

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

PROJETO ENGENHARIA  
MANUAL DO DESENVOLVEDOR DO SOFTWARE  
Análise Incrustação Amostra de Salmoura  
TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA  
PROJETO ENGENHARIA

Versão 1:  
AUTORES Allida Faial e João Vitor Pardo

Versão 2:  
AUTORES  
Prof. André Duarte Bueno

MACAÉ - RJ

Junho - 2025

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Escopo do problema . . . . .	1
1.2	Objetivos . . . . .	2
1.3	Metodologia utilizada . . . . .	2
<b>2</b>	<b>Concepção</b>	<b>4</b>
2.1	Nome do sistema/produto . . . . .	4
2.2	Especificação . . . . .	4
2.3	Requisitos . . . . .	5
2.3.1	Requisitos funcionais . . . . .	5
2.3.2	Requisitos não funcionais . . . . .	5
2.4	Casos de uso . . . . .	5
2.4.1	Diagrama de caso de uso geral . . . . .	6
<b>3</b>	<b>Elaboração</b>	<b>8</b>
3.1	Análise de domínio . . . . .	8
3.2	Formulação teórica . . . . .	8
3.3	Identificação de pacotes – assuntos . . . . .	9
3.4	Diagrama de pacotes – assuntos . . . . .	9
<b>4</b>	<b>AOO – Análise Orientada a Objeto</b>	<b>10</b>
4.1	Diagramas de classes . . . . .	10
4.2	Diagrama de sequência – eventos e mensagens . . . . .	10
4.2.1	Diagrama de sequência geral . . . . .	12
4.2.2	Diagrama de sequência específico . . . . .	14
4.3	Diagrama de comunicação – colaboração . . . . .	14
4.4	Diagrama de estado . . . . .	14
4.5	Diagrama de atividades . . . . .	17
<b>5</b>	<b>Projeto</b>	<b>18</b>
5.1	Projeto do sistema . . . . .	18
5.2	Projeto orientado a objeto – POO . . . . .	19

5.3	Diagrama de componentes . . . . .	20
5.4	Diagrama de implantação . . . . .	21
5.4.1	Lista de características <<features>> . . . . .	22
5.4.2	Tabela classificação sistema . . . . .	23
<b>6</b>	<b>Ciclos de Planejamento/Detalhamento</b>	<b>26</b>
6.1	Versão 0.1 . . . . .	26
<b>7</b>	<b>Ciclos Construção - Implementação</b>	<b>27</b>
7.1	Código fonte . . . . .	27
<b>8</b>	<b>Exemplos de Uso</b>	<b>51</b>
8.1	Estrutura de Arquivos . . . . .	51
8.2	Selecionar o arquivo de íons disponíveis . . . . .	51
8.3	Selecionar o arquivo de sais disponíveis . . . . .	52
8.4	Informar quantas salmouras serão misturadas . . . . .	52
8.5	Inserir dados do cenário . . . . .	52
8.6	Calculo da Precipitação . . . . .	53
8.7	Visualização gráfica (Q vs Ksp) . . . . .	53
8.8	Estrutura de Arquivos . . . . .	53
8.9	Selecionar o arquivo de íons disponíveis . . . . .	56
8.10	Selecionar o arquivo de sais disponíveis . . . . .	56
8.11	Informar quantas salmouras serão misturadas . . . . .	57
8.12	Inserir dados do cenário . . . . .	57
8.13	Calculo da Precipitação . . . . .	57
8.14	Visualização gráfica (Q vs Ksp) . . . . .	59
<b>9</b>	<b>Documentação para o Desenvolvedor</b>	<b>62</b>
9.1	Dependências para compilar o software . . . . .	62
9.2	Como gerar a documentação usando doxygen . . . . .	62

# Lista de Figuras

1.1	Etapas para o desenvolvimento do software - <i>projeto de engenharia</i> . . . . .	3
2.1	Diagrama de caso de uso – Caso de uso geral . . . . .	6
2.2	Diagrama de caso de uso específico – Caso de Uso Simulação de Salmoura .	7
3.1	Diagrama de Pacotes . . . . .	9
4.1	Diagrama de classes . . . . .	11
4.2	Diagrama de seqüência . . . . .	13
4.4	Diagrama de comunicação . . . . .	14
4.3	Diagrama de Sequência Específico . . . . .	15
4.5	Diagrama de máquina de estado . . . . .	16
4.6	Diagrama de atividades . . . . .	17
5.1	Diagrama de componentes . . . . .	21
5.2	Diagrama de implantação . . . . .	22
8.1	Início do Software . . . . .	51
8.2	Documento Íon. . . . .	52
8.3	Documento Salmoura. . . . .	52
8.4	Quantidade de salmoura adicionada. . . . .	54
8.5	Condições termodinâmica . . . . .	54
8.6	Resultado Precipitação . . . . .	54
8.7	Resultado Precipitação Gráfico . . . . .	55
8.8	Condições que o sal se precipita . . . . .	55
8.10	Início do Software . . . . .	56
8.11	Documento Íon. . . . .	56
8.12	Documento Salmoura. . . . .	57
8.9	Sal específico precipitado . . . . .	58
8.13	Quantidade de salmoura adicionada. . . . .	58
8.14	Condições termodinâmica . . . . .	58
8.15	Resultado Precipitação . . . . .	60
8.16	Resultado Precipitação Gráfico . . . . .	60
8.17	Condições que o sal se precipita . . . . .	61

8.18 Sal específico precipitado . . . . .	61
---	----

# Lista de Tabelas

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

# Listagens

7.1	Arquivo de cabeçalho da classe CIons . . . . .	27
7.2	Arquivo de cabeçalho da classe CSalt . . . . .	29
7.3	Arquivo de cabeçalho da classe CSalmoura . . . . .	31
7.4	Arquivo de cabeçalho da classe CMisturaSalmouras . . . . .	33
7.5	Arquivo de cabeçalho da classe CSimuladorPrecipitacao . . . . .	34
7.6	Arquivo de cabeçalho da classe CTabelaPropriedadesIons . . . . .	35
7.7	Arquivo de cabeçalho da classe CPlotPrecipitacao . . . . .	36
7.8	Arquivo de implementação da classe CIons . . . . .	37
7.9	Arquivo de implementação da classe CSalt . . . . .	38
7.10	Arquivo de implementação da classe CSalmoura . . . . .	40
7.11	Arquivo de implementação da classe CMisturaSalmouras . . . . .	40
7.12	Arquivo de implementação da classe CSimuladorPrecipitacao . . . . .	41
7.13	Arquivo de implementação da função <code>main()</code> . . . . .	46
7.14	Arquivo de implementação da classe CPlotPrecipitacao . . . . .	47
7.15	Arquivo de implementação da classe CTabelaPropriedadesIons . . . . .	49

# Capítulo 1

## Introdução

No presente projeto de engenharia desenvolve-se o software *Precipitation Simulator*, uma ferramenta aplicada à engenharia de petróleo, utilizando o paradigma da orientação a objetos. O sistema permite prever a precipitação de sais com base em dados de concentração iônica e condições termodinâmicas, fornecendo suporte didático e técnico para estudos laboratoriais e análises de compatibilidade iônica em poços de petróleo.

### 1.1 Escopo do problema

Segundo o CREA/CONFEA, um dos quesitos fundamentais que diferenciam a atuação de um tecnólogo da atuação de um engenheiro é a capacidade de desenvolver um projeto completo de engenharia. Neste trabalho, desenvolve-se um projeto de engenharia de software voltado à solução de um problema específico da engenharia de petróleo: a precipitação de sais em ambientes de produção offshore, que pode causar incrustações em tubulações, entupimentos e falhas operacionais.

O software desenvolvido visa prever a formação de sais insolúveis a partir da composição iônica da salmoura produzida e das condições de temperatura e pressão do sistema. A modelagem é feita de forma simplificada, mas extensível, utilizando os fundamentos da termodinâmica química e da modelagem orientada a objetos.

Este sistema pode ser aplicado em fases iniciais de projeto de completação, simulações laboratoriais e em atividades acadêmicas, como ferramenta didática. Situa-se no contexto da indústria de óleo e gás brasileira, com foco em ambientes offshore, e se diferencia de outros softwares comerciais por sua leveza, acessibilidade e foco em fundamentos educacionais. Embora limitado inicialmente a combinações binárias de íons, o sistema permite expansão para cenários mais complexos, com suporte futuro a arquivos de entrada, gráficos e ajuste do  $K_{sp}$  com a temperatura.



## 1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:
  - Desenvolver um software de engenharia, utilizando modelagem orientada a objetos em C++, capaz de simular a precipitação de sais a partir de dados de concentração iônica, temperatura e pressão, com aplicação em contextos de produção de petróleo e com foco em uso acadêmico e didático.
- Objetivos específicos:
  - Modelar fisicamente e matematicamente o processo de precipitação de sais em solução aquosa, utilizando o conceito de produto iônico  $Q$  e produto de solubilidade  $K_{sp}$
  - Desenvolver a modelagem estática do software, incluindo os diagramas de classes, pacotes e casos de uso.
  - Desenvolver a modelagem dinâmica do software, com implementação dos fluxos de dados, algoritmos e interações entre classes.
  - Implementar classes responsáveis por íons, sais e condições termodinâmicas, incluindo `ion`, `Salt`, `CreateIons`, `thermodynamicConditions` e `PrecipitationCalculator`.
  - Simular cenários com diferentes sais e condições físico-químicas, exibindo os resultados em formato textual (terminal) e, futuramente, em gráficos.
  - Implementar um manual simplificado de uso do software, com exemplos de entrada e interpretação da saída.

