Buffer = spawn_opt(fun() -> worker_buffer(worker_2,
    undefined, K, [], false) end, [{scheduler, 1}]),
    Worker1 = spawn_opt(fun() -> worker_loop(worker_1, Y,
    Worker1Buffer, Worker2Buffer) end, [{scheduler, 1}]),
    Worker2 = spawn_opt(fun() -> worker_loop(worker_2, Y,
    Worker1Buffer, Worker2Buffer) end, [{scheduler, 1}]),

    ets:new(send_rate, [named_table, public, set]),
    ets:insert(send_rate, {lambda, X}),

    Worker1Buffer ! {set_worker, Worker1},
    Worker2Buffer ! {set_worker, Worker2},

    register(worker_1, Worker1Buffer),
    register(worker_2, Worker2Buffer),
    spawn(fun() -> send(Worker1Buffer) end),
```

```
30
31      {ok, self()}.
32
33  send(Worker1Buffer) ->
34      [{lambda, X}] = ets:lookup(send_rate, lambda),
35      Delay = -math:log(rand:uniform()) / X,
36      timer:sleep(trunc(Delay * 1000)),
37
38      B = maps:new(),
39      {ProbeCtx, Pid} = dqsd_otel:start_span(<<"probe">>),
40      B1 = maps:put(<<"probe_id">>, Pid, B),
41      Baggage = maps:put(<<"probe_ctx">>, ProbeCtx, B1),
42
43      Worker1Buffer ! Baggage,
44      send(Worker1Buffer).
45
46  worker_buffer(worker_1, WorkerPid, K, Queue, ServerBusy) ->
47      CurrentLoad = length(Queue) + if ServerBusy -> 1; true -> 0 end,
48      receive
49          {set_worker, NewPid} ->
50              worker_buffer(worker_1, NewPid, K, Queue, ServerBusy);
51
52          #{<<"probe_ctx">> := _} = Job when CurrentLoad < K ->
53              case ServerBusy of
54                  false ->
55                      {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_1">>),
56                      JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx
    , Job),
57                      JobWithSpan2 = maps:put(<<"worker_id">>,
    WorkerSpan, JobWithSpan),
58                      WorkerPid ! JobWithSpan2,
59                      worker_buffer(worker_1, WorkerPid, K, Queue, true)
    ;
60                  true ->
61                      NewQueue = Queue ++ [Job],
62                      worker_buffer(worker_1, WorkerPid, K, NewQueue,
    ServerBusy)
63              end;
64
65          #{<<"probe_ctx">> := _} = Job when CurrentLoad >= K ->
66              Pid = maps:get(<<"probe_id">>, Job, undefined),
67              {WorkerCtx, WrkPid} = dqsd_otel:start_span(<<"worker_2">>)
    ,
68              dqsd_otel:fail_span(Pid),
69              dqsd_otel:fail_span(WrkPid),
70              worker_buffer(worker_1, WorkerPid, K, Queue, ServerBusy);
71
72          done ->
73              case Queue of
74                  [] ->
75                      worker_buffer(worker_1, WorkerPid, K, [], false);
76                  [NextJob | RestQueue] ->
77                      {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_1">>),
```

