

UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO

PROJETO DE ENGENHARIA
DESENVOLVIMENTO DO SOFTWARE CONTROLADOR
Medidor de bancada Az® pH/mV/ORP/Cond./TDS/SALT - Versão 2

PEDRO HENRIQUE AL JAWABRA RIBEIRO

MACAÉ - RJ
JULHO - 2024

Sumário

1	Introdução	1
1.1	Escopo do problema	1
1.2	Objetivos	1
1.3	Metodologia utilizada	2
2	Concepção	4
2.1	Medidor Az(R) pH/mV/ORP/Cond./TDS/SALT	4
2.2	Especificação	4
2.3	Requisitos	6
2.3.1	Requisitos funcionais	6
2.3.2	Requisitos não funcionais	6
2.4	Casos de uso	6
2.4.1	Diagrama de caso de uso geral	6
2.4.2	Diagrama de caso de uso específico	6
3	Elaboração	8
3.1	Análise de domínio	8
3.2	Formulação teórica	8
3.3	Identificação de pacotes – assuntos	9
3.4	Diagrama de pacotes – assuntos	10
4	AOO – Análise Orientada a Objeto	11
4.1	Diagramas de classes	11
4.1.1	Dicionário de classes	12
4.2	Diagrama de seqüência – eventos e mensagens	13
4.2.1	Diagrama de sequência geral	13
4.2.2	Diagrama de sequência específico	14
4.3	Diagrama de comunicação – colaboração	15
4.4	Diagrama de estado	16
4.5	Diagrama de atividades	17

5 Projeto	18
5.1 Projeto do sistema	18
5.2 Projeto orientado a objeto – POO	18
5.3 Diagrama de componentes	18
5.4 Diagrama de implantação	19
5.4.1 Lista de características <<features>>	20
5.4.2 Tabela classificação sistema	20
6 Ciclos de Planejamento/Detalhamento	23
6.1 Versão 0.1	23
6.2 Versão 0.2	23
7 Ciclos Construção - Implementação	25
7.1 Código fonte	25
8 Teste	43
8.1 Teste 1: Teste de funcionalidade da interface	43
8.2 Teste 2: Teste de gravação em disco	44
8.3 Teste 3: Teste do Instrumentador	46
9 Documentação para o Desenvolvedor	48
9.1 Dependências para compilar o software	48
9.2 Documentação Doxygen	48
A Apêndice	52

Listas de Figuras

1.1	Etapas para o desenvolvimento do software - <i>projeto de engenharia</i>	3
2.1	Especificações gerais do equipamento	5
2.2	Diagrama de caso de uso – Caso de uso geral	7
2.3	Diagrama de caso de uso específico – Usuário faz medidas e realiza comandos na interface	7
3.1	Diagrama de Pacotes	10
4.1	Diagrama de classes	12
4.2	Diagrama de seqüência	14
4.3	Diagrama de Sequência Específico para obtenção de medidas de pH	15
4.4	Diagrama de comunicação	16
4.5	Diagrama de máquina de estado - Admissão de medida e gravação	16
4.6	Diagrama de atividades - Cálculo da média de valores do Array de medidas	17
5.1	Diagrama de componentes	19
5.2	Diagrama de implantação	20
6.1	Versão 0.1, imagem da <i>interface gráfica</i>	23
6.2	Versão 0.2, imagem do <i>instrumentador</i>	24
8.1	Tela do programa mostrando a interface quando aberta	44
8.2	Tela do Programa mostrando as medidas após feito Teste 1	44
8.3	Tela do programa mostrando como iniciar a gravação	45
8.4	Tela do programa mostrando como parar as gravações e o arquivo sendo gerado	45
8.5	Tela do Exel mostrando os dados gravados a partir do Software	46
8.6	Tela do instrumentador	47
9.1	Html gerado pelo Doxygen com a lista de arquivos	48
9.2	Html gerado pelo Doxygen do arquivo main.cpp	49

Lista de Tabelas

2.1	Caso de uso 1	6
-----	---------------	-------	---

Listagens

7.1	Arquivo de cabeçalho da classe measure.h	25
7.2	Arquivo de implementação da classe measure.cpp	26
7.3	Arquivo de cabeçalho da classe sensor.h	28
7.4	Arquivo de implementação da classe sensor.cpp	29
7.5	Arquivo de cabeçalho da classe utils.h	31
7.6	Arquivo de implementação da classe utils.cpp	32
7.7	Arquivo de cabeçalho da classe instrumentor.h	34
7.8	Arquivo de implementação da classe instrumentor.cpp	38
7.9	Arquivo de implementação da função <code>main()</code>	41

Capítulo 1

Introdução

O estudo da qualidade da água

Há muitas razões importantes para estudar a qualidade da água. A qualidade da água nos ajuda a entender o que está acontecendo na subsuperfície, onde e quanto rápido a água está se movendo, quais reações geoquímicas estão ocorrendo, identificando diferentes fontes de água, etc. Isso se relaciona diretamente com os aspectos dos estudos realizados no laboratório de petrofísica do LENEPE, através de diversos equipamentos capazes de fazer a leitura de medidas das propriedades das rochas e dos fluidos de saturação das mesmas. A água, por ser o fluido de saturação mais comum do subsolo, atribui-se de grande interesse o estudo de suas propriedades.

1.1 Escopo do problema

Medidas realizadas no Medidor de bancada e passíveis de análise:

- pH ou Potencial Hidrogeniônico;
- Condutividade;
- Total de sólidos dissolvidos (TDS);
- Potencial de Oxi-redução (ORP); e
- Salinidade.

1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:
 - Analisar a qualidade da água a ser utilizada em experimentos no laboratório de petrofísica.
 - Desenvolver um projeto de engenharia de software para realizar cálculos estatísticos e organizar as medidas realizadas em arquivos de disco.

- Objetivos específicos:
 - Modelar física e matematicamente o problema.
 - Modelagem estática do software (diagramas de caso de uso, de pacotes, de classes).
 - Modelagem dinâmica do software (desenvolver algoritmos e diagramas exemplificando os fluxos de processamento).
 - Acompanhar medidas em tempo real a partir de gráficos integrados ao software.
 - Simular (realizar simulações para teste do software desenvolvido).
 - Desenvolver instrumentador para acompanhar a performance e desempenho do software

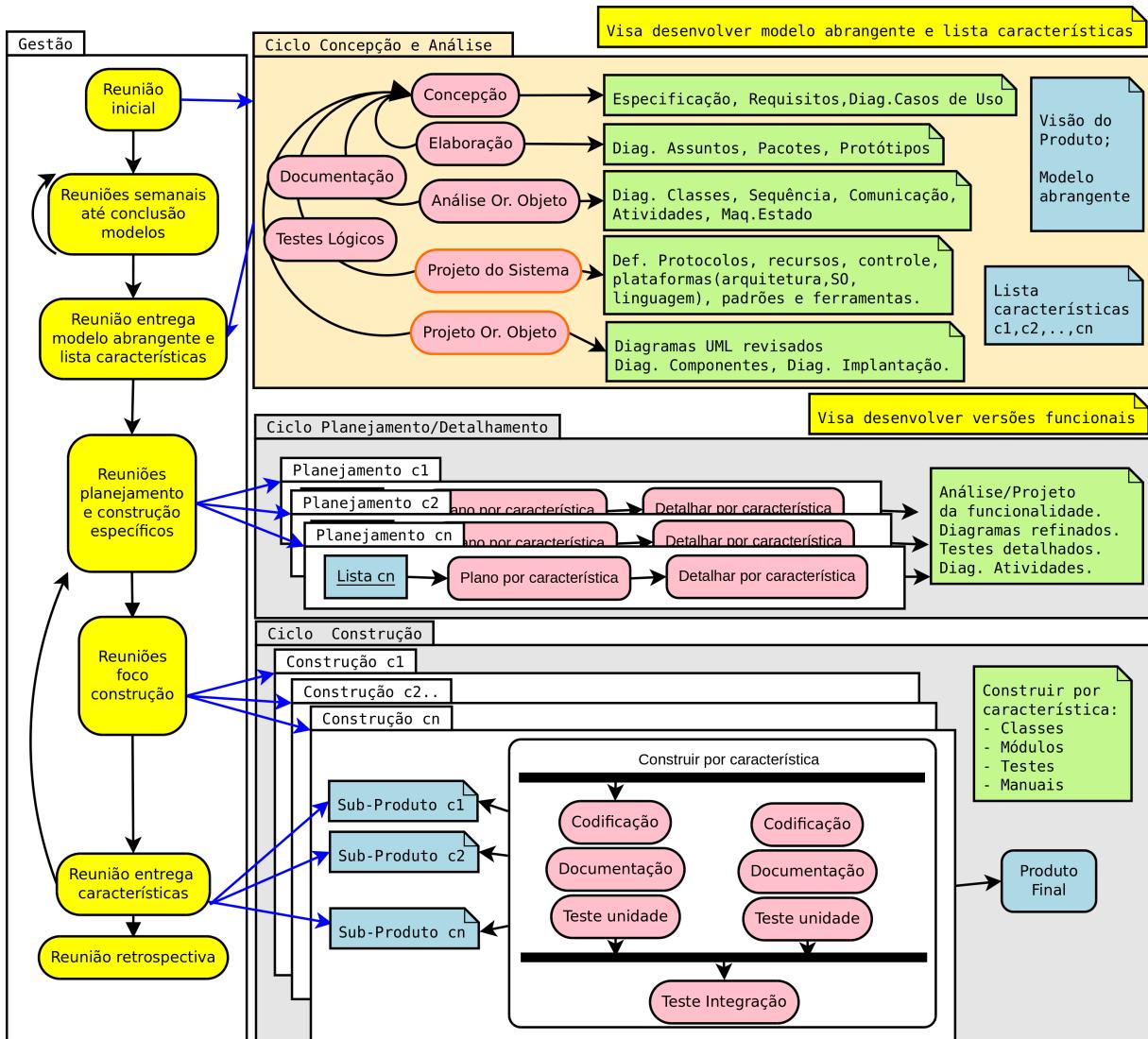
1.3 Metodologia utilizada

O software a ser desenvolvido utiliza a metodologia de engenharia de software apresentada pelo Prof. André Bueno na disciplina de programação e ilustrado na Figura 1.1. Note que o “Ciclo de Concepção e Análise” é composto por diversas partes representadas neste trabalho em diferentes capítulos. Os ciclos de planejamento/detalhamento tem seu próprio capítulo, assim como o ciclo de construção - implementação.

Esta metodologia é utilizada nas disciplinas:

- LEP01447 : Programação Orientada a Objeto em C++.
- LEP01446 : Programação Prática.

Maiores detalhes veja [Bueno, 2003].

Figura 1.1: Etapas para o desenvolvimento do software - *projeto de engenharia*

Capítulo 2

Concepção

2.1 Medidor Az(R) pH/mV/ORP/Cond./TDS/SALT

Nome	Software controlador para o equipamento Medidor de bancada Az(R) pH/mV/ORP/Cond./TDS/SALT
Componentes principais	Aplicação em console com interface gráfica intuitiva
Missão	O software será criado com a intenção de otimizar as análises químicas da qualidade da água, sendo atingido o objetivo através da abstração de processos que o usuário deveria performar manualmente na interface do equipamento. Os dados deverão ser salvos e organizados em um documento do tipo .csv, juntamente com a elaboração dos respectivos parâmetros e cálculos estatísticos para as medidas realizadas. Além de gerado um arquivo instrumentador no formato .json para avaliação de performance do software.