## 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.

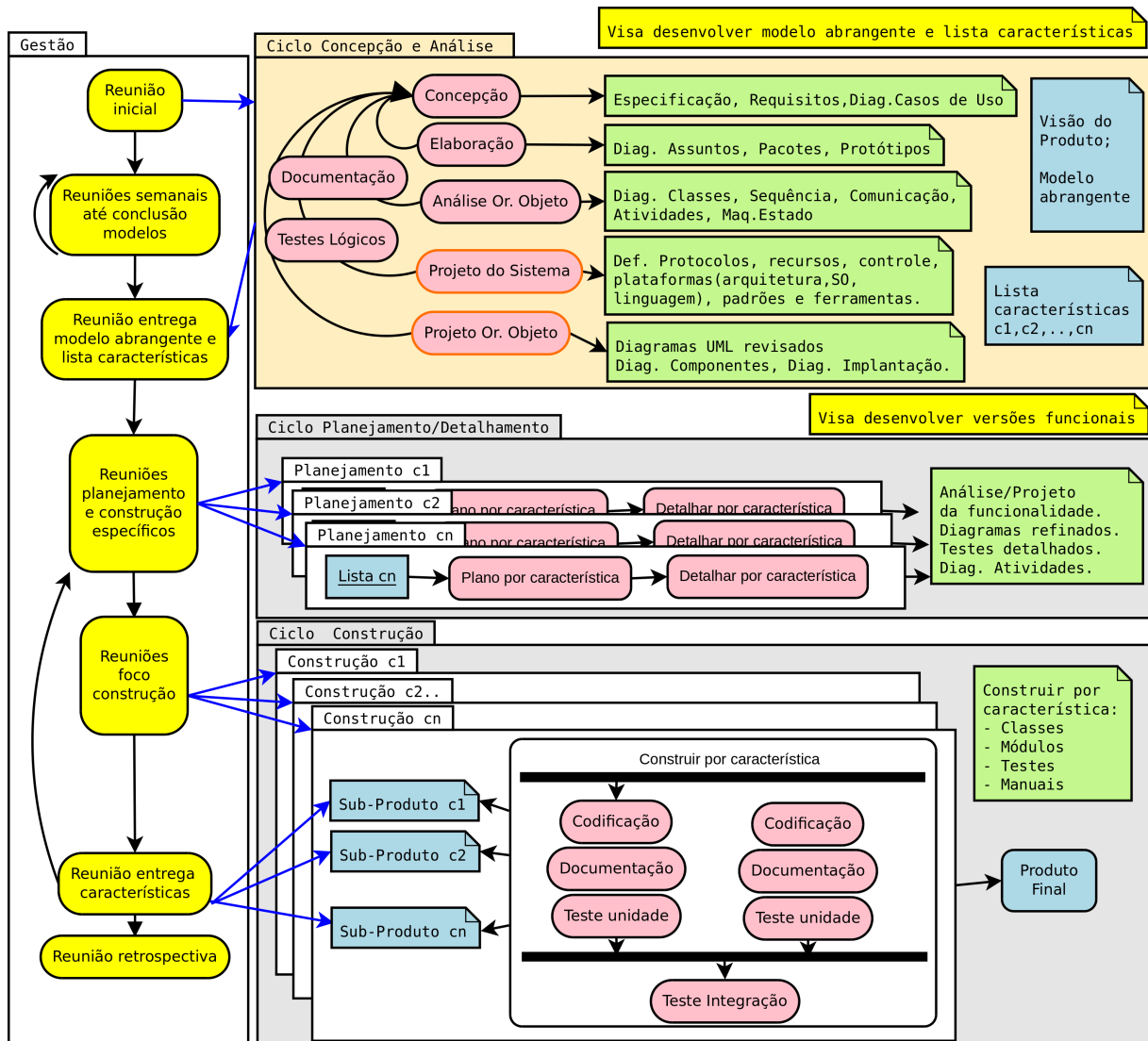


Figura 1.1: Etapas para o desenvolvimento do software - *projeto de engenharia*

# Capítulo 2

## Concepção

Apresenta-se neste capítulo do projeto de engenharia a concepção, a especificação do sistema a ser modelado e desenvolvido.

### 2.1 Nome do sistema/produto

<b>Nome</b>	Sistema de Análise de Incrustação em Amostras de Salmoura
<b>Componentes principais</b>	Interface de entrada para criação de íons, sais, salmouras e condições termodinâmicas; módulo de análise da precipitação; e relatório de resultados.
<b>Missão</b>	Prever a possibilidade de formação de incrustação mineral em ambientes de produção de petróleo a partir da análise de dados de salmouras

### 2.2 Especificação

O software permite ao usuário inserir dados relativos à concentração de íons presentes na salmoura,

informações de temperatura e pressão do meio, e realizar a análise de possibilidade de precipitação com base

na comparação entre o produto iônico e o produto de solubilidade (Kps) de determinados sais. O sistema

calcula automaticamente o produto iônico e emite um diagnóstico sobre a ocorrência ou não de incrustação,

sendo possível aplicar o modelo a sais como calcita ( $\text{CaCO}_3$ ) e barita ( $\text{BaSO}_4$ ).

## 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.

<b>RF-01</b>	O sistema deve permitir ao usuário cadastrar íons com nome, concentração e carga
<b>RF-02</b>	O sistema deve permitir ao usuário criar sais a partir de dois íons e um valor de Kps
<b>RF-03</b>	O sistema deve aceitar a entrada de condições termodinâmicas (temperatura e pressão)
<b>RF-04</b>	O sistema deve calcular o produto iônico com base nas informações inseridas
<b>RF-05</b>	O sistema deve informar se ocorre ou não precipitação para cada sal inserido

### 2.3.2 Requisitos não funcionais

<b>RNF-01</b>	O software deve ser multiplataforma (Windows, Linux, MacOS).
<b>RNF-02</b>	O sistema deve ser desenvolvido em linguagem C++ com estrutura orientada a objeto
<b>RNF-03</b>	O sistema deve ser implementado em linguagem C++ com interface gráfica amigável.

## 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:	Analisar Precipitação de Sal em Salmoura
Resumo/descrição:	O usuário insere dados dos íons presentes na salmoura, as condições de temperatura e pressão, e o sistema calcula se ocorrerá ou não a precipitação de sais com base na comparação entre o produto iônico (Q) e o Kps.
Etapas:	1. Criar íons com nome, carga e concentração. 2. Criar sais com os íons e valor de Kps. 3. Inserir condições termodinâmicas (T e P). 4. Executar a análise de precipitação. 5. Obter diagnóstico (precipita / não precipita).
Cenários alternativos:	O usuário pode inserir um sal que não atinge o Kps, e o sistema deverá corretamente indicar que não há precipitação. O sistema também deve lidar com inserções incompletas ou inconsistentes, emitindo mensagens de erro

2.4.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral (Figura 1.1) representa as interações do usuário com o sistema. O usuário

- pode:
- Criar íons;
  - Criar sais;
  - Criar salmouras;
  - Informar as condições termodinâmicas;
  - Analisar a precipitação

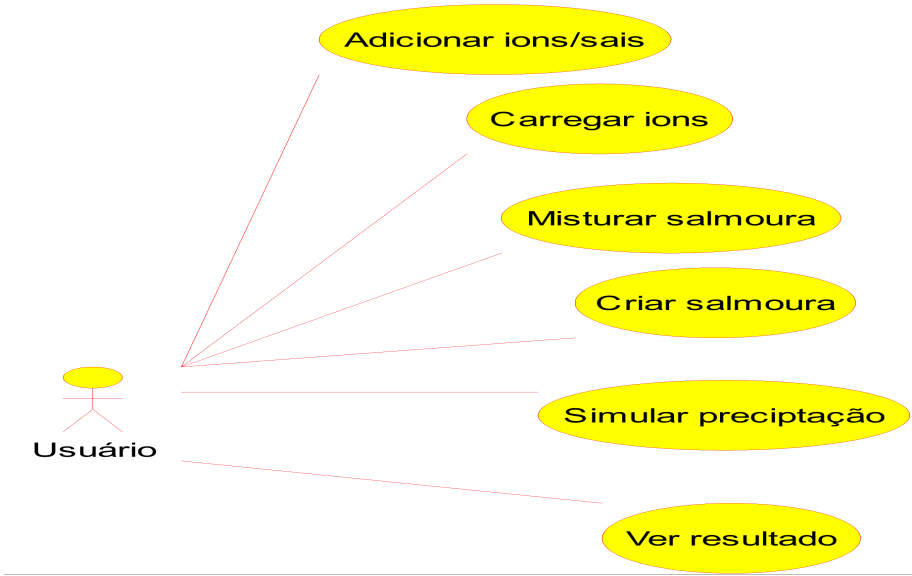


Figura 2.1: Diagrama de caso de uso – Caso de uso geral

Diagrama de caso de uso específico

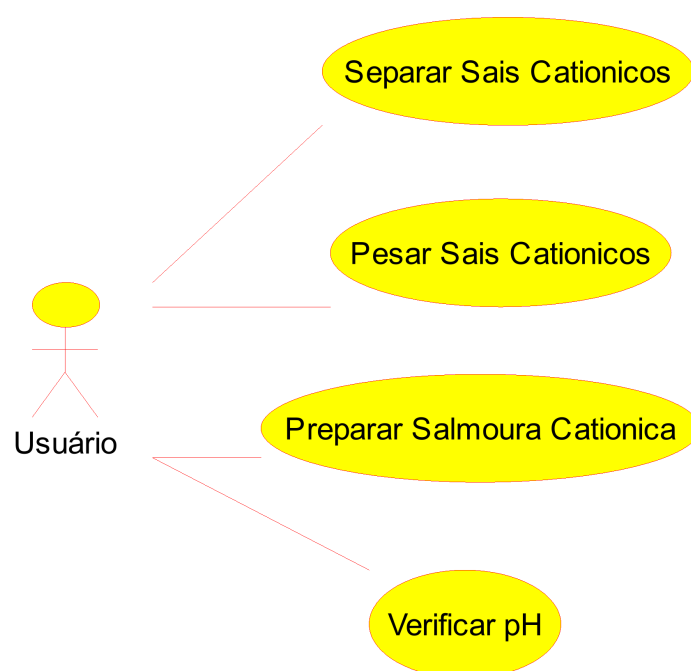


Figura 2.2: Diagrama de caso de uso específico – Caso de Uso Simulação de Salmoura