```
78                      JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx
    , NextJob),
79                      JobWithSpan2 = maps:put(<<"worker_id">>,
    WorkerSpan, JobWithSpan),
80                      WorkerPid ! JobWithSpan2,
81                      worker_buffer(worker_1, WorkerPid, K, RestQueue,
    true)
82              end
83      end;
84
85  worker_buffer(worker_2, WorkerPid, K, Queue, ServerBusy) ->
86      CurrentLoad = length(Queue) + if ServerBusy -> 1; true -> 0 end,
87      receive
88          {set_worker, NewPid} ->
89              worker_buffer(worker_2, NewPid, K, Queue, ServerBusy);
90
91          #{<<"probe_ctx">> := _} = Job when CurrentLoad < K ->
92              case ServerBusy of
93                  false ->
94                      {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_2">>),
95                      JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx
    , Job),
96                      JobWithSpan2 = maps:put(<<"worker_id">>,
    WorkerSpan, JobWithSpan),
97                      WorkerPid ! JobWithSpan2,
98                      worker_buffer(worker_2, WorkerPid, K, Queue, true)
    ;
99                  true ->
100                     NewQueue = Queue ++ [Job],
101                     worker_buffer(worker_2, WorkerPid, K, NewQueue,
    ServerBusy)
102             end;
103
104         #{<<"probe_ctx">> := _} = Job when CurrentLoad >= K ->
105             {WorkerCtx, WrkPid} = dqsd_otel:start_span(<<"worker_2">>)
    ,
106             dqsd_otel:fail_span(WrkPid),
107             Pid = maps:get(<<"probe_id">>, Job, undefined),
108             dqsd_otel:fail_span(Pid),
109             worker_buffer(worker_2, WorkerPid, K, Queue, ServerBusy);
110
111         done ->
112             case Queue of
113                 [] ->
114                     worker_buffer(worker_2, WorkerPid, K, [], false);
115                 [NextJob | RestQueue] ->
116                     {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_2">>),
117                     JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx
    , NextJob),
118                     JobWithSpan2 = maps:put(<<"worker_id">>,
    WorkerSpan, JobWithSpan),
119                     WorkerPid ! JobWithSpan2,
120                     worker_buffer(worker_2, WorkerPid, K, RestQueue,
    true)
```

```
            end
    end.

worker_loop(worker_1, Y, Worker1Buffer, Worker2Buffer) ->
    receive
        CtxBaggage ->
            ProbeCtx = maps:get(<<"probe_ctx">>, CtxBaggage),
            WorkerCtx = maps:get(<<"worker_ctx">>, CtxBaggage),
            WorkerPid = maps:get(<<"worker_id">>, CtxBaggage),
            proc_lib:spawn_link(fun() ->
                        otel_ctx:attach(ProbeCtx),
                        ?set_current_span(WorkerCtx),
                        loop(Y),
                        Worker2Buffer ! CtxBaggage,
                        dqsd_otel:end_span(WorkerCtx, WorkerPid),
                        ?set_current_span(ProbeCtx)
                    end),

            Worker1Buffer ! done,
            worker_loop(worker_1, Y, Worker1Buffer, Worker2Buffer)
    end;

worker_loop(worker_2, Y, Worker1Buffer, Worker2Buffer) ->
    receive
        CtxBaggage ->
            ProbeCtx = maps:get(<<"probe_ctx">>, CtxBaggage),
            WorkerCtx = maps:get(<<"worker_ctx">>, CtxBaggage),
            WorkerPid = maps:get(<<"worker_id">>, CtxBaggage),
            ProbePid = maps:get(<<"probe_id">>, CtxBaggage),
            proc_lib:spawn_link(fun() ->
                        otel_ctx:attach(ProbeCtx),
                        ?set_current_span(WorkerCtx),
                        loop(Y),
                        dqsd_otel:end_span(WorkerCtx, WorkerPid),
                        ?set_current_span(ProbeCtx),
                        dqsd_otel:end_span(ProbeCtx, ProbePid)
                    end),

            Worker2Buffer ! done,
            worker_loop(worker_2, Y, Worker1Buffer, Worker2Buffer)
    end.

loop(0) -> ok;
loop(N) -> loop(N - 1).

stop(_State) ->
    ok.
```

K.1.2 First to finish application

```

-module(ftf).
-export([start/2, start/3, send/2, worker_loop/4, stop/1,
   worker_buffer/5, coordinator/2]).

```

```
-include_lib("opentelemetry_api/include/otel_tracer.hrl").

start(_Type, _Args) ->
    {ok, _} = application:ensure_all_started(opentelemetry),
    {ok, Pid} = example_sup:start_link(),
    {ok, Pid}.

start(X, Y, K) ->
    Worker1Buffer = spawn_opt(fun() -> worker_buffer(worker_1, undefined, K, [], false) end, [{scheduler, 3}]),
    Worker2Buffer = spawn_opt(fun() -> worker_buffer(worker_2, undefined, K, [], false) end, [{scheduler, 4}]),
    Worker1 = spawn_opt(fun() -> worker_loop(worker_1, Y, Worker1Buffer, Worker2Buffer) end, [{scheduler, 1}]),
    Worker2 = spawn_opt(fun() -> worker_loop(worker_2, Y, Worker1Buffer, Worker2Buffer) end, [{scheduler, 1}]),