2.2 Especificação

O equipamento alvo deste projeto de engenharia se trata de um medidor de bancada capaz de aferir com certa precisão as propriedades supracitadas. As especificações gerais do equipamento de modelo 86505 estão dispostas na Figura 2.1 abaixo:

Model	86501	86502	86503	86504	86505
pH range	0.00~14.00	N/A		0.00~14.00	
pH accuracy	+/-0.02	N/A		+/-0.02	
pH resolution	0.01	N/A		0.01	
mV range	+/-1999mV	N/A		+/-1999mV	
mV accuracy	+/-0.2mV(-199.9~199.9mV) or +/-2mV(others)	N/A		+/-0.2mV(-199.9~199.9mV) or +/-2mV(others)	
mV resolution	+/-0.1mV(-199.9~199.9mV) or +/-1mV(others)	N/A		+/-0.1mV(-199.9~199.9mV) or +/-1mV(others)	
Cond. range	N/A		0~19.99, 0~199.9, 0~1999uS/cm; 0~19.99, 0~199.9mS/cm		
Cond. accuracy	N/A		+/-1% F.S +/- 1 digit		
Cond. resolution	N/A		0.05% of Full Scale		
TDS range	N/A		N/A		0~19.99, 0~199.9, 0~1999ppm; 0~19.99, 0~199.9ppt
TDS accuracy	N/A		N/A		+/-1% F.S +/- 1 digit
TDS resolution	N/A		N/A		0.05% of Full Scale
Salinity range	N/A		N/A		0~11.38ppt 0~80.0ppt(NaCl)
Salinity accuracy	N/A		N/A		+/-1% F.S +/- 1 digit
Salinity resolution	N/A		N/A		0.1
TDS Factor	N/A		N/A		0.3~1.00
Temp. coefficient	N/A			0~10.0%/ [°] C	
Temp. range			0~80.0 [°] C		
Temp. accuracy			+/-0.5 [°] C		
Temp. resolution			0.1		
Compatible probe	PH	PH ORP	COND.	PH ORP COND.	PH ORP COND.
LCD size(mm)			40(H)x105(W)		
Operating temp.&RH%			0~50 [°] C, Humidity<80%		
Storage temp.&RH%			-20~60 [°] C, Humidity < 90%		
Dimension(mm)			260(L)x168(W)x58(H)		
Weight			150g		
Power			9V adaptor		
Standard Package			Meter/ USA plug type adaptor/manual/probe/ solution/cable/software CD/probe stand		

Figura 2.1: Especificações gerais do equipamento

2.3 Requisitos

Apresenta-se nesta seção os requisitos funcionais e não funcionais.

2.3.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	Interface: gráfica (Qt).
RF-02	Resultados serão armazenados em disco.
RF-03	Poderão ser gerados gráficos nas planilhas geradas.
RF-04	Deve permitir a escolha de modelo do aparelho.
RF-05	O usuário deve ter tal liberdade para escolher o modo de medida realizado pelo medidor.
RF-06	O usuário poderá acompanhar seus resultados em um gráfico integrado à interface gráfica em tempo real.
RF-07	O usuário poderá verificar a performance do software através de um instrumentador.

2.3.2 Requisitos não funcionais

RNF-01	Os cálculos devem ser feitos utilizando-se métodos de estatística da literatura.
RNF-02	O programa não deverá ser multi-plataforma, podendo ser executado apenas em <i>Windows</i> (x32 bits e x64 bits).

2.4 Casos de uso

A Tabela 2.1 mostra a descrição de um caso de uso.

Tabela 2.1: Caso de uso 1	
Nome do caso de uso:	Pesquisador realiza estudo de qualidade da água a ser utilizada no experimento

2.4.1 Diagrama de caso de uso geral

2.4.2 Diagrama de caso de uso específico

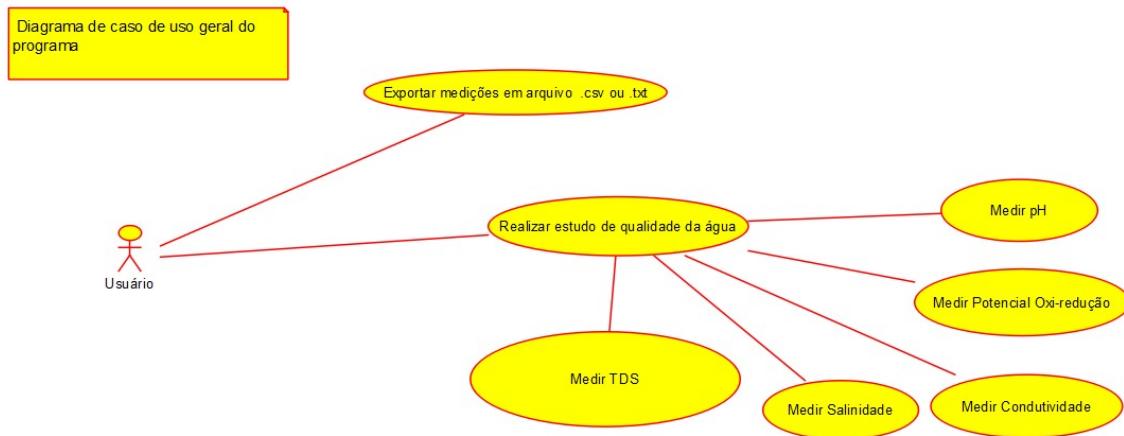


Figura 2.2: Diagrama de caso de uso – Caso de uso geral



Figura 2.3: Diagrama de caso de uso específico – Usuário faz medidas e realiza comandos na interface

Capítulo 3

Elaboração

Depois da definição dos objetivos, da especificação do software e da montagem dos primeiros diagramas de caso de uso, a equipe de desenvolvimento do projeto de engenharia passa por um processo de elaboração que envolve o estudo de conceitos relacionados ao sistema a ser desenvolvido, a análise de domínio e a identificação de pacotes.

Na elaboração fazemos uma análise dos requisitos, ajustando os requisitos iniciais de forma a desenvolver um sistema útil, que atenda às necessidades do usuário e, na medida do possível, permita seu reuso e futura extensão.

Eliminam-se os requisitos "impossíveis" e ajusta-se a idéia do sistema de forma que este seja flexível, considerando-se aspectos como custos e prazos.

3.1 Análise de domínio

Após estudo dos requisitos/especificações do sistema, algumas entrevistas, estudos na biblioteca e disciplinas do curso foi possível identificar nosso domínio de trabalho:

- O Domínio a ser investigado trata-se de propriedades físico-químicas como pH, potencial de oxi-redução, total de sólidos dissolvidos, condutividade e salinidade para entender a composição da água em subsuperfície.
- Neste trabalho reuniremos as áreas de programação orientada a objeto em C++, Probabilidade e estatística, Química e Petrofísica.
- O Software será utilizado no laboratório de petrofísica com intuito de facilitar a interface do pesquisador com o equipamento medidor de bancada Az® pH/mV/ORP/-Cond./TDS/SALT.

3.2 Formulação teórica

Nesta sessão serão descritos as formulações teóricas para entendimento das medidas realizadas pelo software.

Em química, pH é uma escala numérica adimensional utilizada para especificar a acidez ou basicidade de uma solução aquosa. A rigor, o pH é definido como o cologaritmo da atividade de íons hidrônio. Podemos aproximar o cálculo do pH usando a concentração molar o íon hidrônio ao invés da atividade hidrogeniônica.

Potencial eletroquímico, potencial de redução, potencial redox, potencial de oxidação/- redução, potencial de eletrodo ou ORP é a espontaneidade, ou a tendência de uma espécie química adquirir elétrons e, desse modo, ser reduzida. Cada espécie tem seu potencial intrínseco de redução.

Sólidos dissolvidos totais é o conjunto de todas as substâncias orgânicas e inorgânicas contidas num líquido sob formas moleculares, ionizadas ou micro-granulares. É um parâmetro de determinação da qualidade da água, pois avalia o peso total dos constituintes minerais presentes na água, por unidade de volume.

Condutividade elétrica é usada para especificar o caráter elétrico de um material. Ela é simplesmente o recíproco da resistividade, ou seja, inversamente proporcionais e é indicativa da facilidade com a qual um material é capaz de conduzir uma corrente elétrica. A unidade é a recíproca de ohm-metro, isto é, mho^{-1} .

Salinidade é uma medida da quantidade de sais existentes em massas de água naturais, como sejam um oceano, um lago, um estuário ou um aquífero. A forma mais simples de descrever a salinidade é como a relação entre o conteúdo de sais dissolvidos em uma dada quantidade de água.

3.3 Identificação de pacotes – assuntos

- Propriedades da água: propriedades medidas pelo aparelho. São medidas propriedades empíricas das amostras de águas, as quais fazem parte da matéria de físico-química.
- Físico-química:é a disciplina que estuda os fenômenos químicos sob a ótica dos princípios, conceitos e práticas da física, sendo assim a combinação dessas duas ciências: a física e a química. Dessa forma, busca-se explicar os fenômenos químicos através dos conceitos de movimento, energia, força, tempo, temperatura, pressão, volume, calor, e trabalho, química quântica e equilíbrio químico, mecânica estatística e eletroquímica.
- Propriedades Químicas: uma propriedade química é uma propriedade qualificada das substâncias, ou seja, varia de substância para substância, seja ela simples (elemento) ou não (composto). Seria por assim dizer uma propriedade accidental e não essencial.
- Literatura Qualidade da Água: a qualidade da água é um conjunto de características físicas, químicas e biológicas que ela apresenta, de acordo com a sua utilização. Os

padrões de classificação mais usados pretendem classificar a água de acordo com a sua potabilidade, a segurança que apresenta para o ser humano e para o bem estar dos ecossistemas. Usa das propriedades da água para definir parâmetros de qualidade.

- Gráficos: é a tentativa de se expressar visualmente dados ou valores numéricos, de maneiras diferentes, assim facilitando a sua compreensão. Faz parte da matemática como um todo e utiliza elementos da matéria estatística.
- Estatística: é uma ciência que se dedica à coleta, análise e interpretação de dados. Preocupa-se com os métodos de coleta, organização, resumo, apresentação e interpretação dos dados, assim como tirar conclusões sobre as características das fontes donde estes foram retirados, para melhor compreender as situações. Faz parte da matemática como um todo.
- Matemática: o trabalho matemático consiste em procurar e relacionar padrões, de modo a formular conjecturas cuja veracidade ou falsidade é provada por meio de deduções rigorosas a partir de axiomas e definições.

3.4 Diagrama de pacotes – assuntos

Diagrama de assuntos e pacotes para elaboração do software controlador Az(R) pH/mV/ORP/-Cond./TDS/SALT. Veja a Figura 3.1.

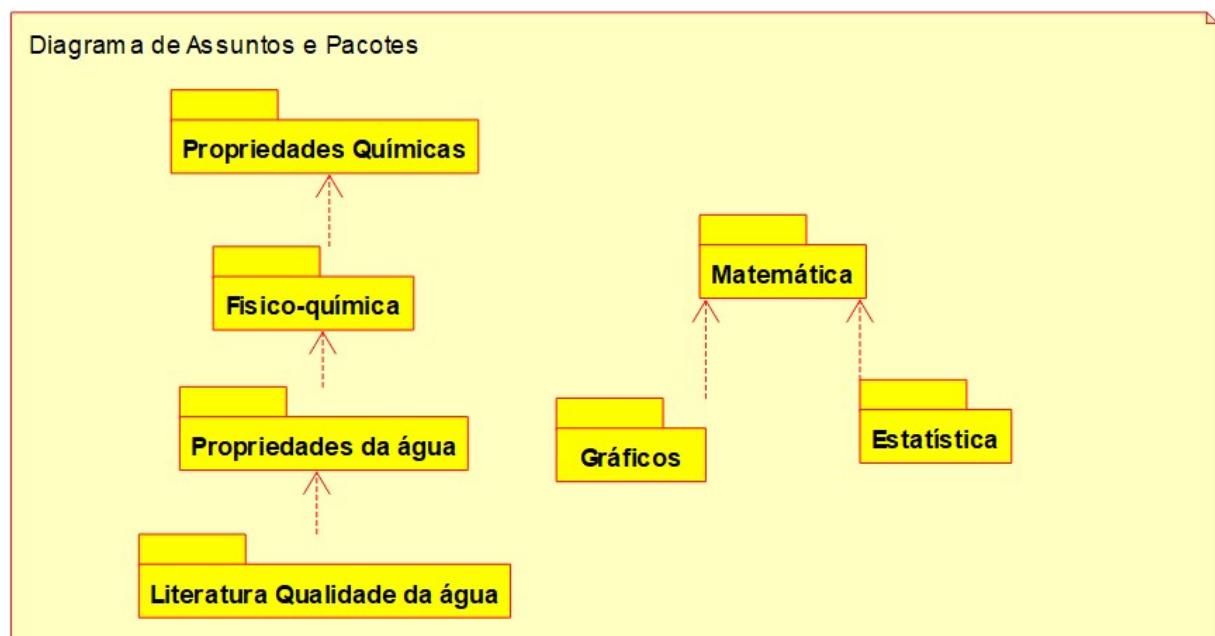


Figura 3.1: Diagrama de Pacotes

Capítulo 4

AOO – Análise Orientada a Objeto

A terceira etapa do desenvolvimento de um projeto de engenharia, no nosso caso um software aplicado a engenharia de petróleo, é a AOO – Análise Orientada a Objeto. A AOO utiliza algumas regras para identificar os objetos de interesse, as relações entre os pacotes, as classes, os atributos, os métodos, as heranças, as associações, as agregações, as composições e as dependências.