# 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

O sistema pertence à área de engenharia de petróleo e foca nos problemas relacionados à formação de

incrustações minerais em tubulações e equipamentos. As incrustações são formadas a partir da

precipitação de sais em águas de formação e representam um desafio significativo por reduzirem a

eficiência da produção.

### 3.2 Formulação teórica

O software se baseia na equação do produto iônico (Q):

$$Q = [A]^a * [B]^b$$

Onde [A] e [B] são as concentrações dos íons, e "a" e "b" seus coeficientes estequiométricos. A

comparação entre Q e o Kps (produto de solubilidade) define o estado do sistema:

- Se  $Q < Kps$ : solução insaturada (não precipita);

- Se  $Q = K_{ps}$ : solução saturada;
- Se  $Q > K_{ps}$ : ocorre precipitação.

### 3.3 Identificação de pacotes – assuntos

Pacote CConcentracaoIons: Responsável por armazenar e recuperar íons cadastrados.

Pacote CSal: Modela os sais e seus dados ( $K_{ps}$ , íons e coeficientes).

Pacote CCondicoesTermodinamicas: Armazena temperatura e pressão.

Pacote CCalcularPrecipitacao: Realiza os cálculos do produto iônico e compara com o  $K_{ps}$ .

Pacote CIon: representa íons com nome, concentração e carga.

### 3.4 Diagrama de pacotes – assuntos

O diagrama de pacotes mostra como os módulos se organizam e se comunicam. Cada pacote representa

uma responsabilidade específica no sistema.

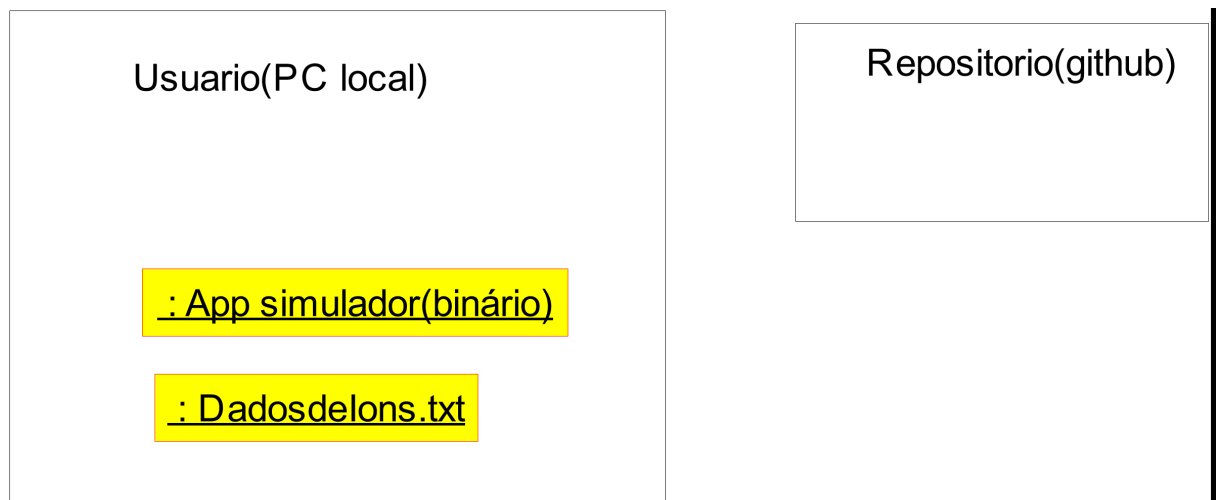


Figura 3.1: Diagrama de Pacotes



# 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 (Figura 3.1) mostra as classes do sistema, seus atributos e métodos, bem como

suas relações. As principais classes incluem:

CCondicoesTermodinamicas: atributos de pressão e temperatura, com getters/setters.

CIon: representa íons com nome, concentração e carga.

CConcentracaoIons: gerencia um mapa de íons.

CSal: modela sais com dois íons, seus coeficientes e o valor de Kps.

CCalcularPrecipitacao: módulo que executa a comparação entre Q e Kps.

### 4.2 Diagrama de seqüência – eventos e mensagens

O diagrama de seqüência (Figura 3.2) mostra a ordem das chamadas entre os objetos do sistema. O

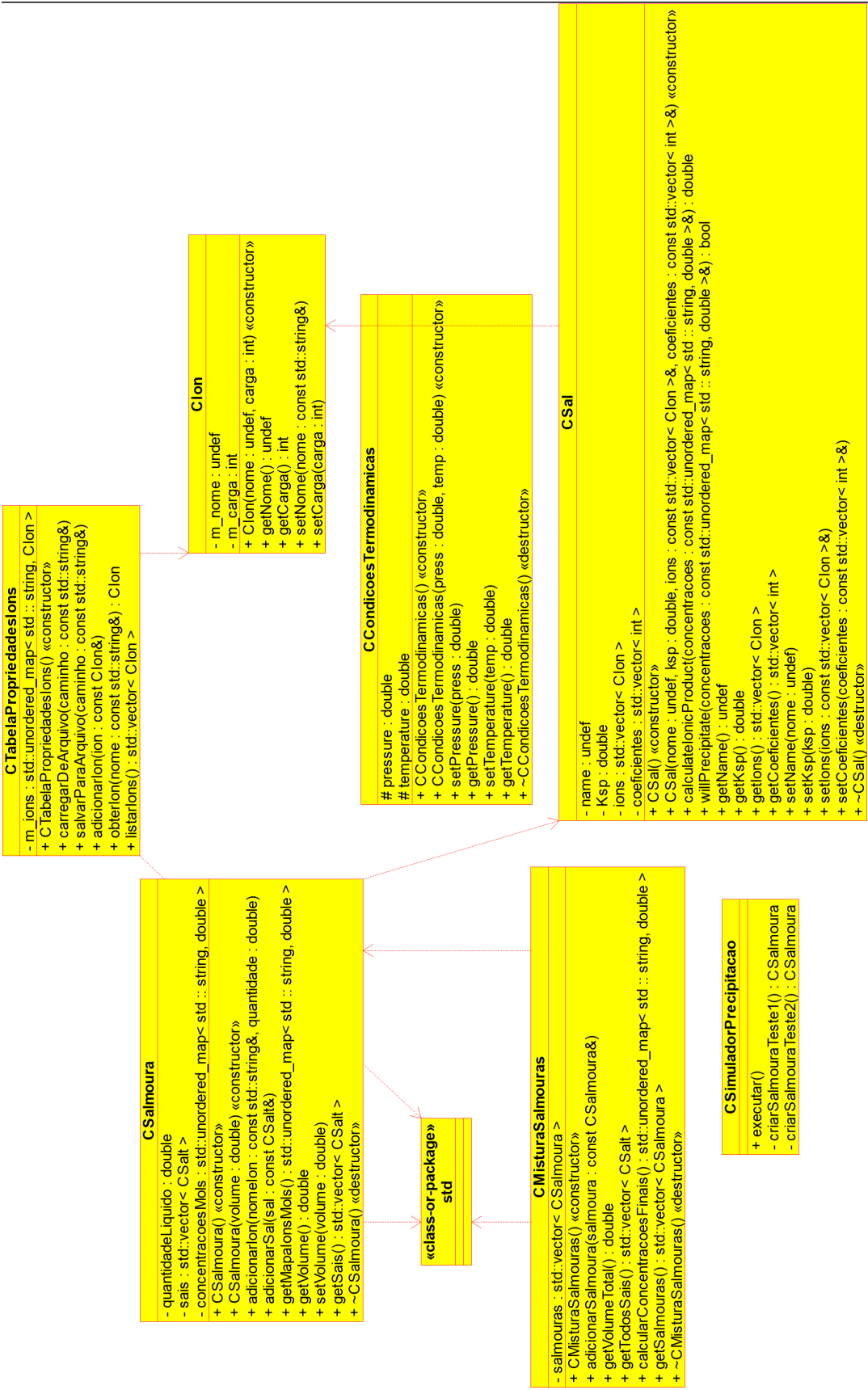


Figura 4.1: Diagrama de classes

fluxo inicia na main, passando pelas criações de íons, sais, entrada de condições termodinâmicas e

chamada do método AnalisePrecipitacao() da classe CCalcularPrecipitacao.