    Worker1Buffer ! {set_worker, Worker1},
    Worker2Buffer ! {set_worker, Worker2},

    register(worker_1, Worker1Buffer),
    register(worker_2, Worker2Buffer),
    spawn(fun() -> send(X, {Worker1Buffer, Worker2Buffer}) end),

    {ok, self()}.

send(X, {Worker1Buffer, Worker2Buffer}) ->
    Delay = -math:log(rand:uniform()) / X,
    timer:sleep(trunc(Delay * 1000)),

    B = maps:new(),
    {ProbeCtx, ProbePid} = dqsd_otel:start_span(<<"ftf">>),
    B1 = maps:put(<<"probe_id">>, ProbePid, B),
    Baggage = maps:put(<<"probe_ctx">>, ProbeCtx, B1),

    CoordinatorPid = spawn(fun() -> coordinator(ProbeCtx, ProbePid) end),
    FullBaggage = maps:put(<<"coordinator">>, CoordinatorPid, Baggage),

    Worker1Buffer ! FullBaggage,
    Worker2Buffer ! FullBaggage,

    send(X, {Worker1Buffer, Worker2Buffer}).

coordinator(ProbeCtx, ProbePid) ->
    receive
        {done, _FromWorker} ->
            dqsd_otel:end_span(ProbeCtx, ProbePid)
    end.

worker_buffer(worker_1, WorkerPid, K, Queue, ServerBusy) ->
    CurrentLoad = length(Queue) + if ServerBusy -> 1; true -> 0 end,
    receive
        {set_worker, NewPid} ->
            worker_buffer(worker_1, NewPid, K, Queue, ServerBusy);
```

```
55
56          #{<<"probe_ctx">> := _} = Job when CurrentLoad < K ->
57              case ServerBusy of
58                  false ->
59                      {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_1">>),
60                      JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx
    , Job),
61                      JobWithSpan2 = maps:put(<<"worker_id">>,
    WorkerSpan, JobWithSpan),
62                      WorkerPid ! JobWithSpan2,
63                      worker_buffer(worker_1, WorkerPid, K, Queue, true)
    ;
64                  true ->
65                      NewQueue = Queue ++ [Job],
66                      worker_buffer(worker_1, WorkerPid, K, NewQueue,
    ServerBusy)
67              end;
68
69          #{<<"probe_ctx">> := _} = Job when CurrentLoad >= K ->
70              Pid = maps:get(<<"probe_id">>, Job, undefined),
71              {WorkerCtx, WrkPid} = dqsd_otel:start_span(<<"worker_2">>)
    ,
72              dqsd_otel:fail_span(Pid),
73              dqsd_otel:fail_span(WrkPid),
74              worker_buffer(worker_1, WorkerPid, K, Queue, ServerBusy);
75
76          done ->
77              case Queue of
78                  [] ->
79                      worker_buffer(worker_1, WorkerPid, K, [], false);
80                  [NextJob | RestQueue] ->
81                      {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_1">>),
82                      JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx
    , NextJob),
83                      JobWithSpan2 = maps:put(<<"worker_id">>,
    WorkerSpan, JobWithSpan),
84                      WorkerPid ! JobWithSpan2,
85                      worker_buffer(worker_1, WorkerPid, K, RestQueue,
    true)
86              end
87      end;
88
89  worker_buffer(worker_2, WorkerPid, K, Queue, ServerBusy) ->
90      CurrentLoad = length(Queue) + if ServerBusy -> 1; true -> 0 end,
91      receive
92          {set_worker, NewPid} ->
93              worker_buffer(worker_2, NewPid, K, Queue, ServerBusy);
94
95          #{<<"probe_ctx">> := _} = Job when CurrentLoad < K ->
96              case ServerBusy of
97                  false ->
98                      {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_2">>),
```