O modelo de análise deve ser conciso, simplificado e deve mostrar o que deve ser feito, não se preocupando como isso será realizado.

O resultado da análise é um conjunto de diagramas que identificam os objetos e seus relacionamentos.

4.1 Diagramas de classes

O diagrama de classes é apresentado na Figura 4.1.



Figura 4.1: Diagrama de classes

4.1.1 Dicionário de classes

- Classe CMeasure: Classe que realiza as medidas.
- Classe CSensor: Classe que realiza a escolha do sensor e realaciona as medidas que podem ser realizadas por ele.

- Classe CUtils: Classe que realiza os cálculos estatísticos
- Classe CAbout: Classe que define e escreve na aba “Sobre” na interface
- Classe CMainWindow: Define métodos e seta a funcionalidade de cada botão, caixa de texto, janela,.. da interface gráfica
- Classe CQCustomplot: Constrói o gráfico na interface gráfica.
- Classe EMeasureUnit: representa a classificação das unidades de medida.
- Classe EMeasureMode: representa a classificação do modo de medida.
- Classe ESensortype: representa a classificação dos sensores.
- Classe CInstrumentador: Este criador de perfil irá gerar um arquivo .json que pode ser visualizado com o chrome://tracing para avaliação de performance.

4.2 Diagrama de seqüência – eventos e mensagens

O diagrama de seqüência enfatiza a troca de eventos e mensagens e sua ordem temporal. Contém informações sobre o fluxo de controle do software. Costuma ser montado a partir de um diagrama de caso de uso e estabelece o relacionamento dos atores (usuários e sistemas externos) com alguns objetos do sistema.

4.2.1 Diagrama de sequência geral

Veja o diagrama de seqüência na Figura 4.2.

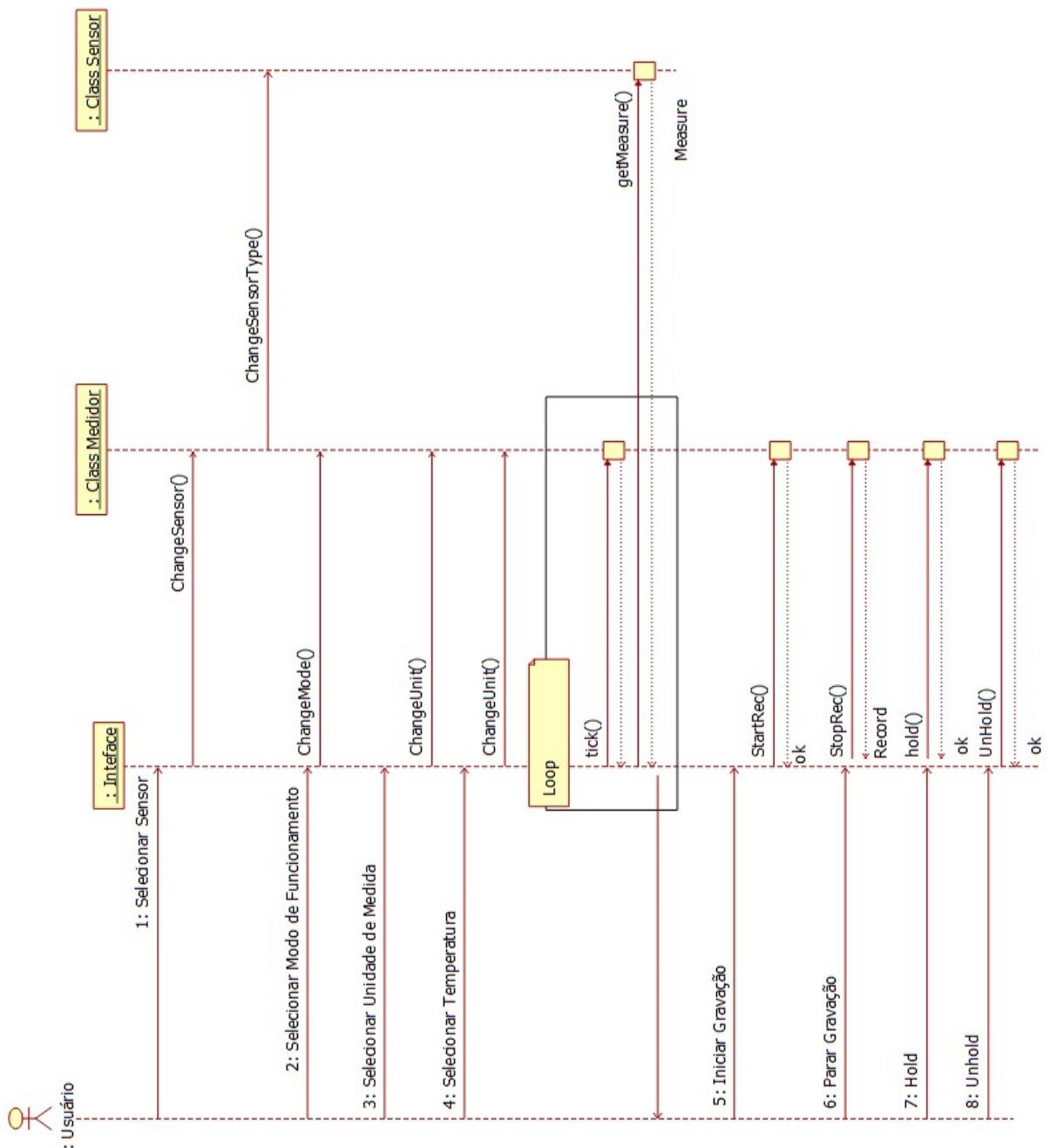


Figura 4.2: Diagrama de seqüência

4.2.2 Diagrama de sequência específico

Veja o diagrama de sequência específico para a aquisição de medidas de pH na Figura 4.3

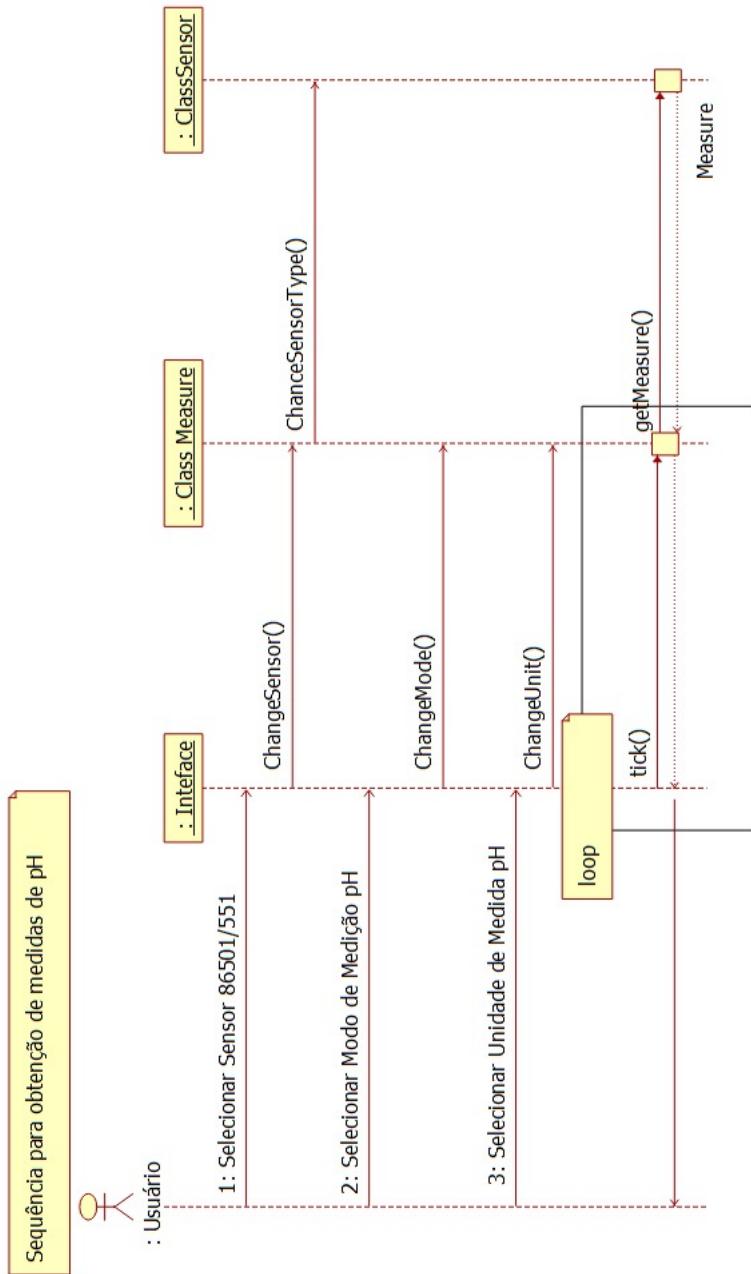


Figura 4.3: Diagrama de Sequência Específico para obtenção de medidas de pH

4.3 Diagrama de comunicação – colaboração

No diagrama de comunicação o foco é a interação e a troca de mensagens e dados entre os objetos.

Veja na Figura 4.3

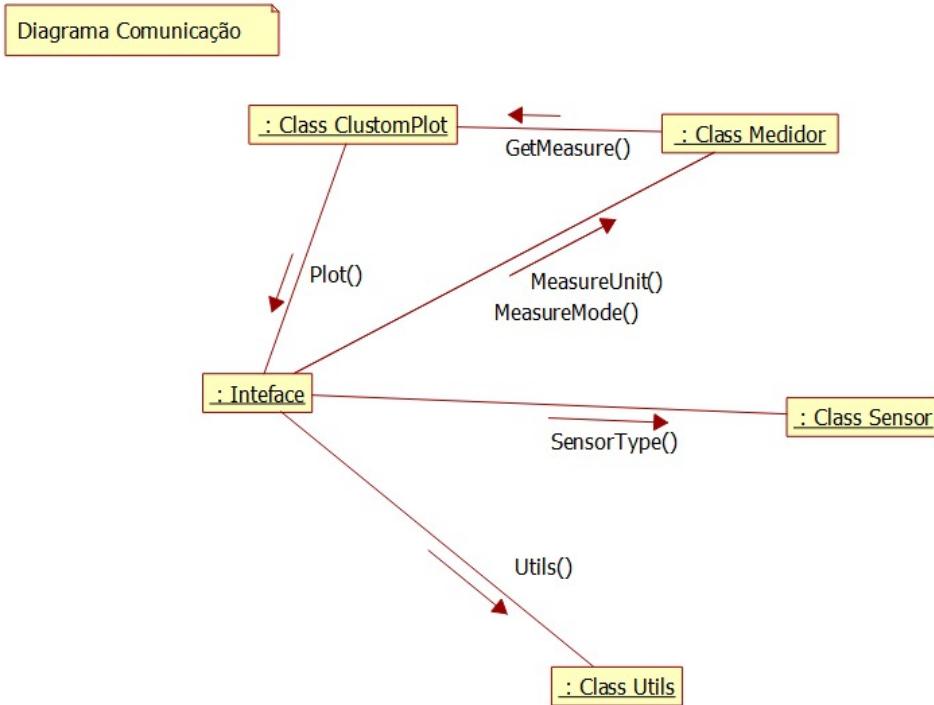


Figura 4.4: Diagrama de comunicação

4.4 Diagrama de estado

Um diagrama de máquina de estado representa os diversos estados que o objeto assume e os eventos que ocorrem ao longo de sua vida ou mesmo ao longo de um processo (histórico do objeto). É usado para modelar aspectos dinâmicos do objeto.