### 4.2.1 Diagrama de sequência geral

Veja o diagrama de sequência na Figura 4.2.

- [Aqui a ênfase é o entendimento da sequência com que as mensagens são trocadas, a ordem temporal.]

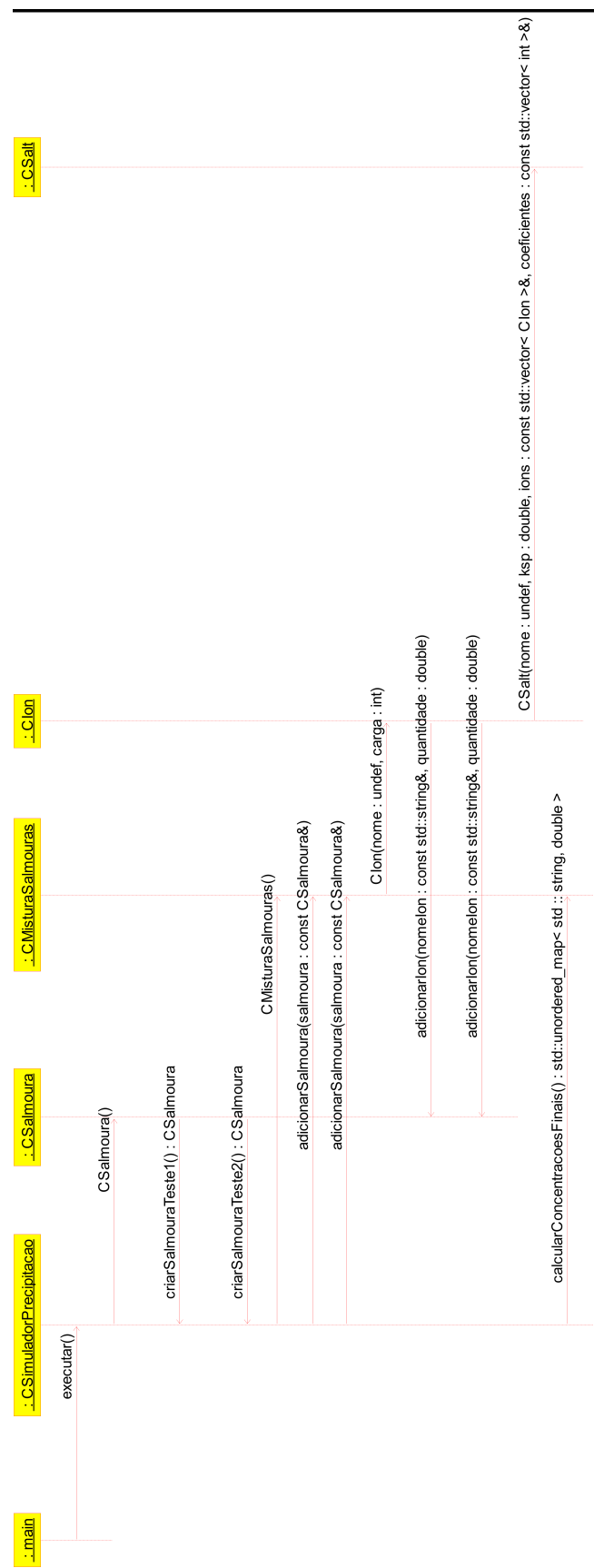


Figura 4.2: Diagrama de seqüência

### 4.2.2 Diagrama de sequência específico

Na Figura 3.2.2 observamos o diagrama de sequência específico.

## 4.3 Diagrama de comunicação – colaboração

O diagrama de estado (Figura 3.3) representa os estados principais de execução do sistema: criar dados

→ analisar dados → decidir se precipita ou não.

Figura 3.3 - Diagrama de Estado: representação dos estados do processo de análise