```
99                      JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx
    , Job),
100                     JobWithSpan2 = maps:put(<<"worker_id">>,
    WorkerSpan, JobWithSpan),
101                     WorkerPid ! JobWithSpan2,
102                     worker_buffer(worker_2, WorkerPid, K, Queue, true)
    ;
103                 true ->
104                     NewQueue = Queue ++ [Job],
105                     worker_buffer(worker_2, WorkerPid, K, NewQueue,
    ServerBusy)
106             end;
107
108         #{<<"probe_ctx">> := _} = Job when CurrentLoad >= K ->
109             {WorkerCtx, WrkPid} = dqsd_otel:start_span(<<"worker_2">>)
    ,
110             dqsd_otel:fail_span(WrkPid),
111             Pid = maps:get(<<"probe_id">>, Job, undefined),
112             dqsd_otel:fail_span(Pid),
113             worker_buffer(worker_2, WorkerPid, K, Queue, ServerBusy);
114
115         done ->
116             case Queue of
117                 [] ->
118                     worker_buffer(worker_2, WorkerPid, K, [], false);
119                 [NextJob | RestQueue] ->
120                     {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_2">>),
121                     JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx
    , NextJob),
122                     JobWithSpan2 = maps:put(<<"worker_id">>,
    WorkerSpan, JobWithSpan),
123                     WorkerPid ! JobWithSpan2,
124                     worker_buffer(worker_2, WorkerPid, K, RestQueue,
    true)
125             end
126     end.
127
128 worker_loop(worker_1, Y, Worker1Buffer, Worker2Buffer) ->
129     receive
130         CtxBaggage ->
131             ProbeCtx = maps:get(<<"probe_ctx">>, CtxBaggage),
132             WorkerCtx = maps:get(<<"worker_ctx">>, CtxBaggage),
133             WorkerPid = maps:get(<<"worker_id">>, CtxBaggage),
134             CoordinatorPid = maps:get(<<"coordinator">>, CtxBaggage),
135             proc_lib:spawn_link(fun() ->
136                     otel_ctx:attach(ProbeCtx),
137                     ?set_current_span(WorkerCtx),
138                     loop(Y),
139                     dqsd_otel:end_span(WorkerCtx, WorkerPid),
140                     ?set_current_span(ProbeCtx)
141                 end),
142             CoordinatorPid ! {done, worker_1},
143             Worker1Buffer ! done,
144
145             worker_loop(worker_1, Y, Worker1Buffer, Worker2Buffer)
```

```
146     end;
147
148 worker_loop(worker_2, Y, Worker1Buffer, Worker2Buffer) ->
149     receive
150         CtxBaggage ->
151             ProbeCtx = maps:get(<<"probe_ctx">>, CtxBaggage),
152             WorkerCtx = maps:get(<<"worker_ctx">>, CtxBaggage),
153             WorkerPid = maps:get(<<"worker_id">>, CtxBaggage),
154             ProbePid = maps:get(<<"probe_id">>, CtxBaggage),
155             CoordinatorPid = maps:get(<<"coordinator">>, CtxBaggage),
156
157             proc_lib:spawn_link(fun() ->
158                         otel_ctx:attach(ProbeCtx),
159                         ?set_current_span(WorkerCtx),
160                         loop(Y),
161                         dqsd_otel:end_span(WorkerCtx, WorkerPid),
162                         ?set_current_span(ProbeCtx),
163                         dqsd_otel:end_span(ProbeCtx, ProbePid)
164                     end),
165             CoordinatorPid ! {done, worker_2},
166             Worker2Buffer ! done,
167             worker_loop(worker_2, Y, Worker1Buffer, Worker2Buffer)
168     end.
169
170 loop(0) -> ok;
171 loop(N) -> loop(N - 1).
172
173 stop(_State) ->
174     ok.
```

K.1.3 All to finish application

```
1  -module(atf).
2  -export([start/2, start/3, send/1, worker_loop/4, stop/1,
       worker_buffer/5, coordinator/2, set_lambda/1]).
3
4  -include_lib("opentelemetry_api/include/otel_tracer.hrl").
5
6
7  start(_Type, _Args) ->
8      {ok, _} = application:ensure_all_started(opentelemetry),
9      {ok, Pid} = example_sup:start_link(),
10     {ok, Pid}.
11
12 set_lambda(NewX) when is_number(NewX), NewX > 0 ->
13     ets:insert(send_rate, {lambda, NewX}),
14     {ok, NewX}.
15
16 start(X, Y, K) ->
17     Worker1Buffer = spawn_opt(fun() -> worker_buffer(worker_1,
       undefined, K, [], false) end, [{scheduler, 1}]),
18     Worker2Buffer = spawn_opt(fun() -> worker_buffer(worker_2,
       undefined, K, [], false) end, [{scheduler, 1}]),
19     Worker1 = spawn_opt(fun() -> worker_loop(worker_1, Y,
       Worker1Buffer, Worker2Buffer) end, [{scheduler, 1}]),
```