Veja na Figura 4.5 o diagrama de máquina de estado para o objeto “medida”.

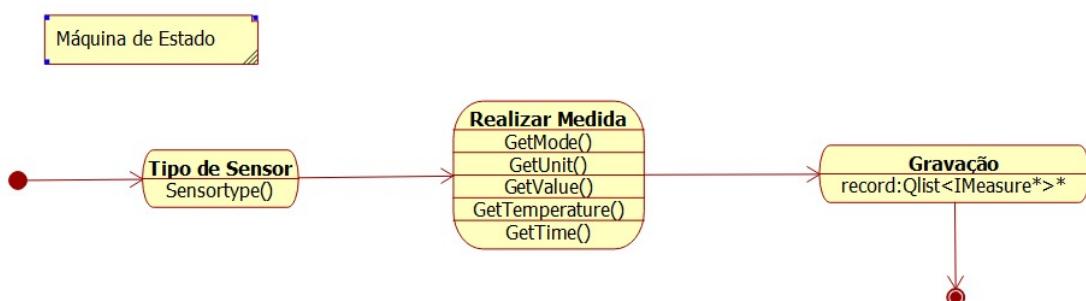


Figura 4.5: Diagrama de máquina de estado - Admissão de medida e gravação

4.5 Diagrama de atividades

Veja na Figura 4.6 o diagrama de atividades correspondente a uma atividade específica do cálculo da média.

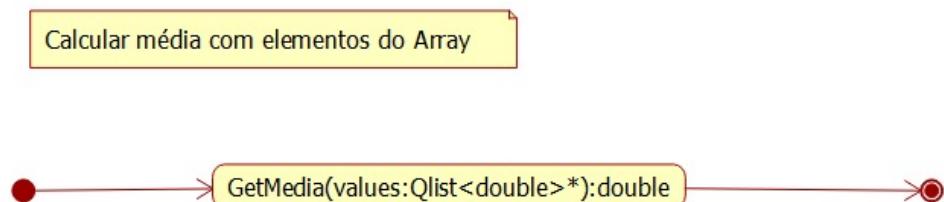


Figura 4.6: Diagrama de atividades - Cálculo da média de valores do Array de medidas

Capítulo 5

Projeto

Neste capítulo do projeto de engenharia veremos questões associadas ao projeto do sistema, incluindo protocolos, recursos, plataformas suportadas, implicações nos diagramas feitos anteriormente, diagramas de componentes e implantação. Na segunda parte revisamos os diagramas levando em conta as decisões do projeto do sistema.

5.1 Projeto do sistema

5.2 Projeto orientado a objeto – POO

O projeto orientado a objeto é a etapa posterior ao projeto do sistema. Baseia-se na análise, mas considera as decisões do projeto do sistema. Acrescenta a análise desenvolvida e as características da plataforma escolhida (hardware, sistema operacional e linguagem de software). Passa pelo maior detalhamento do funcionamento do software, acrescentando atributos e métodos que envolvem a solução de problemas específicos não identificados durante a análise.

Envolve a otimização da estrutura de dados e dos algoritmos, a minimização do tempo de execução, de memória e de custos. Existe um desvio de ênfase para os conceitos da plataforma selecionada.

5.3 Diagrama de componentes

O diagrama de componentes mostra a forma como os componentes do software se relacionam, suas dependências.

Veja na Figura 5.1 o diagrama de componentes do software.

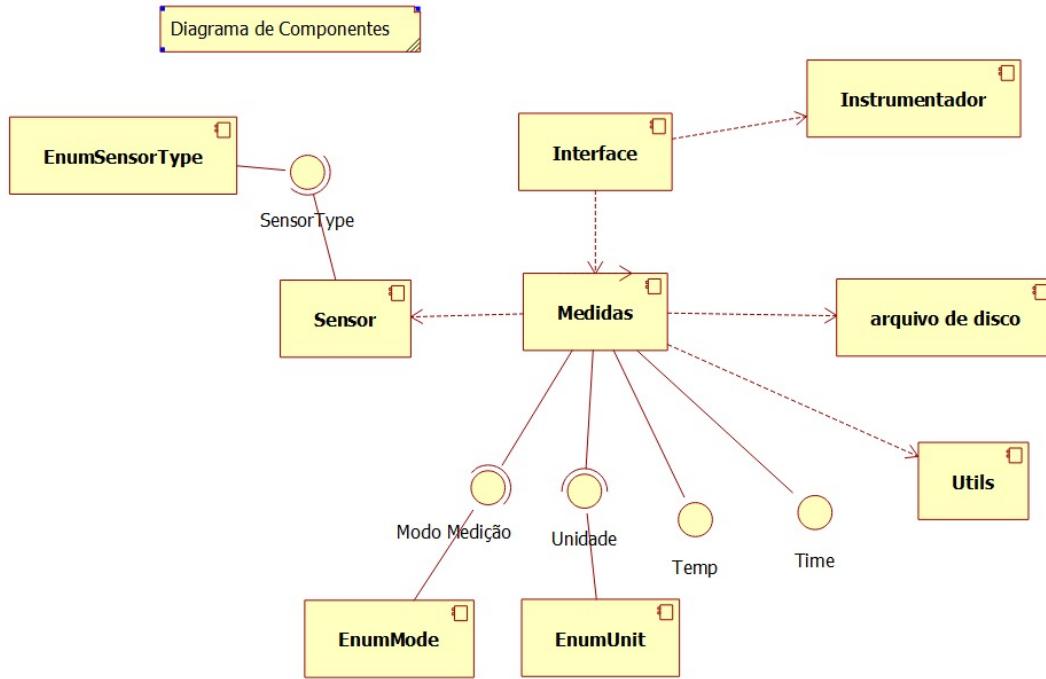


Figura 5.1: Diagrama de componentes

5.4 Diagrama de implantação

O diagrama de implantação é um diagrama de alto nível que inclui relações entre o sistema e o hardware e que se preocupa com os aspectos da arquitetura computacional escolhida. Seu enfoque é o hardware, a configuração dos nós em tempo de execução.

O diagrama de implantação deve incluir os elementos necessários para que o sistema seja colocado em funcionamento: computador, periféricos, processadores, dispositivos, nós, relacionamentos de dependência, associação, componentes, subsistemas, restrições e notas.

Veja na Figura 5.2 um exemplo de diagrama de implantação de um cluster. Observe a presença de um servidor conectado a um switch. Os nós do cluster (ou clientes) também estão conectados ao switch. Os resultados das simulações são armazenados em um servidor de arquivos (*storage*).

Pode-se utilizar uma anotação de localização para identificar onde determinado componente está residente, por exemplo {localização: sala 3}.

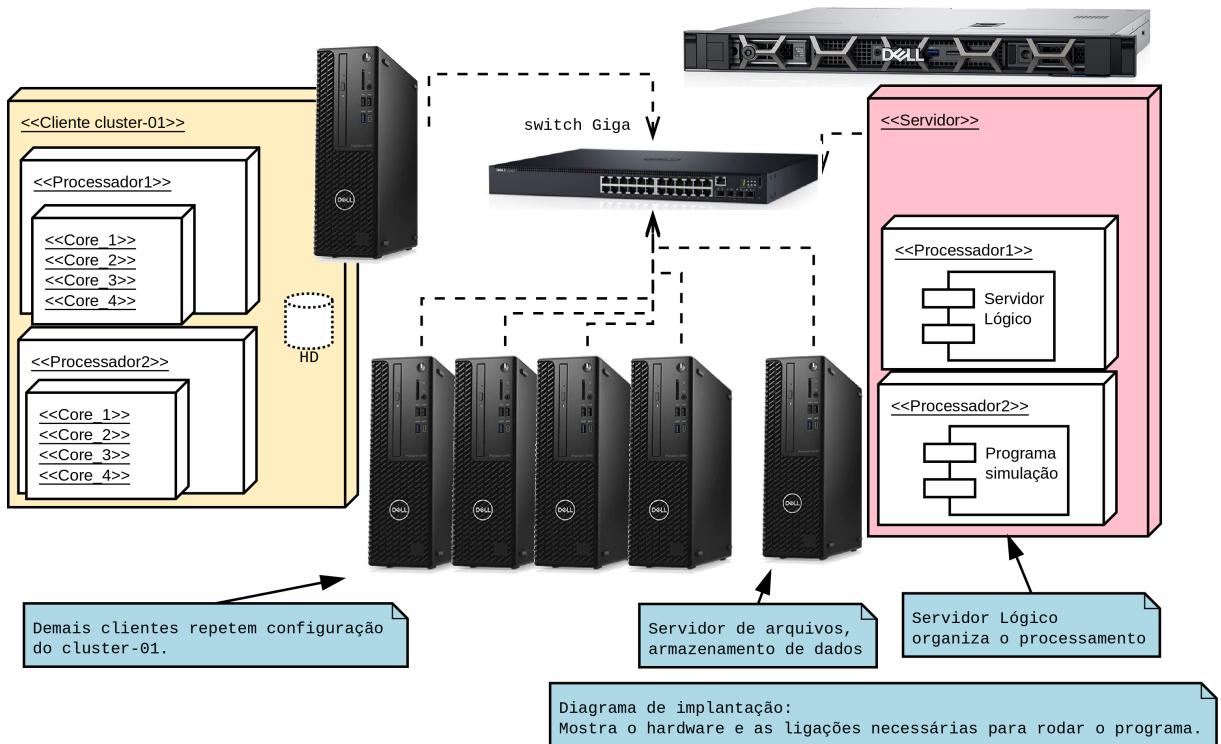


Figura 5.2: Diagrama de implantação

5.4.1 Lista de características <<features>>

No final do ciclo de concepção e análise chegamos a uma lista de características <<features>> que teremos de implementar.

Após a análises desenvolvidas e considerando o requisito de que este material deve ter um formato didático, chegamos a seguinte lista:

- v0.2
 - Aquisição de Medidas
 - Interface gráfica
 - Gráficos em tempo real
 - Gravação de Medidas em arquivo de disco
 - Instrumentador para avaliação de performance
 - Testes

5.4.2 Tabela classificação sistema

A Tabela a seguir é utilizada para classificação do sistema desenvolvido. Deve ser preenchida na etapa de projeto e revisada no final, quando o software for entregue na sua versão final.