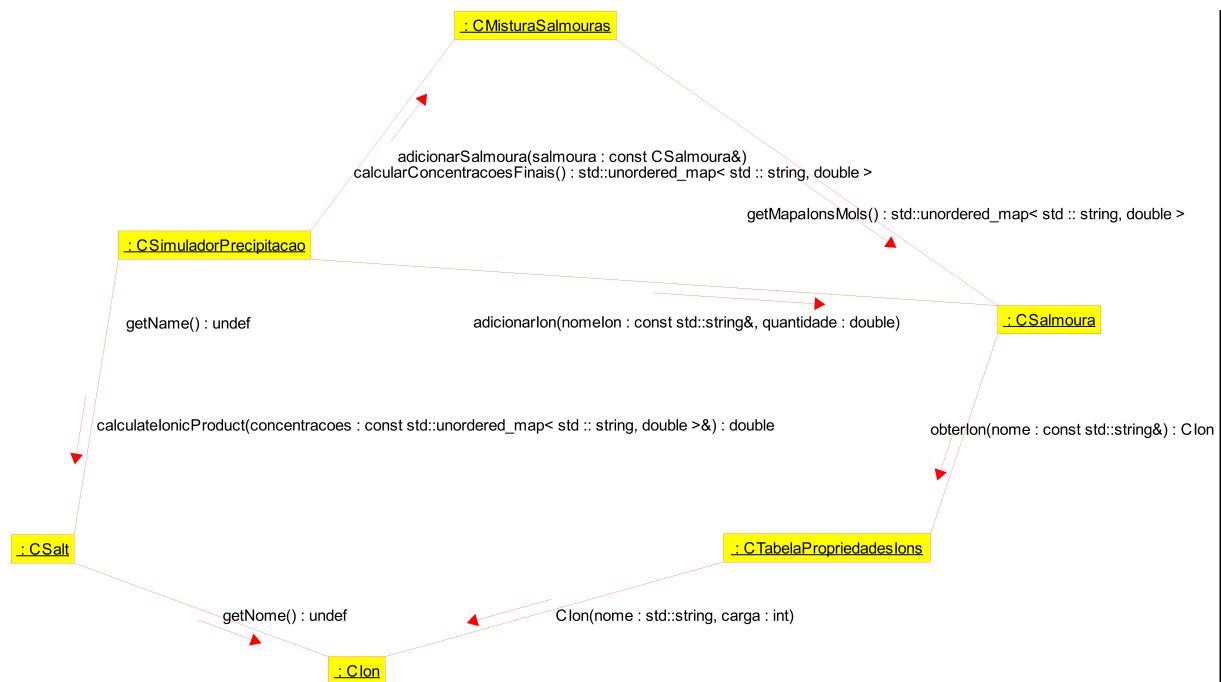


Figura 4.4: Diagrama de comunicação

## 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.



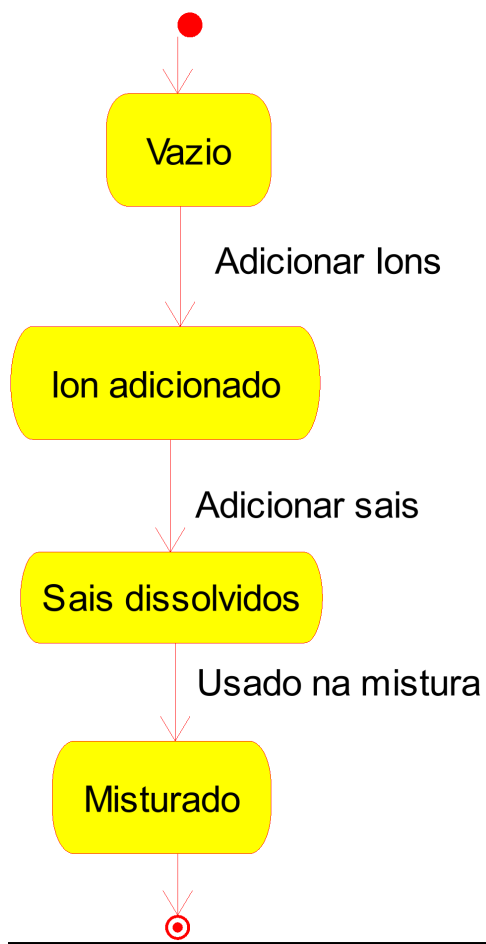


Figura 4.5: Diagrama de máquina de estado

## 4.5 Diagrama de atividades

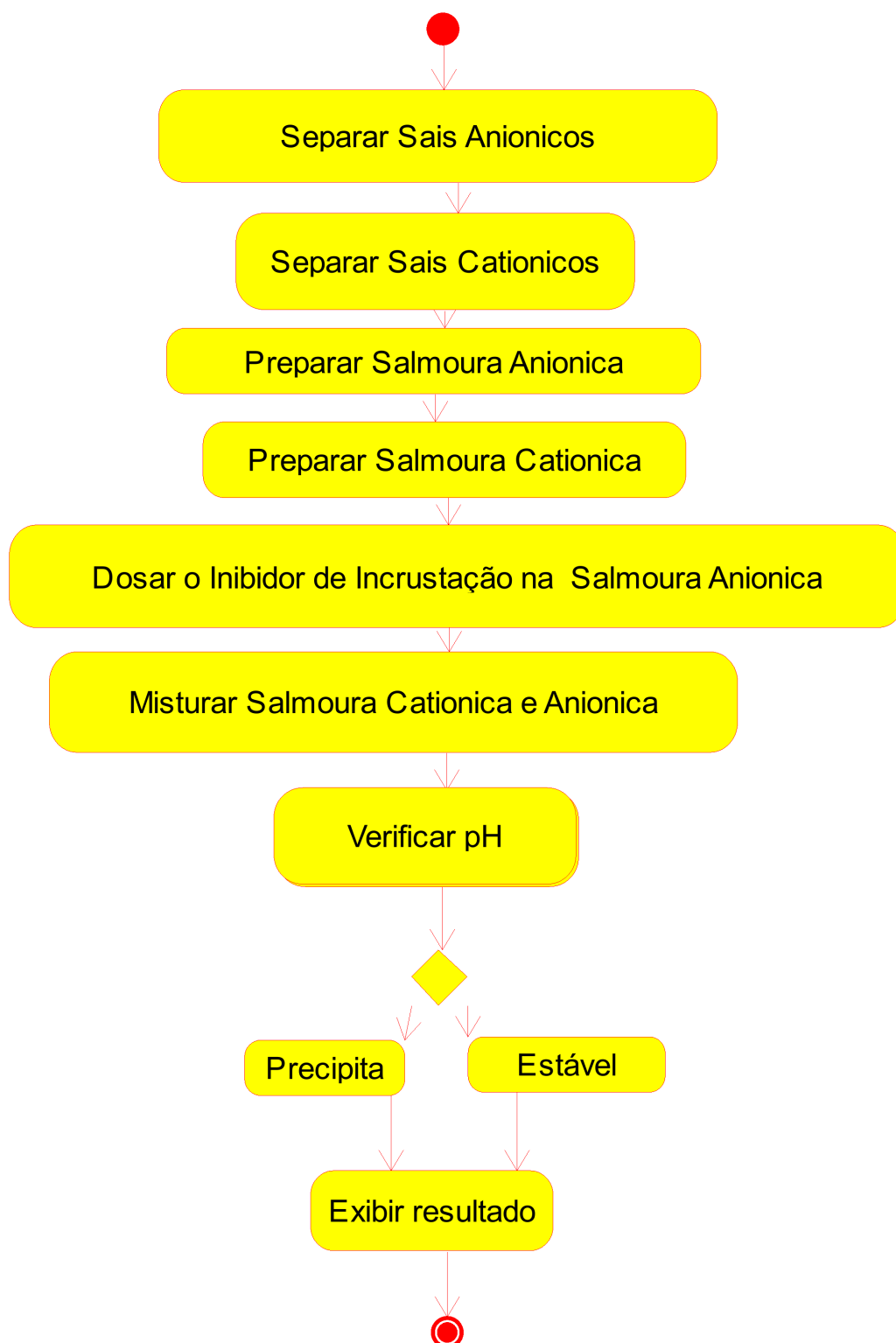


Figura 4.6: Diagrama de atividades



# 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

Depois da análise orientada a objeto desenvolve-se o projeto do sistema, qual envolve etapas como a definição dos protocolos, da interface API, o uso de recursos, a subdivisão do sistema em subsistemas, a alocação dos subsistemas ao hardware e a seleção das estruturas de controle, a seleção das plataformas do sistema, das bibliotecas externas, dos padrões de projeto, além da tomada de decisões conceituais e políticas que formam a infraestrutura do projeto.

Deve-se definir padrões de documentação, padrões para o nome das classes, padrões de retorno e de parâmetros em métodos, características da interface do usuário e características de desempenho.

Segundo [Rumbaugh et al., 1994, Blaha and Rumbaugh, 2006], o projeto do sistema é a estratégia de alto nível para resolver o problema e elaborar uma solução. Você deve se preocupar com itens como:

#### 1. Protocolos

- Definição dos protocolos de comunicação entre os diversos elementos externos  
Neste projeto não há comunicação com dispositivos externos, mas o sistema pode ser adaptado para ler sensores de pH ou condutividade via serial no futuro.
- A comunicação interna entre objetos ocorre por chamadas diretas a métodos públicos.
- A interface API é implementada implicitamente nas classes que encapsulam a lógica de precipitação e manipulação de dados (ex: `CalcularPrecipitacao`, `Sal`,

ion)

- Os arquivos gerados são .txt ou .csv, formatos abertos e compatíveis com editores comuns e softwares de análise.

## 2. Recursos

- Os recursos são gerenciados pelo próprio sistema, que armazena os dados em memória RAM. Não há alocação externa.
- Não há necessidade de banco de dados; todos os dados são temporários e manipulados em tempo de execução.
- O sistema não utiliza armazenamento de massa dedicado, apenas arquivos locais temporários para exportação de dados.

## 3. Controle

- O controle do sistema é sequencial, baseado em eventos gerados pela interação do usuário com a interface (via terminal ou GUI futura).
- O sistema prevê condições inválidas (ex: sais sem coeficientes definidos) com mensagens de erro.
- A otimização de código é feita com uso de bibliotecas padrão, evitando redundâncias.
- Loops de controle são utilizados nas análises de precipitação (ex: iteração sobre sais).
- O sistema não possui concorrência ou paralelismo nesta versão.

## 4. Plataformas

- O sistema segue arquitetura tradicional cliente-desktop.
- Os subsistemas identificados são: Entrada de dados, Cálculo, Interface, Termodinâmica e Química.
- O sistema suporta as plataformas Windows, Linux e Mac via compilação com CMake.
- As bibliotecas externas utilizadas são padrão do C++ (STL).

## 5.2 Projeto orientado a objeto – POO

O projeto orientado a objeto considera as decisões tomadas no projeto do sistema e implementa uma solução concreta. Foram feitas otimizações de métodos e atributos, modularização do código, uso de encapsulamento, clareza nos nomes e divisão lógica das responsabilidades.

### **Efeitos do projeto no modelo estrutural**

- Os diagramas foram atualizados com inclusão de bibliotecas e classes utilitárias.
- Foram criadas classes adicionais como CalcularPrecipitacao e CriarIon para modularização.
- Dependências entre Sal, Ion e CriarIons foram estabelecidas com composição.

### **Efeitos do projeto no modelo dinâmico**

- O diagrama de sequência foi ajustado para representar interações com o novo fluxo de entrada de dados e execução.
- Não há necessidade de novos diagramas de máquina de estado nesta versão.

### **Efeitos do projeto nos atributos**

- Foram incluídos atributos auxiliares como coef1, coef2, Ksp, concentração, carga, e nome com validadores nas classes.

### **Efeitos do projeto nos métodos**

- Métodos para análise de precipitação foram adicionados (CalcularProdutoIonico, VaiPrecipitar).
- Métodos foram organizados em funções públicas de controle e funções internas de cálculo.

### **Efeitos do projeto nas heranças**

- Não há uso de herança nesta versão para manter o sistema simples e modular. Futuras versões poderão introduzir polimorfismo para tipos de sais.

### **Efeitos do projeto nas associações**

- As associações são diretas (ex: Sal contém dois ions).....

## **5.3 Diagrama de componentes**

O diagrama de componentes mostra a forma como os módulos do software se relacionam entre si e suas dependências. No projeto Software de Análise de Incrustação de Amostras de Salmouras, cada pacote funcional se transforma em um componente lógico, facilitando a modularização e manutenção do código.

Na Figura 5.1 observa-se que:

- O componente ModuloEntradaDados é responsável pela criação e armazenamento de íons e sais a partir da entrada do usuário, utilizando as classes CCriarIons e CSal.
- O ModuloTermodinamico utiliza a classe CCondicoesTermodinamicas para registrar temperatura e pressão da salmoura.
- O ModuloAnalise depende de Sal, ion e CalcularPrecipitacao para processar os dados e verificar precipitação.
- O ModuloInterface agrupa os arquivos do main() e os módulos de teste, controlando a visualização no terminal (e futuramente em Qt).
- Todos esses componentes são compilados no executável final SimuladorPrecipitacao.exe. Nota:

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.

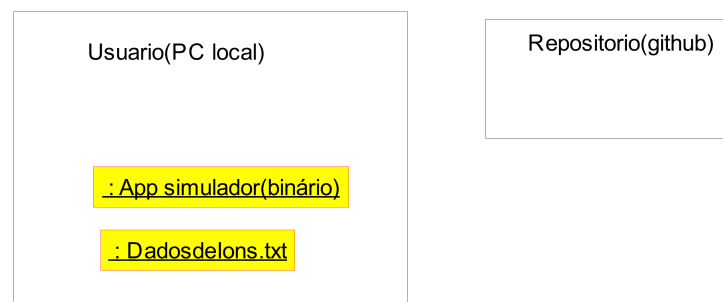


Figura 5.1: Diagrama de componentes

## 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.

No caso do Software de Análise de Incrustação de Amostras de Salmouras,, o sistema é executado localmente em um computador pessoal (desktop ou notebook), sem necessidade de rede. Todos os módulos estão compilados em um único executável e são armazenados localmente, sendo a execução feita por linha de comando ou futura interface gráfica.

Veja na Figura 5.2 o diagrama de implantação adaptado ao projeto. Ele mostra:

- Um nó denominado Computador Corporativo, com anotações do tipo {localização: laboratório de simulação}.

- Conectado a este nó está o executável `SimuladorPrecipitacao.exe`, que contém internamente os módulos `Analise`, `EntradaDados`, `Termodinamico` e `Interface`.
- Todos os arquivos `.cpp` e `.h` estão acessíveis localmente, não sendo necessário banco de dados nem rede para o funcionamento do sistema.

Essa arquitetura simples e portátil facilita a execução do sistema em ambientes acadêmicos e industriais, com poucos requisitos de hardware. O uso de arquivos `.txt` para entrada e saída também favorece integração com outros softwares externos.