```
20      Worker2 = spawn_opt(fun() -> worker_loop(worker_2, Y,
        Worker1Buffer, Worker2Buffer) end, [{scheduler, 1}]),
21
22      ets:new(send_rate, [named_table, public, set]),
23      ets:insert(send_rate, {lambda, X}),
24
25      Worker1Buffer ! {set_worker, Worker1},
26      Worker2Buffer ! {set_worker, Worker2},
27
28      register(worker_1, Worker1Buffer),
29      register(worker_2, Worker2Buffer),
30      spawn(fun() -> send({Worker1Buffer, Worker2Buffer}) end),
31
32      {ok, self()}.
33
34
35  send({Worker1Buffer, Worker2Buffer}) ->
36      [{lambda, X}] = ets:lookup(send_rate, lambda),
37      Delay = -math:log(rand:uniform()) / X,
38      timer:sleep(trunc(Delay * 1000)),
39
40      B = maps:new(),
41      {ProbeCtx, ProbePid} = dqsd_otel:start_span(<<"atf">>),
42      B1 = maps:put(<<"probe_id">>, ProbePid, B),
43      Baggage = maps:put(<<"probe_ctx">>, ProbeCtx, B1),
44
45      CoordinatorPid = spawn(fun() -> coordinator(ProbeCtx, ProbePid)
        end),
46      FullBaggage = maps:put(<<"coordinator">>, CoordinatorPid, Baggage)
        ,
47
48      Worker1Buffer ! FullBaggage,
49      Worker2Buffer ! FullBaggage,
50
51      send({Worker1Buffer, Worker2Buffer}).
52
53  coordinator(ProbeCtx, ProbePid) ->
54      coordinator_wait(ProbeCtx, ProbePid, []).
55
56  coordinator_wait(ProbeCtx, ProbePid, WorkersDone) ->
57      receive
58          {done, Worker} ->
59              NewDone = lists:usort([Worker | WorkersDone]),
60              case NewDone of
61                  [worker_1, worker_2] ->
62                      dqsd_otel:end_span(ProbeCtx, ProbePid);
63                  _ ->
64                      coordinator_wait(ProbeCtx, ProbePid, NewDone)
65              end
66      end.
67
68
69  worker_buffer(worker_1, WorkerPid, K, Queue, ServerBusy) ->
70      CurrentLoad = length(Queue) + if ServerBusy -> 1; true -> 0 end,
71      receive
72          {set_worker, NewPid} ->
```

```
73              worker_buffer(worker_1, NewPid, K, Queue, ServerBusy);
74
75          #{<<"probe_ctx">> := _} = Job when CurrentLoad < K ->
76              case ServerBusy of
77                  false ->
78                      {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_1">>),
79                      JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx
    , Job),
80                      JobWithSpan2 = maps:put(<<"worker_id">>,
    WorkerSpan, JobWithSpan),
81                      WorkerPid ! JobWithSpan2,
82                      worker_buffer(worker_1, WorkerPid, K, Queue, true)
    ;
83                  true ->
84                      NewQueue = Queue ++ [Job],
85                      worker_buffer(worker_1, WorkerPid, K, NewQueue,
    ServerBusy)
86              end;
87
88          #{<<"probe_ctx">> := _} = Job when CurrentLoad >= K ->
89              Pid = maps:get(<<"probe_id">>, Job, undefined),
90              {WorkerCtx, WrkPid} = dqsd_otel:start_span(<<"worker_2">>)
    ,
91              dqsd_otel:fail_span(Pid),
92              dqsd_otel:fail_span(WrkPid),
93              worker_buffer(worker_1, WorkerPid, K, Queue, ServerBusy);
94
95          done ->
96              case Queue of
97                  [] ->
98                      worker_buffer(worker_1, WorkerPid, K, [], false);
99                  [NextJob | RestQueue] ->
100                     {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_1">>),
101                     JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx
    , NextJob),
102                     JobWithSpan2 = maps:put(<<"worker_id">>,
    WorkerSpan, JobWithSpan),
103                     WorkerPid ! JobWithSpan2,
104                     worker_buffer(worker_1, WorkerPid, K, RestQueue,
    true)
105             end
106     end;
107
108 worker_buffer(worker_2, WorkerPid, K, Queue, ServerBusy) ->
109     CurrentLoad = length(Queue) + if ServerBusy -> 1; true -> 0 end,
110     receive
111         {set_worker, NewPid} ->
112             worker_buffer(worker_2, NewPid, K, Queue, ServerBusy);
113
114         #{<<"probe_ctx">> := _} = Job when CurrentLoad < K ->
115             case ServerBusy of
116                 false ->
117                     {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"
    worker_2">>),
```