Licença:	[X] livre GPL-v3 [] proprietária
Engenharia de software:	[] tradicional [X] ágil [] outras
Paradigma de programação:	[] estruturada [X] orientado a objeto - POO [] funcional
Modelagem UML:	[X] básica [X] intermediária [] avançada
Algoritmos:	[X] alto nível [X] baixo nível
	implementação: [] recursivo ou [X] iterativo; [X] determinístico ou [] não-determinístico; [] exato ou [X] aproximado
	concorrências: [X] serial [X] concorrente [X] paralelo
	paradigma: [X] dividir para conquistar [] programação linear [] transformação/ redução [] busca e enumeração [] heurístico e probabilístico [] baseados em pilhas
Software:	[] de base [X] aplicados [] de cunho geral [X] específicos para determinada área [X] educativo [X] científico
	instruções: [X] alto nível [] baixo nível
	otimização: [X] serial não otimizado [X] serial otimizado [X] concorrente [X] paralelo [] vetorial
	interface do usuário: [] kernel numérico [] linha de comando [] modo texto [] híbrida (texto e saídas gráficas) [X] modo gráfico (ex: Qt) [] navegador
Recursos de C++:	[X] C++ básico (FCC): variáveis padrões da linguagem, estruturas de controle e repetição, estruturas de dados, struct, classes(objetos, atributos, métodos), funções; entrada e saída de dados (<i>streams</i>), funções de cmath
	[X] C++ intermediário: funções lambda. Ponteiros, referências, herança, herança múltipla, polimorfismo, sobrecarga de funções e de operadores, tipos genéricos (templates), <i>smart pointers</i> . Diretrizes de pré-processador, classes de armazenamento e modificadores de acesso. Estruturas de dados: enum, uniões. Bibliotecas: entrada e saída acesso com arquivos de disco, redirecionamento. Bibliotecas: <i>filesystem</i>
	[X] C++ intermediário 2: A biblioteca de gabaritos de C++ (a STL), containers, iteradores, objetos funções e funções genéricas. Noções de processamento paralelo (múltiplas threads, uso de <i>thread, join</i> e <i>mutex</i>). Bibliotecas: <i>random, threads</i>

	[] C++ avançado: Conversão de tipos do usuário, especializações de templates, excessões. Cluster de computadores, processamento paralelo e concorrente, múltiplos processos (pipes, memória compartilhada, sinais). Bibliotecas: <i>expressões regulares, múltiplos processos</i>
Bibliotecas de C++:	[X] Entrada e saída de dados (<i>streams</i>) [X] cmath [X] filesystem [] random [X] threads [X] <i>expressões regulares</i> [] <i>múltiplos processos</i>
Bibliotecas externas:	[] CGnuplot [X] QCustomPlot [X] Qt diálogos [] QT Janelas/menus/BT _____
Ferramentas auxiliares:	Montador: [X] make [] cmake [X] qmake
IDE:	[] Editor simples: kate/gedit/emacs [] kdevelop [X] QT-Creator [] -----
SCV:	[] cvs [] svn [X] git
Disciplinas correlacionadas	[X] estatística [] cálculo numérico [] modelamento numérico [X] análise e processamento de imagens

Capítulo 6

Ciclos de Planejamento/Detalhamento

Apresenta-se neste capítulo os ciclos de planejamento/detalhamento para as diferentes versões desenvolvidas.

6.1 Versão 0.1

Na primeira versão foi construído todo o corpo do software, sendo as classes principais do software e toda a interface gráfica

Veja Figura 6.1.

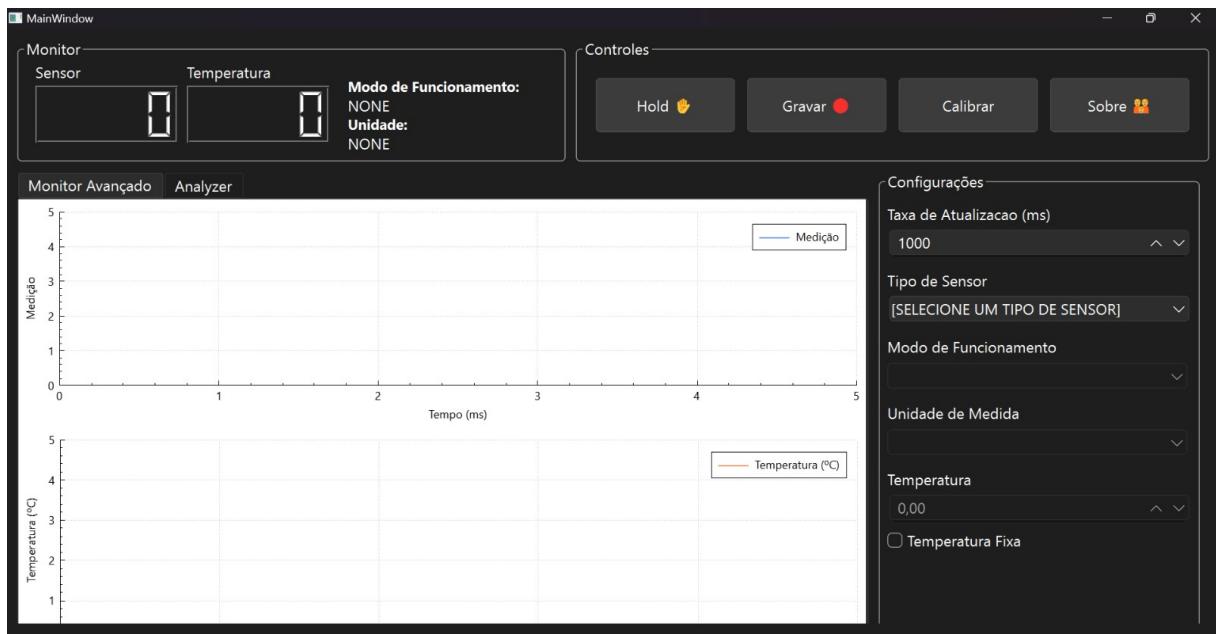


Figura 6.1: Versão 0.1, imagem da interface gráfica

6.2 Versão 0.2

Nesta segunda versão foi adicionado um instrumentador ao software para avaliação de performance, o mesmo gera um arquivo .json para ser lido no chrome://tracy

Veja o instrumentador na figura 6.2

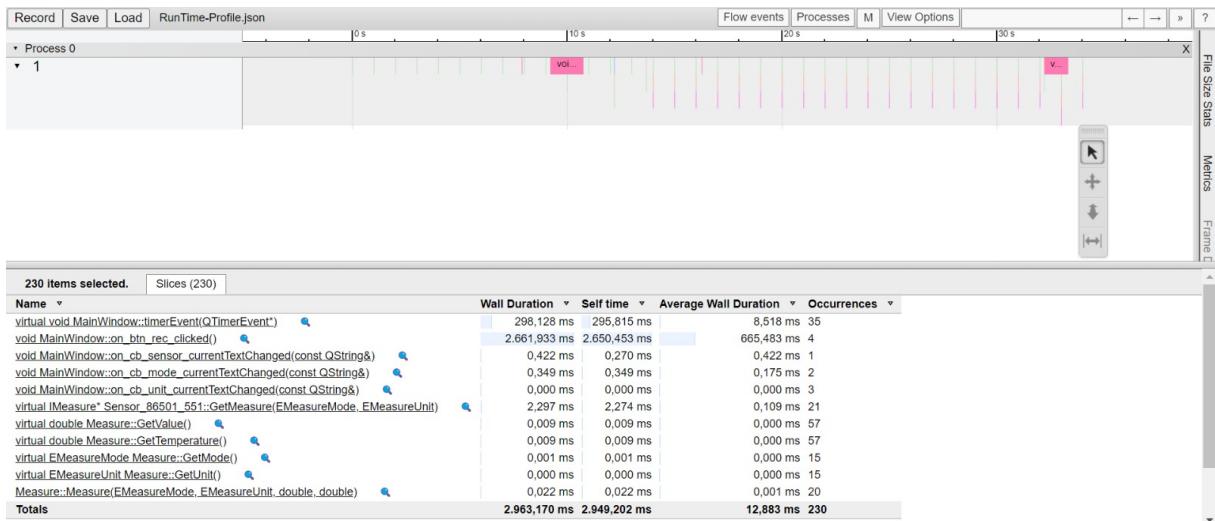


Figura 6.2: Versão 0.2, imagem do *instrumentador*

Capítulo 7

Ciclos Construção - Implementação

Neste capítulo, são apresentados os códigos fonte implementados.

7.1 Código fonte

Apresenta-se a seguir um conjunto de classes (arquivos .h e .cpp) além do programa `main`.

Apresenta-se na listagem 7.1 o arquivo com código da classe `measure.h`.

Listagem 7.1: Arquivo de cabeçalho da classe `measure.h`

```
1 #ifndef MEASURE_H
2 #define MEASURE_H
3
4
5 #include <ctime>
6 #include "enums.h"
7
8 #include "Instrumentor.h"
9
10 class IMeasure {
11 public:
12     /// <summary>
13     /// Get the measure mode.
14     /// </summary>
15     /// <returns></returns>
16     virtual EMeasureMode GetMode() = 0;
17
18     /// <summary>
19     /// Get the measure unit.
20     /// </summary>
21     /// <returns></returns>
```

```

22     virtual EMeasureUnit GetUnit() = 0;
23
24     /// <summary>
25     /// Get the measure value.
26     /// </summary>
27     /// <returns></returns>
28     virtual double GetValue() = 0;
29
30     /// <summary>
31     /// Get the measure value.
32     /// </summary>
33     /// <returns></returns>
34     virtual double GetTemperature() = 0;
35
36     /// <summary>
37     /// Get the measure timestamp.
38     /// </summary>
39     /// <returns></returns>
40     virtual time_t GetTime() = 0;
41 };
42
43 #endif // MEASURE_H

```

Apresenta-se na listagem 7.2 o arquivo de implementação da classe measure.cpp.

Listagem 7.2: Arquivo de implementação da classe measure.cpp

```

1 #include "measure.h"
2 #include "enums.h"
3 #include <ctime>
4
5
6     /// <summary>
7     /// Represents a measurement.
8     /// </summary>
9 class Measure : public IMeasure {
10 public:
11     Measure(EMeasureMode mode, EMeasureUnit unit, double value,
12             double temperature) {
13         STUDY_PROFILE_FUNCTION();
14         m_measureMode = mode;
15         m_measureUnit = unit;
16         m_temperature = temperature;
17         m_value = value;

```

```
17         m_timestamp = time(0);
18     }
19
20     // Inherited via IMeasure
21
22     /// <summary>
23     /// Get the measure mode.
24     /// </summary>
25     /// <returns></returns>
26     virtual EMeasureMode GetMode() override
27     {
28         STUDY_PROFILE_FUNCTION();
29         return m_measureMode;
30     }
31
32     /// <summary>
33     /// Get the measure unit.
34     /// </summary>
35     /// <returns></returns>
36     virtual EMeasureUnit GetUnit() override
37     {
38         STUDY_PROFILE_FUNCTION();
39         return m_measureUnit;
40     };
41
42     /// <summary>
43     /// Get the measure value.
44     /// </summary>
45     /// <returns></returns>
46     virtual double GetValue() override
47     {
48         STUDY_PROFILE_FUNCTION();
49         return m_value;
50     };
51
52     /// <summary>
53     /// Get the measure timestamp.
54     /// </summary>
55     /// <returns></returns>
56     virtual time_t GetTime() override
57     {
58         STUDY_PROFILE_FUNCTION();
```

```
59         return m_timestamp;
60     };
61
62     virtual double GetTemperature() override
63     {
64         STUDY_PROFILE_FUNCTION();
65         return m_temperature;
66     }
67
68
69 private:
70     /// <summary>
71     /// Measure mode.
72     /// </summary>
73     EMeasureMode m_measureMode;
74
75     /// <summary>
76     /// Measure unit.
77     /// </summary>
78     EMeasureUnit m_measureUnit;
79
80     /// <summary>
81     /// Measure value.
82     /// </summary>
83     double m_value;
84
85     /// <summary>
86     /// Measure timestamp.
87     /// </summary>
88     time_t m_timestamp;
89
90     /// <summary>
91     /// Measure temperature.
92     /// </summary>
93     double m_temperature;
94
95 };
```

Apresenta-se na listagem 7.3 o arquivo com código da classe sensor.h.

Listagem 7.3: Arquivo de cabeçalho da classe sensor.h

```
1 #ifndef SENSOR_H
2 #define SENSOR_H
```

```

3
4 #include "measure.h"
5 #include "Instrumentor.h"
6
7 class ISensor
8 {
9 public:
10     virtual IMeasure* GetMeasure(EMeasureMode mode, EMeasureUnit
11                                 unit) = 0;
12     virtual ESensorType GetSensorType() = 0;
13 };
14 #endif // SENSOR_H

```

Apresenta-se na listagem 7.4 o arquivo de implementação da classe sensor.cpp.