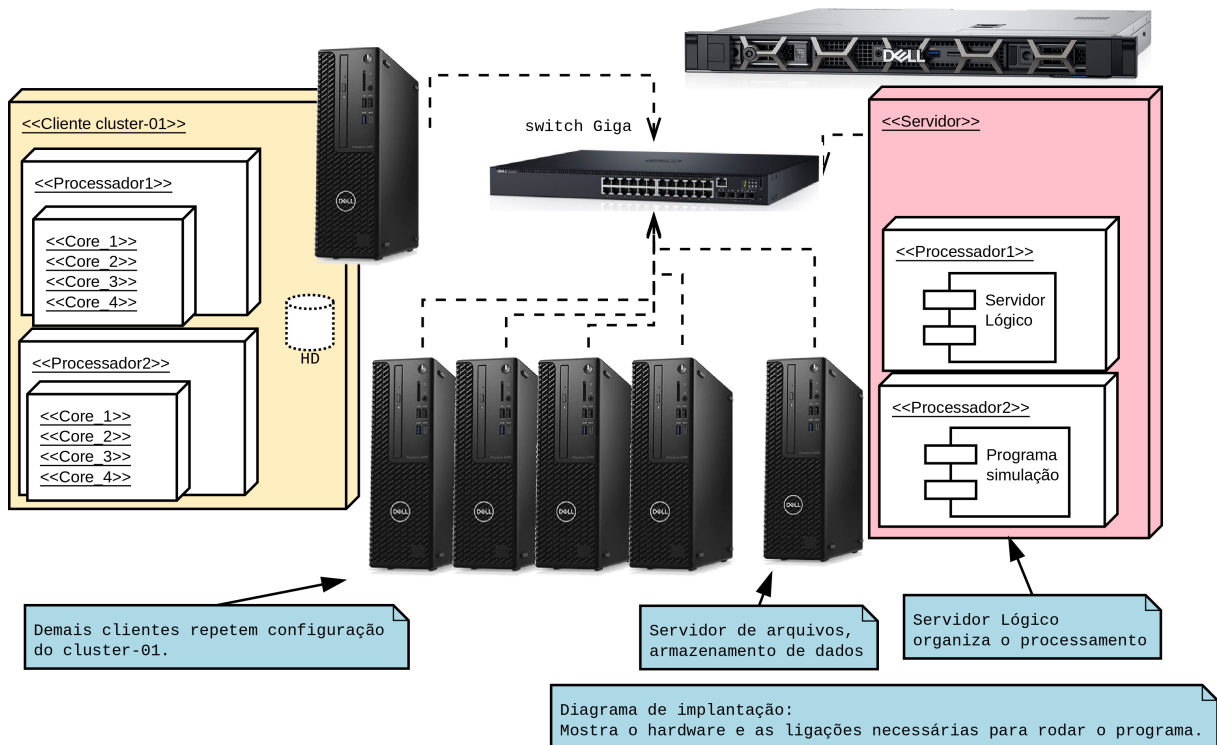


Figura 5.2: Diagrama de implantação

#### 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:

- v0.1
  - O sistema deve permitir o cadastro de íons com nome, concentração e carga elétrica.
  - O sistema deve permitir recuperar dados de íons previamente cadastrados (Ex: "Ca2+", "CO3\_2-").

- O sistema deve permitir definir um sal com nome, produto de solubilidade ( $K_{sp}$ ), dois íons e seus coeficientes estequiométricos.
  - O sistema deve calcular o produto iônico e compará-lo com o  $K_{sp}$  para verificar se há precipitação.
  - O sistema deve plotar a saída no terminal, indicando o nome do sal, o valor de  $Q$  (produto iônico), o valor do  $K_{sp}$  e se ocorre ou não precipitação.
  - O sistema deve permitir inserir condições termodinâmicas (pressão e temperatura), mesmo que ainda não influenciem os cálculos diretamente.
- v0.3
    - O sistema deve armazenar múltiplos sais em um vetor e avaliar todos com um único comando.
    - O sistema deve ser modular e permitir expansão para novos tipos de sais, com diferentes coeficientes estequiométricos.
    - O sistema deve incluir mensagens mais claras para o usuário final (ex: usar nomes comerciais dos sais).
  - v0.7
    - O sistema deve apresentar uma interface simples (GUI) ou permitir leitura de dados a partir de arquivos externos (.txt, .dat ou .csv).
    - O sistema deve plotar gráficos de pressão vs. temperatura, simulando cenários reais de precipitação de sais.
    - O sistema deve considerar a influência da temperatura e pH sobre o  $K_{sp}$  (ex: ajuste com base em curvas experimentais).
    - O sistema deve gerar relatórios automáticos com os resultados da análise (ex: quais sais precipitam em quais condições).

#### 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:	<input checked="" type="checkbox"/> livre GPL-v3 <input type="checkbox"/> proprietária
Engenharia de software:	<input type="checkbox"/> tradicional <input checked="" type="checkbox"/> ágil <input type="checkbox"/> outras
Paradigma de programação:	<input type="checkbox"/> estruturada <input checked="" type="checkbox"/> orientado a objeto - POO <input type="checkbox"/> 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] aplicativos [ ] 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 [X] híbrida (texto e saídas gráficas) [ ] 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>smarth 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</i> , <i>join</i> e <i>mutex</i> ). Bibliotecas: <i>random</i> , <i>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</i> , <i>múltiplos processos</i>

Bibliotecas de C++:	<input checked="" type="checkbox"/> Entrada e saída de dados ( <i>streams</i> ) <input checked="" type="checkbox"/> <i>cmath</i> <input checked="" type="checkbox"/> <i>filesystem</i> <input type="checkbox"/> <i>random</i> <input checked="" type="checkbox"/> <i>threads</i> <input type="checkbox"/> <i>expressões regulares</i> <input type="checkbox"/> <i>múltiplos processos</i>
Bibliotecas externas:	<input checked="" type="checkbox"/> CGnuplot <input checked="" type="checkbox"/> QCustomPlot <input type="checkbox"/> Qt diálogos <input type="checkbox"/> QT Janelas/menus/BT_____
Ferramentas auxiliares:	Montador: <input checked="" type="checkbox"/> make <input type="checkbox"/> cmake <input checked="" type="checkbox"/> qmake
IDE:	<input checked="" type="checkbox"/> Editor simples: kate/gedit/emacs <input type="checkbox"/> kdevelop <input type="checkbox"/> QT-Creator <input type="checkbox"/> _____
SCV:	<input type="checkbox"/> cvs <input type="checkbox"/> svn <input checked="" type="checkbox"/> git
Disciplinas correlacionadas	<input type="checkbox"/> estatística <input type="checkbox"/> cálculo numérico <input type="checkbox"/> modelamento numérico <input checked="" type="checkbox"/> 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 versão inicial, a entrada e saída de dados foram implementadas exclusivamente por meio do terminal, sem a utilização de bibliotecas gráficas. A geração e visualização dos gráficos foram realizadas com o auxílio do Gnuplot, oferecendo uma maneira prática e objetiva de apresentar os resultados ao usuário. Todo o desenvolvimento ocorreu em um ambiente de terminal, utilizando o sistema operacional Windows 11.

# Capítulo 7

## Ciclos Construção - Implementação

Neste capítulo, são apresentados os códigos fonte implementados.

**Nota:** os códigos devem ser documentados usando padrão **javadoc**. Posteriormente usar o programa **doxygen** para gerar a documentação no formato html.

- Veja informações gerais aqui <http://www.doxygen.org/>.
- Veja exemplo aqui <http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html>.

**Nota:** ao longo deste capítulo usamos inclusão direta de arquivos externos usando o pacote *listings* do L<sup>A</sup>T<sub>E</sub>X. Maiores detalhes de como a saída pode ser gerada estão disponíveis nos links abaixo.

- [http://en.wikibooks.org/wiki/LaTeX/Source\\_Code\\_Listings](http://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings).
- <http://mirrors.ctan.org/macros/latex/contrib/listings/listings.pdf>.

### 7.1 Código fonte

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

Apresenta-se na listagem 7.1o arquivo com código da classe CIons.

Código 7.1: Arquivo de cabeçalho da classe CIons

```
#ifndef CIONS_H
#define CIONS_H

#include <iostream>
#include <unordered_map>

/**
 * @brief Estrutura que representa um íon na solução química.
 */
```

```
* @param name Nome do íon (ex: "Ca2+").
* @param charge Carga elétrica do íon.
*/

class CIon {
public:

    /**
     * @brief Construtor da estrutura ion.
     *
     * @param _name Nome do íon.
     * @param _charge Carga do íon.
     */
    CIon(std::string _nome = "", int _carga = 0);

    std::string getNome() const;
    int getCarga() const;

private:
    std::string nome;
    int carga;
};

/**
 * @brief Classe responsável pela criação e gerenciamento de íons.
 *
 * @param ions Mapa ('unordered_map') que armazena os íons criados.
 */
class CreateIons {
public:

    // Construtor
    CreateIons();

    // Adiciona um íon ao mapa
    void addIon(const std::string& _name, int _charge);
    // Obtém um íon do mapa
    CIon getIon(const std::string& _name);

    // Destrutor
    ~CreateIons();

protected:
    // Contêiner de íons armazenados
    std::unordered_map<std::string, CIon> ions;
};
```

```
#endif
```

Apresenta-se na listagem ??o arquivo com código da classe CSal.