```
                    JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx, Job),
                    JobWithSpan2 = maps:put(<<"worker_id">>, WorkerSpan, JobWithSpan),
                    WorkerPid ! JobWithSpan2,
                    worker_buffer(worker_2, WorkerPid, K, Queue, true);
                true ->
                    NewQueue = Queue ++ [Job],
                    worker_buffer(worker_2, WorkerPid, K, NewQueue, ServerBusy)
            end;

        #{<<"probe_ctx">> := _} = Job when CurrentLoad >= K ->
            {WorkerCtx, WrkPid} = dqsd_otel:start_span(<<"worker_2">>),
            dqsd_otel:fail_span(WrkPid),
            Pid = maps:get(<<"probe_id">>, Job, undefined),
            dqsd_otel:fail_span(Pid),
            worker_buffer(worker_2, WorkerPid, K, Queue, ServerBusy);

        done ->
            case Queue of
                [] ->
                    worker_buffer(worker_2, WorkerPid, K, [], false);
                [NextJob | RestQueue] ->
                    {WorkerCtx, WorkerSpan} = dqsd_otel:start_span(<<"worker_2">>),
                    JobWithSpan = maps:put(<<"worker_ctx">>, WorkerCtx, NextJob),
                    JobWithSpan2 = maps:put(<<"worker_id">>, WorkerSpan, JobWithSpan),
                    WorkerPid ! JobWithSpan2,
                    worker_buffer(worker_2, WorkerPid, K, RestQueue, true)
            end
    end.

worker_loop(worker_1, Y, Worker1Buffer, Worker2Buffer) ->
    receive
        CtxBaggage ->
            ProbeCtx = maps:get(<<"probe_ctx">>, CtxBaggage),
            WorkerCtx = maps:get(<<"worker_ctx">>, CtxBaggage),
            WorkerPid = maps:get(<<"worker_id">>, CtxBaggage),
            CoordinatorPid = maps:get(<<"coordinator">>, CtxBaggage),
            proc_lib:spawn_link(fun() ->
                        otel_ctx:attach(ProbeCtx),
                        ?set_current_span(WorkerCtx),
                        loop(Y),
                        dqsd_otel:end_span(WorkerCtx, WorkerPid),
                        ?set_current_span(ProbeCtx)
                    end),
            CoordinatorPid ! {done, worker_1},
            Worker1Buffer ! done,

            worker_loop(worker_1, Y, Worker1Buffer, Worker2Buffer)
```

```
165     end;
166
167 worker_loop(worker_2, Y, Worker1Buffer, Worker2Buffer) ->
168     receive
169         CtxBaggage ->
170             ProbeCtx = maps:get(<<"probe_ctx">>, CtxBaggage),
171             WorkerCtx = maps:get(<<"worker_ctx">>, CtxBaggage),
172             WorkerPid = maps:get(<<"worker_id">>, CtxBaggage),
173             ProbePid = maps:get(<<"probe_id">>, CtxBaggage),
174             CoordinatorPid = maps:get(<<"coordinator">>, CtxBaggage),
175
176             proc_lib:spawn_link(fun() ->
177                         otel_ctx:attach(ProbeCtx),
178                         ?set_current_span(WorkerCtx),
179                         loop(Y),
180                         dqsd_otel:end_span(WorkerCtx, WorkerPid),
181                         ?set_current_span(ProbeCtx),
182                         dqsd_otel:end_span(ProbeCtx, ProbePid)
183                     end),
184             CoordinatorPid ! {done, worker_2},
185             Worker2Buffer ! done,
186             worker_loop(worker_2, Y, Worker1Buffer, Worker2Buffer)
187     end.
188 loop(0) -> ok;
189 loop(N) -> loop(N - 1).
190
191 stop(_State) ->
192     ok.
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**