Listagem 7.4: Arquivo de implementação da classe sensor.cpp

```

1 #include "measure.cpp"
2 #include "sensor.h"
3
4
5 #include <stdexcept>
6
7 class Sensor_86501_551 : public ISensor {
8 private:
9     ESensorType m_sensor_type = ESensorType_86501_551;
10    double lastValue = 0.01;
11
12    double fRand5(double fMin, double fMax)
13    {
14        double f = (double)rand() / RAND_MAX;
15        return fMin + f * (fMax - fMin);
16    }
17 public:
18    Sensor_86501_551() {}
19    ~Sensor_86501_551() {}
20
21    bool flag = true;
22
23    virtual IMeasure* GetMeasure(EMeasureMode mode, EMeasureUnit
24                                unit) override
25    {
26        STUDY_PROFILE_FUNCTION();

```

```

26     double value;
27
28     if(flag) {
29         value = fRand5(lastValue, lastValue+0.1);
30
31         if(value > 14) {
32             flag = false;
33         }
34     }
35     else {
36         value = fRand5(lastValue-1, lastValue);
37         if(value < 0) {
38             flag = true;
39         }
40     }
41     lastValue = value;
42
43     // TODO: Implementar adaptador do Sensor
44     switch (mode) {
45         case EMeasureMode_pH:
46             return new Measure(EMeasureMode::EMeasureMode_pH,
47                               EMeasureUnit_pH , value, fRand5(24, 25));
48         case EMeasureMode_Unknown:
49             return new Measure(EMeasureMode::EMeasureMode_Unknown ,
50                               EMeasureUnit_pH , 00, fRand5(24, 25));
51             break;
52         case EMeasureMode_Conductivity:
53             return new Measure(EMeasureMode::
54                               EMeasureMode_Conductivity, EMeasureUnit_pH , fRand5
55                               (0, 10), fRand5(24, 25));
56             break;
57         case EMeasureMode_TDS :
58             return new Measure(EMeasureMode::
59                               EMeasureMode_Conductivity, EMeasureUnit_pH , fRand5
60                               (0, 10), fRand5(24, 25));
61             break;
62         case EMeasureMode_ORP :
63             return new Measure(EMeasureMode::
64                               EMeasureMode_Conductivity, EMeasureUnit_pH , fRand5
65                               (0, 10), fRand5(24, 25));
66             break;
67         case EMeasureMode_Salinity:

```

```

60         return new Measure(EMeasureMode::
61             EMeasureMode_Conductivity, EMeasureUnit_pH , fRand5
62             (0, 10), fRand5(24, 25));
63             break;
64         case EMeasureMode_mV:
65             return new Measure(EMeasureMode::EMeasureMode_mV,
66                 EMeasureUnit_pH , fRand5(-100, 100), fRand5(24, 25))
67                 ;
68             break;
69         default:
70             throw std::invalid_argument("Measure mode not
71                 implemented");
72             break;
73     }
74 }
75
76 virtual ESensorType GetSensorType() override
77 {
78     STUDY_PROFILE_FUNCTION();
79     return ESensorType::ESensorType_86501_551;
80 }
81
82 };

```

Apresenta-se na listagem 7.5 o arquivo com código da classe utils.h.

Listagem 7.5: Arquivo de cabeçalho da classe utils.h

```

1 #ifndef UTILS_H
2 #define UTILS_H
3
4
5 #include <list>
6 #include <qlist.h>
7
8 #include "Instrumentor.h"
9
10 class Utils
11 {
12 private:
13     Utils() {}
14 public:
15     static double GetMediana(QList<double>* values);
16     static double GetMax(QList<double>* values);
17     static double GetMin(QList<double>* values);

```

```

18     static double GetDesvioPadrao(QList<double>* values);
19     static double GetMedia(QList<double>* values);
20 };
21
22 #endif // UTILS_H

```

Apresenta-se na listagem 7.6 o arquivo de implementação da classe measure.cpp.

Listagem 7.6: Arquivo de implementação da classe utils.cpp

```

1 #include "utils.h"
2
3
4 #include <QList>
5
6 double Utils::GetMediana(QList<double>* values)
7 {
8     STUDY_PROFILE_FUNCTION();
9     std::sort(values->begin(), values->end());
10
11     double m1, m2;
12
13     switch (values->count() % 2) // Seletor para calculo da mediana
14     {
15         case 0:
16             m1 = values->value(values->count() / 2 - 1);
17             m2 = values->value(values->count() / 2);
18             m1 += m2;
19             return (m1 / 2);
20
21         case 1: // Faixa de valores do vetor e IMPAR.
22             m1 = values->value((values->count() - 1) / 2);
23             return m1;
24     }
25 }
26
27 double Utils::GetMax(QList<double>* values)
28 {
29     STUDY_PROFILE_FUNCTION();
30     double max = std::numeric_limits<double>::min();
31
32     for (int i = 0; i < values->count(); ++i) {
33         double value = values->value(i);

```

```
34
35         if (value > max)
36     {
37         max = value;
38     }
39 }
40
41     return max;
42 }
43
44 double Utils::GetMin(QList<double>* values)
45 {
46     STUDY_PROFILE_FUNCTION();
47     double min = std::numeric_limits<double>::max();
48
49     for (int i = 0; i < values->count(); ++i) {
50         double value = values->value(i);
51
52         if (value < min)
53     {
54             min = value;
55     }
56 }
57
58     return min;
59 }
60
61 double Utils::GetDesvioPadrao(QList<double>* values)
62 {
63     STUDY_PROFILE_FUNCTION();
64     double media = Utils::GetMedia(values);
65
66     float variacoes = 0;
67     for (int i = 0; i < values->count(); i++) {
68         float v = values->value(i) - media;
69         variacoes += v * v;
70     }
71
72     return sqrt(variacoes / values->count());
73 }
74
75 double Utils::GetMedia(QList<double>* values)
```

```

76 {
77     STUDY_PROFILE_FUNCTION();
78     double soma = 0;
79
80     for (int i = 0; i < values->count(); ++i) {
81         double value = values->value(i);
82
83         soma += value;
84     }
85
86     return soma / values->count();
87 }
```

Apresenta-se na listagem 7.7 o arquivo com código da classe utils.h.

Listagem 7.7: Arquivo de cabeçalho da classe instrumentor.h

```

1 #ifndef INSTRUMENTOR_H
2 #define INSTRUMENTOR_H
3
4
5
6 #include <algorithm>
7 #include <chrono>
8 #include <fstream>
9 #include <iomanip>
10 #include <string>
11 #include <thread>
12 #include <mutex>
13 #include <sstream>
14
15 using FloatingPointMicroseconds = std::chrono::duration<double, std
16     ::micro>;
17
18 struct ProfileResult
19 {
20     std::string Name;
21
22     FloatingPointMicroseconds Start;
23     std::chrono::microseconds ElapsedTime;
24     std::thread::id ThreadID;
25 };
26
27 struct InstrumentationSession
```

```
27 {
28     std::string Name;
29 };
30
31 class Instrumentor
32 {
33 public:
34     Instrumentor(const Instrumentor&) = delete;
35     Instrumentor(Instrumentor&&) = delete;
36
37     void BeginSession(const std::string& name, const std::string&
38         filepath = "results.json");
39
40     void EndSession();
41
42     void WriteProfile(const ProfileResult& result);
43
44     static Instrumentor& Get();
45
46 private:
47     Instrumentor();
48
49     ~Instrumentor();
50
51     void WriteHeader();
52
53     void WriteFooter();
54
55     void InternalEndSession();
56
57 private:
58     std::mutex m_Mutex;
59     InstrumentationSession* m_CurrentSession;
60     std::ofstream m_OutputStream;
61 };
62
63 class InstrumentationTimer
64 {
65 public:
66     InstrumentationTimer(const char* name);
67 }
```

```
68     ~InstrumentationTimer();
69
70     void Stop();
71
72 private:
73     const char* m_Name;
74     std::chrono::time_point<std::chrono::steady_clock>
75         m_StartTimepoint;
76     bool m_Stopped;
77 };
78
79 namespace InstrumentorUtils {
80
81     template <size_t N>
82     struct ChangeResult
83     {
84         char Data[N];
85     };
86
87     template <size_t N, size_t K>
88     constexpr auto CleanupOutputString(const char(&expr)
89                                     [N], const char(&remove)[K])
90     {
91         ChangeResult<N> result = {};
92
93         size_t srcIndex = 0;
94         size_t dstIndex = 0;
95         while (srcIndex < N)
96         {
97             size_t matchIndex = 0;
98             while (matchIndex < K - 1 &&
99                   srcIndex + matchIndex < N - 1 &&
100                  expr[srcIndex + matchIndex] ==
101                  remove[matchIndex])
102                 matchIndex++;
103             if (matchIndex == K - 1)
104                 srcIndex += matchIndex;
105             result.Data[dstIndex++] = expr[
106                 srcIndex] == '"' ? '\\' : expr[
107                 srcIndex];
108             srcIndex++;
109         }
110     }
111 }
```

```

103                     return result;
104                 }
105             }
106
107 #define STUDY_PROFILE 1
108 #if STUDY_PROFILE
109
110     #if defined(__GNUC__) || (defined(__MWERKS__) && (
111         __MWERKS__ >= 0x3000)) || (defined(__ICC) && (__ICC >=
112         600)) || defined(__ghs__)
113         #define STUDY_FUNC_SIG __PRETTY_FUNCTION__
114     #elif defined(__DMC__) && (__DMC__ >= 0x810)
115         #define STUDY_FUNC_SIG __PRETTY_FUNCTION__
116     #elif (defined(__FUNCSIG__) || (_MSC_VER))
117         #define STUDY_FUNC_SIG __FUNCSIG__
118     #elif (defined(__INTEL_COMPILER) && (__INTEL_COMPILER >=
119         600)) || (defined(__IBMCPP__) && (__IBMCPP__ >= 500))
120         #define STUDY_FUNC_SIG __FUNCTION__
121     #elif defined(__BORLANDC__) && (__BORLANDC__ >= 0x550)
122         #define STUDY_FUNC_SIG __FUNC__
123     #elif defined(__STDC_VERSION__) && (__STDC_VERSION__ >=
124         199901)
125         #define STUDY_FUNC_SIG __func__
126     #elif defined(__cplusplus) && (__cplusplus >= 201103)
127         #define STUDY_FUNC_SIG __func__
128     #else
129         #define STUDY_FUNC_SIG "STUDY_FUNC_SIG_unknown!"
130     #endif
131
132     #define STUDY_PROFILE_BEGIN_SESSION(name, filepath) :::
133         Instrumentor::Get().BeginSession(name, filepath)
134     #define STUDY_PROFILE_END_SESSION() ::Instrumentor::Get().
135         EndSession()
136     #define STUDY_PROFILE_SCOPE_LINE2(name, line) constexpr auto
137         fixedName##line = ::InstrumentorUtils::CleanupOutputString(
138             name, "__cdecl"); \
139
140         ::InstrumentationTimer
141             timer##line(
142                 fixedName##line,
143                 Data)
144
145     #define STUDY_PROFILE_SCOPE_LINE(name, line)

```

```

    STUDY_PROFILE_SCOPE_LINE2(name, line)
133 #define STUDY_PROFILE_SCOPE(name) STUDY_PROFILE_SCOPE_LINE(
134     name, __LINE__)
135 #else
136     #define STUDY_PROFILE_BEGIN_SESSION(name, filepath)
137     #define STUDY_PROFILE_END_SESSION()
138     #define STUDY_PROFILE_SCOPE(name)
139     #define STUDY_PROFILE_FUNCTION()
140 #endif
141
142#endif

```

Apresenta-se na listagem 7.8 o arquivo de implementação da classe measure.cpp.