Código 7.2: Arquivo de cabeçalho da classe CSalt

```
#ifndef CSALT_H
#define CSALT_H

#include "CIons.h"
#include <vector>
#include <string>
#include <unordered_map>

/**
 * @brief Classe responsável por modelar um sal e calcular sua
        solubilidade.
 *
 * Um sal é formado por uma combinação de íons com coeficientes
        estequiométricos conhecidos.
 * Sua precipitação depende do produto iônico comparado ao Ksp (
        constante de solubilidade).
 */
class CSalt {

public:

    /**
     * @brief Construtor completo do sal.
     * @param nome Nome do sal
     * @param kspRef Ksp de referência
     * @param deltaH  $\Delta H^\circ$  de dissolução (J/mol)
     * @param temperaturaRef Temperatura de referência (K)
     * @param ions Vetor de íons
     * @param coeficientes Coeficientes estequiométricos
     */
    CSalt(std::string nome, double kspRef, double deltaH, double
        temperaturaRef, const std::vector<CIon>& ions, const std::vector<
        int>& coeficientes);

    /**
     * @brief Corrige o Ksp com base na temperatura atual.
     * @param temperatura Temperatura (K)
     * @return Ksp corrigido
     */
    double KspCorrigido(double temperatura) const;

    /**
```

```

    * @brief Calcula o produto iônico Q do sal com base nas
      concentrações fornecidas.
    *
    * @param concentracoes Mapa nome-do-íon ? concentração (mol/L)
    * @return Valor de Q (produto iônico)
    */
double calculateIonicProduct(const std::unordered_map<std::string,
    double>& concentracoes) const;

/**
    * @brief Verifica se o sal irá precipitar.
    * @param concentracoes Mapa de concentração molar por íon
    * @param temperatura Temperatura (K)
    * @return true se  $Q > K_{sp\_corrigido}$ 
    */
bool willPrecipitate(const std::unordered_map<std::string, double>&
    concentracoes, double temperatura) const;

/// @return Nome do sal.
std::string getName() const;

/// @return Produto de solubilidade do sal ( $K_{sp}$ ).
double getKsp() const;

/// @return Vetor de íons que compõem o sal.
std::vector<CIon> getIons() const;

/// @return Vetor de coeficientes estequiométricos.
std::vector<int> getCoeficientes() const;

/// Define o nome do sal.
void setName(std::string nome);

/// Define o valor do  $K_{sp}$ .
void setKsp(double ksp);

/// Define os íons do sal.
void setIons(const std::vector<CIon>& ions);

/// Define os coeficientes dos íons.
void setCoeficientes(const std::vector<int>& coeficientes);

/// Destrutor.
~CSalt();

private:
    std::string name;                // Nome do sal
    double kspRef;                   //  $K_{sp}$  de referência (em

```

```

        temperaturaRef)
double deltaH;                // ?H° de dissolução (J/mol)
double temperaturaRef;        // Temperatura de referência (K)
std::vector<CIon> ions;        // Íons que compõem o sal
std::vector<int> coeficientes; // Coeficientes estequiométricos
};

#endif

```

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

### Código 7.3: Arquivo de cabeçalho da classe CSalmoura

```

#ifndef CSALMOURA_H
#define CSALMOURA_H

#include "CSalt.h"
#include <unordered_map>
#include <vector>
#include <string>

/**
 * @brief Representa uma amostra de salmoura composta por água e sais
        dissolvidos.
 *
 * Contém informações sobre o volume total da água, os sais adicionados
        e a quantidade (em mols) de cada íon presente.
 */
class CSalmoura {

public:

    /**
     * @brief Construtor padrão.
     */
    CSalmoura();

    /**
     * @brief Construtor parametrizado.
     *
     * @param volume Volume da salmoura em litros.
     */
    CSalmoura(double volume);

    /**
     * @brief Define condições termodinamicas.
     *
     * @param temperaturaK temperatura em keven
     * @param quantidade pressao atm.
     */

```

```

void DefinirCondicoes(double temperaturaK, double pressaoAtm);

/**
 * @brief Define a quantidade (em mols) de um íon na solução.
 *
 * @param nomeIon Nome do íon (ex: "Na").
 * @param quantidade Número de mols adicionados.
 */
void adicionarIon(const std::string& nomeIon, double mols);

/**
 * @brief Obtém todos os íons e suas quantidades (mols).
 *
 * @return Mapa nome-do-íon ? número de mols.
 */
std::unordered_map<std::string, double> getMapaIonsMols() const;

/**
 * @brief Obtém o volume da salmoura (em litros).
 */
double getVolume() const;

/**
 * @brief Obtém a temperatura da salmoura (em litros).
 */
double getTemperatura() const;

/**
 * @brief Define o volume da salmoura.
 *
 * @param volume Volume em litros.
 */
void setVolume(double volume);

~CSalmoura();

private:
    double quantidadeLiquido; /**< Volume total da água em litros */
    double temperatura; ///< Temperatura da salmoura (em Kelvin)
    double pressao; ///< Pressão da salmoura (em atm)
    std::unordered_map<std::string, double> concentracoesMols; /**<
        Quantidade de mols de cada íon */
};

#endif

```

Apresenta-se na listagem ??o arquivo com código da classe CMisturaSalmoura.

Código 7.4: Arquivo de cabeçalho da classe CMisturaSalmouras

```
#ifndef CMISTURA_SALMOURAS_H
#define CMISTURA_SALMOURAS_H

#include "CSalmoura.h"
#include <vector>
#include <string>

/**
 * @brief Representa uma mistura de várias amostras de salmouras com
 *        diferentes volumes.
 *
 * Combina as contribuições de cada salmoura e permite cálculo das
 * concentrações finais de íons.
 */
class CMisturaSalmouras {

public:

    /**
     * @brief Construtor padrao.
     */
    CMisturaSalmouras();

    const std::vector<CSalmoura>& getSalmouras() const;

    /**
     * @brief Adiciona uma salmoura é mistura.
     *
     * @param salmoura Objeto 'CSalmoura'.
     */
    void adicionarSalmoura(const CSalmoura& salmoura);

    /**
     * @brief Retorna um mapa de concentrações finais (mol/L) de cada íon
     *        na mistura.
     */
    std::unordered_map<std::string, double> calcularConcentracoesFinais
        () const;

    ~CMisturaSalmouras();

private:
    std::vector<CSalmoura> salmouras; /**< Lista de salmouras
        adicionadas */
};

#endif
```



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

Código 7.5: Arquivo de cabeçalho da classe CSimuladorPrecipitacao

```
#ifndef CSIMULADORPRECIPITACAO_H
#define CSIMULADORPRECIPITACAO_H

#include "CMisturaSalmouras.h"
#include "CSalt.h"
#include "CTabelaPropriedadesIons.h"
#include "CPlotConfig.h"
#include <vector>

/**
 * @class CSimuladorPrecipitacao
 * @brief Classe responsável por gerenciar a interação com o usuário e a
 *        execução da simulação química.
 *
 * Esta classe encapsula a lógica de interação via terminal, coleta de
 * dados como temperatura e pressão,
 * construção das salmouras, sais, e delega os cálculos de precipitação
 * com base nas concentrações finais.
 */
class CSimuladorPrecipitacao {
private:
    CTabelaPropriedadesIons tabelaIons;
    std::vector<CSalt> saisDisponiveis;
    size_t m_Quality = 1;
    QualidadePlot qualidade = QualidadePlot::Alta;

    void ConstruirSalmoura(CSalmoura& s) const;
    void CarregarSais(const std::string& caminhoArquivo);
    void ListarArquivosTxt(const std::string& titulo, std::string&
        selecionado) const;

public:
    /**
     * @brief Executa a interface interativa de terminal com o usuário.
     *
     * Permite definir condições termodinâmicas, criar salmouras, sais e
     * executar a simulação completa.
     */
    void executar();

    /**
     * @brief Executa a simulação de precipitação com base na mistura e
     *        nos sais fornecidos.
     * @param mistura Mistura de salmouras criada pelo usuário.
     */
}
```

```

        void simular(const CMisturaSalmouras& mistura);
};

#endif

```

Apresenta-se na listagem 7.6o arquivo com código da classe CTabelaPropriedadesIons.

Código 7.6: Arquivo de cabeçalho da classe CTabelaPropriedadesIons

```

#ifndef CTABELA_PROPRIEDADES_IONS_H
#define CTABELA_PROPRIEDADES_IONS_H

#include "CIons.h"
#include <string>
#include <unordered_map>
#include <vector>

/**
 * @brief Classe responsável por carregar, armazenar e fornecer acesso
 * a Ions conhecidos da literatura.
 *
 * Essa tabela é persistente e pode ser carregada de ou salva em
 * arquivos do disco, contendo dados como nome e carga do Ion.
 */
class CTabelaPropriedadesIons {

public:

    /**
     * @brief Construtor padrão da tabela de Ions.
     */
    CTabelaPropriedadesIons();

    /**
     * @brief Carrega os Ions de um arquivo texto (formato CSV: nome;
     * carga).
     *
     * @param caminho Caminho para o arquivo a ser lido.
     */
    void carregarDeArquivo(const std::string& caminho);

    /**
     * @brief Salva os Ions da tabela em um arquivo texto (formato CSV:
     * nome;carga).
     *
     * @param caminho Caminho para o arquivo de destino.
     */
    void salvarParaArquivo(const std::string& caminho) const;

    /**

```

```

    * @brief Adiciona um novo ã on ã tabela.
    *
    * @param ion Objeto CIon contendo nome e carga do ã on.
    */
void adicionarIon(const CIon& ion);

/**
 * @brief Retorna um ã on a partir de seu nome.
 *
 * @param nome Nome do ã on a ser buscado.
 * @return Objeto CIon correspondente (ou vazio, caso não exista).
 */
CIon obterIon(const std::string& nome) const;

/**
 * @brief Retorna todos os ã ons da tabela como um map.
 *
 * @return mapa contendo todos os ã ons armazenados.
 */
const std::unordered_map<std::string, CIon>& getMapa() const;

private:
    std::unordered_map<std::string, CIon> m_ions; /**< Mapa de ã ons:
        nome -> CIon */
};

#endif

```

Apresenta-se na listagem 7.7 o arquivo de implementação da classe CPlotPrecipitacao.