Listagem 7.8: Arquivo de implementação da classe instrumentor.cpp

```

1 #include "Instrumentor.h"
2
3 #include <iostream>
4
5
6     void Instrumentor::BeginSession(const std::string &name, const
7         std::string &filepath)
8     {
9         std::lock_guard lock(m_Mutex);
10        if (m_CurrentSession)
11        {
12            std::cout << "Instrumentor::BeginSession(\"" <<
13                name << "\") when session {" <<
14                m_CurrentSession << "} already open." << std::
15                endl;
16        }
17        m_OutputStream.open(filepath);
18
19        if (m_OutputStream.is_open())
20        {
21            m_CurrentSession = new
22                InstrumentationSession({name});
23                WriteHeader();
24        }

```

```
22                     else
23                     {
24
25                         std::cout << "Instrumentor could not open results
26                           file\"{" << filepath << "}\\"." << std::endl;
27                     }
28                 }
29             }
30
31     void Instrumentor::EndSession()
32     {
33         std::lock_guard lock(m_Mutex);
34         InternalEndSession();
35     }
36
37     void Instrumentor::WriteProfile(const ProfileResult &result)
38     {
39         std::stringstream json;
40
41         json << std::setprecision(3) << std::fixed;
42         json << ",{";
43         json << "\"cat\":\"function\",";
44         json << "\"dur\":"
45             << (result.ElapsedTime.count())
46             << ',';
47         json << "\"name\":\""
48             << result.Name << "\",";
49         json << "\"ph\":\"X\",";
50         json << "\"pid\":0,";
51         json << "\"tid\":"
52             << result.ThreadID << ",";
53         json << "\"ts\":"
54             << result.Start.count();
55         json << "}";
56
57         std::lock_guard lock(m_Mutex);
58         if (m_CurrentSession)
59         {
60             m_OutputStream << json.str();
61             m_OutputStream.flush();
62         }
63     }
64
65     Instrumentor &Instrumentor::Get()
66     {
```

```
62             static Instrumentor instance;
63             return instance;
64         }
65
66         Instrumentor::Instrumentor() : m_CurrentSession(nullptr)
67     {
68     }
69
70         Instrumentor::~Instrumentor()
71     {
72             EndSession();
73         }
74
75         void Instrumentor::WriteHeader()
76     {
77             m_OutputStream << "{\"otherData\":[],\"traceEvents"
78             \":[{}]";
79             m_OutputStream.flush();
80         }
81
82         void Instrumentor::WriteFooter()
83     {
84             m_OutputStream << "]}";
85             m_OutputStream.flush();
86         }
87
88         void Instrumentor::InternalEndSession()
89     {
90             if (m_CurrentSession)
91             {
92                 WriteFooter();
93                 m_OutputStream.close();
94                 delete m_CurrentSession;
95                 m_CurrentSession = nullptr;
96             }
97
98         InstrumentationTimer::InstrumentationTimer(const char *name) :
99             m_Name(name), m_Stopped(false)
100        {
101            m_StartTimepoint = std::chrono::steady_clock::now()
102            ;
```

```

101         }
102
103     InstrumentationTimer::~InstrumentationTimer()
104     {
105         if (!m_Stopped)
106             Stop();
107     }
108
109     void InstrumentationTimer::Stop()
110     {
111         auto endTimepoint = std::chrono::steady_clock::now()
112             ();
113         auto highResStart = FloatingPointMicroseconds{
114             m_StartTimepoint.time_since_epoch() };
115         auto elapsedTime = std::chrono::time_point_cast<std
116             ::chrono::microseconds>(endTimepoint).
117             time_since_epoch() - std::chrono::
118             time_point_cast<std::chrono::microseconds>(
119             m_StartTimepoint).time_since_epoch();
120
121         Instrumentor::Get().WriteProfile({ m_Name,
122             highResStart, elapsedTime, std::this_thread::
123             get_id() });
124
125         m_Stopped = true;
126     }

```

Apresenta-se na listagem 7.9 o programa main.cpp.

Listagem 7.9: Arquivo de implementação da função `main()`

```

1 #include "mainwindow.h"
2 #include "Instrumentor.h"
3
4
5 #include <QApplication>
6 #include <QLocale>
7 #include <QTranslator>
8
9
10
11 int main(int argc, char *argv[])
12 {
13

```

```
14     QApplication a(argc, argv);
15
16     QTranslator translator;
17     const QStringList uiLanguages = QLocale::system().uiLanguages()
18         ;
18     for (const QString &locale : uiLanguages) {
19         const QString baseName = "Pedro_" + QLocale(locale).name();
20         if (translator.load(":/i18n/" + baseName)) {
21             a.installTranslator(&translator);
22             break;
23         }
24     }
25     STUDY_PROFILE_BEGIN_SESSION("CreateWindow", "Create-Profile.
26         json");
26     MainWindow w;
27     STUDY_PROFILE_END_SESSION();
28
29     w.show();
30
31     STUDY_PROFILE_BEGIN_SESSION("execWindow", "RunTime-Profile.
32         json");
32     int exe = a.exec();
33     STUDY_PROFILE_END_SESSION();
34
35     return exe;
36 }
```

Capítulo 8

Teste

Todo projeto de engenharia passa por uma etapa de testes. Neste capítulo apresentamos alguns testes do software desenvolvido. Estes testes devem dar resposta aos diagramas de caso de uso inicialmente apresentados (diagramas de caso de uso geral e específicos).

8.1 Teste 1: Teste de funcionalidade da interface

O Teste 1 será realizado para verificar o perfeito funcionamento da interface e dos gráficos em tempo real das medidas. Para realizarmos este teste, primeiramente abriremos o programa no QT, rodando em modo release o executável Pedro.pro que está localizado na pasta versão-2/src, após isso abrirá a tela inicial do programa. Seleccionaremos a aba “Monitor Avançado” que mostra as medidas sendo realizadas em tempo real, em seguida selecionaremos a aba “Tipo de Sensor” o sensor desejado, para nível de teste usaremos o sensor ‘86501/551’ e em seguida apertar em calibrar. Após calibração, continuaremos a configuração do modo de medição, manteremos fixa a taxa de atualização e temperatura (sabendo que podem ser alteradas) e na aba “Modo de Funcionamento” aparecerá as medições possíveis de serem feitas pelo sensor escolhido, neste caso ‘pH’ e ‘mV’, selecionaremos pH para nível de teste, após isso a unidade de medida já será automaticamente selecionada na aba “Unidade de Medida” e o gráfico começará a realizar as medições, porém até então não estão sendo gravadas. Na parte superior do programa podemos selecionar o botão “Hold” para travar as medições e “Resume” para voltar com as medições. Após todos esses procedimentos, verificamos que as medidas estão sendo feitas e acompanhadas nos gráficos de Medição e Temperatura em tempo real e conseguimos validar o perfeito funcionamento da interface.

Veja Figura 8.1 a tela inicial com a descrição dos botões a serem utilizados no teste e na Figura 8.2 como aparecerá o gráfico após realização do teste.



Figura 8.1: Tela do programa mostrando a interface quando aberta

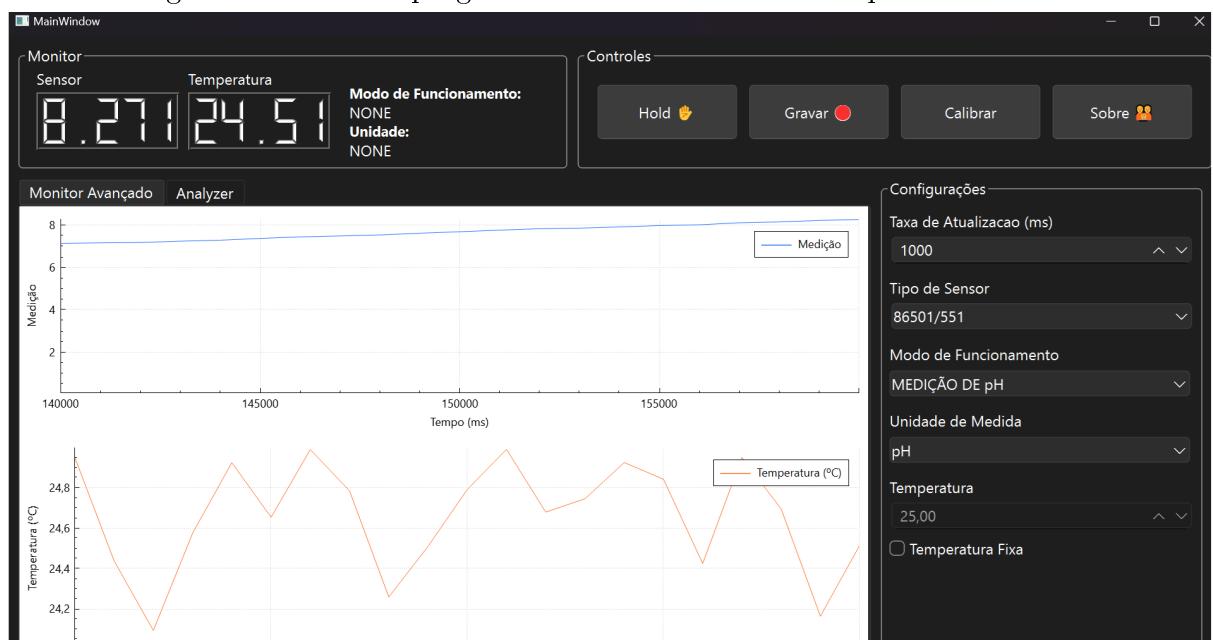


Figura 8.2: Tela do Programa mostrando as medidas após feito Teste 1

8.2 Teste 2: Teste de gravação em disco

No teste 2 testaremos a gravação das medidas em arquivo de disco. Este é um teste subsequente ao teste 1 e portanto todas as tarefas feitas neste teste devem ser feitas após o teste 1. Neste teste, apertaremos o botão “Gravar” e após alguns instantes de gravação, no mesmo local, apertaremos o botão “Parar gravação” e automaticamente será gerado um arquivo .csv com as medidas que estará na pasta versão-

2/src/build/Desktop_Qt_6_7_0_MinGW_64_bit-Release/records. Com este teste podemos verificar o funcionamento do sistema de gravações e atestar seu perfeito funcionamento através dos arquivos gerados.

Veja Figura 8.3 como é iniciada a gravação, Figura 8.4 como param as gravações e Figura 8.5 como são armazenadas as gravações.

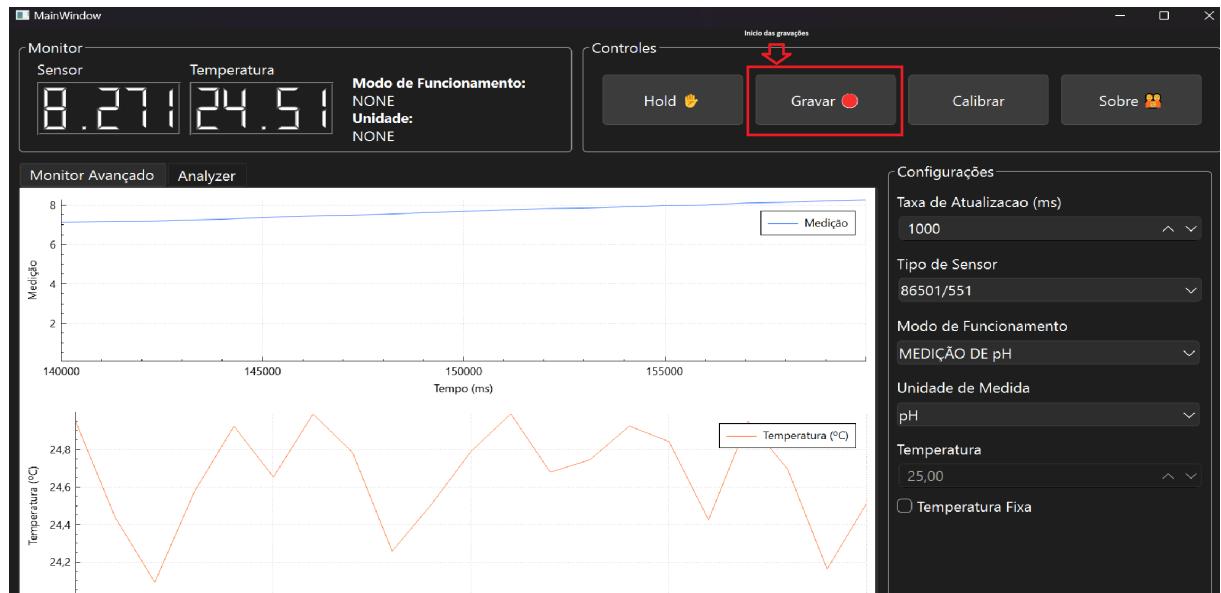


Figura 8.3: Tela do programa mostrando como iniciar a gravação

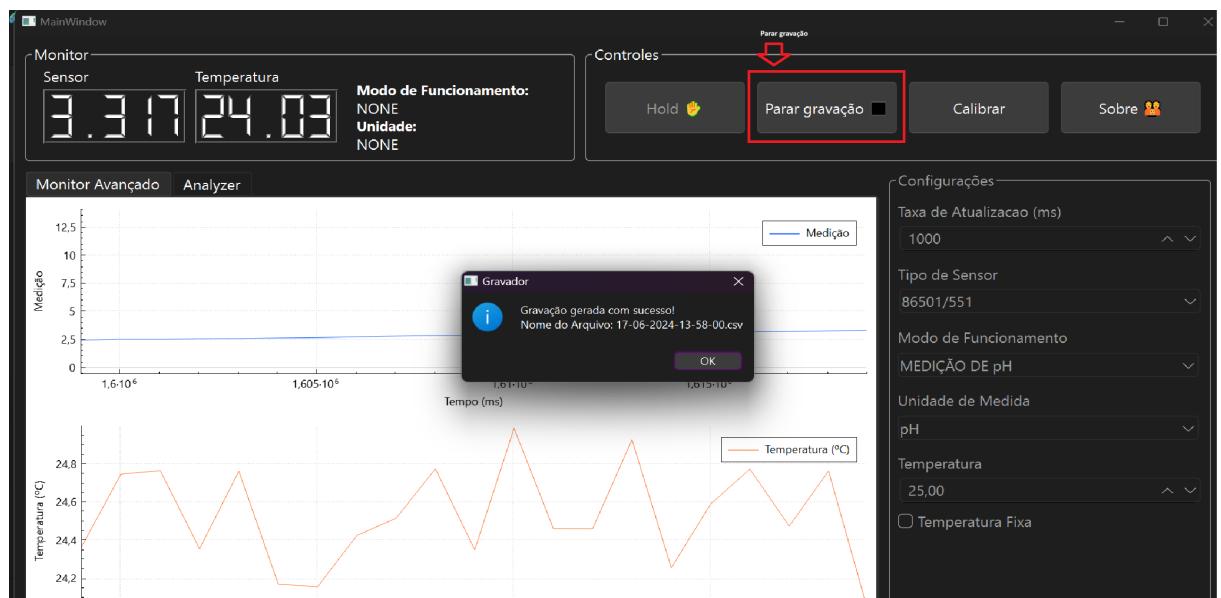


Figura 8.4: Tela do programa mostrando como parar as gravações e o arquivo sendo gerado

	A	B	C	D	E	F
1	Mode	Unit	Value	Temp	Time	Date
2	MEDIÃ±O DE pH	pH	0.42	24.19	1000	#####
3	MEDIÃ±O DE pH	pH	0.48	24.35	2000	#####
4	MEDIÃ±O DE pH	pH	0.49	24.63	3000	#####
5	MEDIÃ±O DE pH	pH	0.49	24.68	4000	#####
6	MEDIÃ±O DE pH	pH	0.49	24.68	5000	#####
7	MEDIÃ±O DE pH	pH	0.58	24.52	6000	#####
8	MEDIÃ±O DE pH	pH	0.64	24.74	7000	#####
9	MEDIÃ±O DE pH	pH	0.71	24.80	8000	#####
10	MEDIÃ±O DE pH	pH	0.73	24.50	9000	#####
11	MEDIÃ±O DE pH	pH	0.78	24.15	10000	#####
12	MEDIÃ±O DE pH	pH	0.82	24.01	11000	#####
13	MEDIÃ±O DE pH	pH	0.85	24.46	12000	#####
14	MEDIÃ±O DE pH	pH	0.90	24.81	13000	#####
15	MEDIÃ±O DE pH	pH	0.96	24.28	14000	#####
16	MEDIÃ±O DE pH	pH	0.97	24.85	15000	#####

Figura 8.5: Tela do Exel mostrando os dados gravados a partir do Software

8.3 Teste 3: Teste do Instrumentador

No Teste 3, testaremos o instrumentador. Este teste é subsequente ao teste 1 e 2, após realizar todas as medidas e tudo o que se quer extrair do programa, fecharemos o mesmo no “X” no canto superior direito e automaticamente será atualizado um arquivo chamado RumTime-Profile.json do instrumentador que está localizado em versão-2/src/build/Desktop_Qt_6_7_0_MinGW_64_bit-Release e este arquivo poderá ser aberto no site chrome://tracy para verificação da performance das funções executadas pelo programa. Cada vez que o programa for aberto e realizado qualquer tipo de função dentro dele, será atualizado esse arquivo. Com a abertura do arquivo no chrome://tracy podemos verificar o tempo de execução das funções do programa e o perfeito funcionamento do instrumentador.

Veja Figura 8.6 o arquivo json aberto no chrome://tracy mostrando o RunTime das funções.

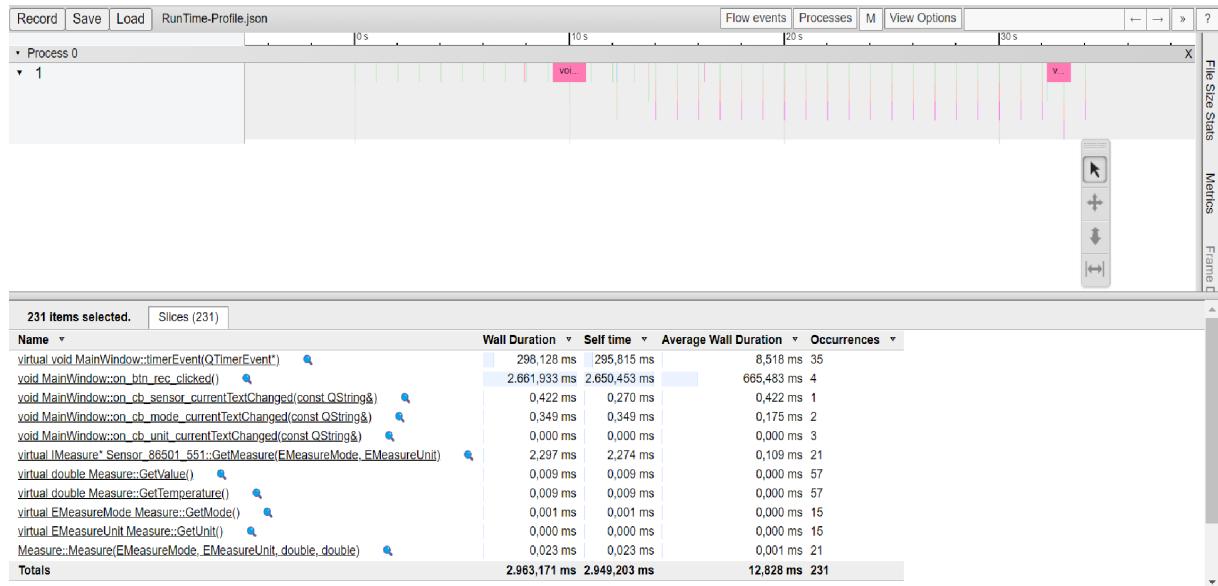


Figura 8.6: Tela do instrumentador

Capítulo 9

Documentação para o Desenvolvedor

9.1 Dependências para compilar o software

Para compilar o software é necessário atender as seguintes dependências:

- Instalar o compilador g++ da GNU disponível em <http://gcc.gnu.org>. Para instalar no GNU/Linux use o comando `yum install gcc`.
- QT Creator e QCustomPlot disponível em <https://www.qt.io/product/development-tools>

9.2 Documentação Doxygen

A Documentação gerada pelo Doxygen está na pasta “Versao-2 /doc /ManualDoDesenvolvedor/Doxygen.”

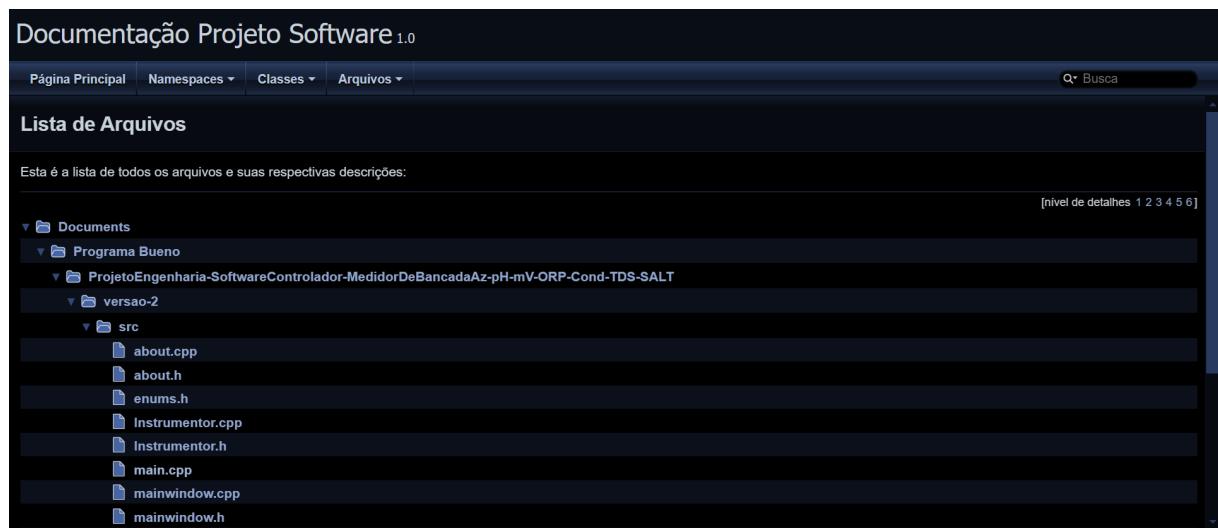


Figura 9.1: Html gerado pelo Doxygen com a lista de arquivos

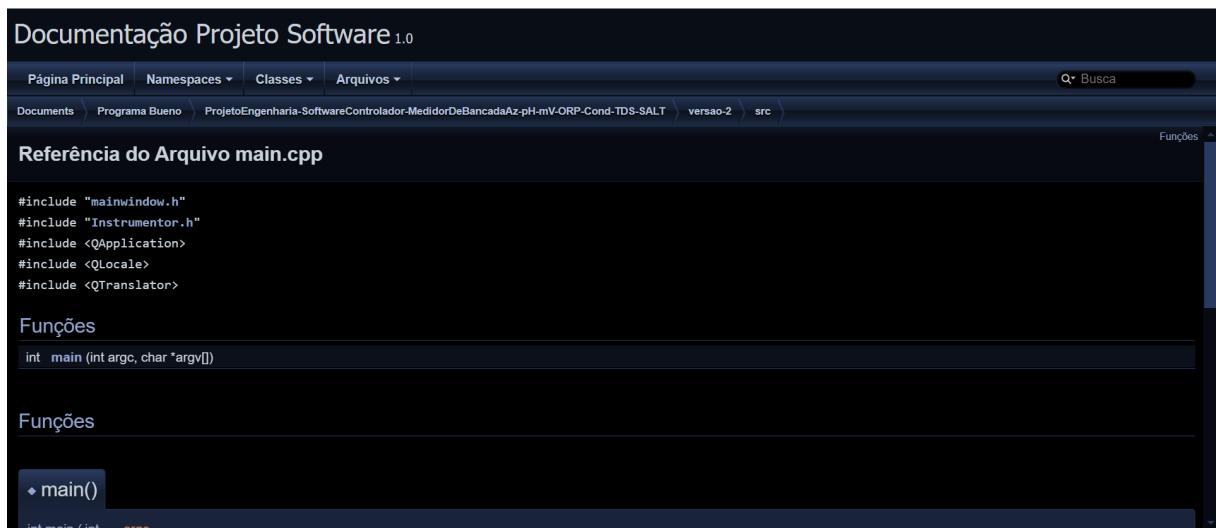


Figura 9.2: Html gerado pelo Doxygen do arquivo main.cpp

Referências Bibliográficas

[Bueno, 2003] Bueno, A. D. (2003). *Programação Orientada a Objeto com C++ - Aprenda a Programar em Ambiente Multiplataforma com Software Livre.* Novatec, São Paulo. 2

Apêndice A

Apêndice

- Neste link <https://gist.github.com/thiago-rezende/979bd63b3b8d3cd4eb1097367cf9b4bb> pode-se verificar o código base que usei para construção do instrumentador para este programa.