Código 7.7: Arquivo de cabeçalho da classe CPlotPrecipitacao

```

#ifndef CGNUPLOTPLOT_H
#define CGNUPLOTPLOT_H

#include "CPlotConfig.h"

#include <string>
#include <vector>

class CPlotPrecipitacao {
public:

    CPlotPrecipitacao(QualidadePlot q = QualidadePlot::Alta) :
        config(q) {}

    void PlotQvsKsp(const std::vector<std::string>& nomes,
                   const std::vector<double>& qs,
                   const std::vector<double>& ksps);

```

```

        void PlotQvsKspVsTemperatura(const std::string& nomeSal,
                                     double qFixo,
                                     const std::vector<double>&
                                     temperaturas,
                                     const std::vector<double>& ksps);

        void MultiplotQvsKspVsTemperatura(const std::vector<std::string
                                     >& nomesSalmouras,

                                     const std::vector<std::vector<
                                     std::string>>&
                                     nomesSaisPorSalmoura,
                                     const std::vector<std::vector<
                                     double>>& todosQs,
                                     const std::vector<std::vector<
                                     double>>& todosKsps);

    private:
        CPlotConfig config;
};

#endif

```

Apresenta-se na listagem 7.8o arquivo com código da classe CIons.

Código 7.8: Arquivo de implementação da classe CIons

```

#include "CIons.h"

CIon::CIon(std::string nome, int carga) : nome(nome), carga(carga)
{
}

std::string CIon::getNome() const
{
    return nome;
}

int CIon::getCarga() const
{
    return carga;
}

CreateIons::CreateIons()
{
}

void CreateIons::addIon(const std::string &name, int charge)
{
    ions[name] = CIon(name, charge);
}

```

```

}

CIon CreateIons::getIon(const std::string &name)
{
    if (ions.find(name) != ions.end()) {
        return ions[name];
    } else {
        std::cerr << "Erro: ?on '" << name << "' n?o encontrado!\n";
        return CIon(); // Retorna um ?on vazio
    }
}

CreateIons::~CreateIons()
{
}

```

Apresenta-se na listagem 7.9o arquivo com código da classe CSalt.

Código 7.9: Arquivo de implementação da classe CSalt

```

#include "CSalt.h"
#include <cmath>
#include <iostream>

CSalt::CSalt(std::string nome, double kspRef, double deltaH, double
    temperaturaRef,
    const std::vector<CIon>& ions, const std::vector<int>&
    coeficientes)
    : name(nome), kspRef(kspRef), deltaH(deltaH), temperaturaRef(
    temperaturaRef), ions(ions), coeficientes(coeficientes) {}

double CSalt::calculateIonicProduct(const std::unordered_map<std::string
    , double>& concentracoes) const {
    double Q = 1.0;

    for (size_t i = 0; i < ions.size(); ++i) {
        std::string nomeIon = ions[i].getNome();
        auto it = concentracoes.find(nomeIon);
        if (it != concentracoes.end()) {
            Q *= std::pow(it->second, coeficientes[i]);
        } else {
            std::cerr << "Aviso: concentração para íon " << nomeIon << "
                não encontrada. Q = 0\n";
            return 0.0;
        }
    }

    return Q;
}

```

```
double CSalt::KspCorrigido(double temperatura) const {
    constexpr double R = 8.314;
    double lnK = std::log(kspRef);
    double ajuste = -deltaH / R * (1.0 / temperatura - 1.0 /
        temperaturaRef);
    return std::exp(lnK + ajuste);
}

bool CSalt::willPrecipitate(const std::unordered_map<std::string, double
    >& concentracoes, double temperatura) const {
    double Q = calculateIonicProduct(concentracoes);
    double kspAtual = KspCorrigido(temperatura);

    return Q > kspAtual;
}

std::string CSalt::getName() const {
    return name;
}

double CSalt::getKsp() const {
    return kspRef;
}

std::vector<CIon> CSalt::getIons() const {
    return ions;
}

std::vector<int> CSalt::getCoeficientes() const {
    return coeficientes;
}

void CSalt::setName(std::string _name) {
    name = _name;
}

void CSalt::setKsp(double _Kps) {
    kspRef = _Kps;
}

void CSalt::setIons(const std::vector<CIon>& _ions) {
    ions = _ions;
}

void CSalt::setCoeficientes(const std::vector<int>& _coef) {
    coeficientes = _coef;
}
```

```
CSalt::~~CSalt() {}
```

Apresenta-se na listagem 7.10 o arquivo de implementação da classe CSalmoura.

Código 7.10: Arquivo de implementação da classe CSalmoura

```
#include "CSalmoura.h"

CSalmoura::CSalmoura()
    : quantidadeLiquido(1.0) {}

CSalmoura::CSalmoura(double volume)
    : quantidadeLiquido(volume) {}

void CSalmoura::adicionarIon(const std::string& nomeIon, double mols) {
    concentracoesMols[nomeIon] += mols;
}

void CSalmoura::DefinirCondicoes(double temperaturaK, double pressaoAtm)
{
    temperatura = temperaturaK;
    pressao = pressaoAtm;
}

std::unordered_map<std::string, double> CSalmoura::getMapaIonsMols()
const {
    return concentracoesMols;
}

double CSalmoura::getVolume() const {
    return quantidadeLiquido;
}

double CSalmoura::getTemperatura() const
{
    return temperatura;
}

void CSalmoura::setVolume(double volume) {
    quantidadeLiquido = volume;
}

CSalmoura::~~CSalmoura() {}
```

Apresenta-se na listagem 7.11o arquivo com código da classe CMisturaSalmouras.

Código 7.11: Arquivo de implementação da classe CMisturaSalmouras

```
#include "CMisturaSalmouras.h"
```

```

CMisturaSalmouras::CMisturaSalmouras() {}

void CMisturaSalmouras::adicionarSalmoura(const CSalmoura& salmoura) {
    salmouras.push_back(salmoura);
}

std::unordered_map<std::string, double> CMisturaSalmouras::
    calcularConcentracoesFinais() const {
    std::unordered_map<std::string, double> acumulado;
    double totalVolume = 0.0;

    for (const auto& s : salmouras) {
        const auto mapa = s.getMapaIonsMols();
        double v = s.getVolume();
        totalVolume += v;

        for (const auto& [ion, mols] : mapa) {
            acumulado[ion] += mols;
        }
    }

    for (auto& [ion, totalMols] : acumulado) {
        totalMols /= totalVolume;
    }

    return acumulado;
}

const std::vector<CSalmoura>& CMisturaSalmouras::getSalmouras() const {
    return salmouras;
}

CMisturaSalmouras::~~CMisturaSalmouras() {}

```

Apresenta-se na listagem 7.12 o arquivo de implementação da classe CSimuladorPrecipitacao.

Código 7.12: Arquivo de implementação da classe CSimuladorPrecipitacao

```

#include "CSimuladorPrecipitacao.h"
#include "CPlotPrecipitacao.h"
#include <fstream>
#include <sstream>
#include <iostream>
#include <filesystem>

namespace fs = std::filesystem;

void CSimuladorPrecipitacao::executar() {

```



```

std::cout << "\nEscolha a qualidade dos graficos:\n";
std::cout << "[1] Alta (1920x1080)\n";
std::cout << "[2] Media (1280x720)\n";
std::cout << "[3] Baixa (800x600)\n";
std::cout << "Selecao: ";

std::cin >> m_Quality;

switch (m_Quality) {
    case 2: qualidade = QualidadePlot::Media; break;
    case 3: qualidade = QualidadePlot::Baixa; break;
    default: qualidade = QualidadePlot::Alta; break;
}

std::string arqIons;
ListarArquivosTxt("Selecione o arquivo de ions:", arqIons);
tabelaIons.carregarDeArquivo(arqIons);

std::string arqSais;
ListarArquivosTxt("Selecione o arquivo de sais:", arqSais);
CarregarSais(arqSais);

CMisturaSalmouras mistura;
int num;
std::cout << "Quantas salmouras deseja adicionar? ";
std::cin >> num;

for (int i = 0; i < num; ++i) {
    CSalmoura s;
    std::cout << "\n--- Salmoura " << (i + 1) << " ---" << std::endl
        ;
    ConstruirSalmoura(s);
    mistura.adicionarSalmoura(s);
}

simular(mistura);
}

void CSimuladorPrecipitacao::ConstruirSalmoura(CSalmoura& s) const {
    double v, t, p;
    std::cout << "Volume (L): ";
    std::cin >> v;
    std::cout << "Temperatura (K): ";
    std::cin >> t;
    std::cout << "Pressao (atm): ";
    std::cin >> p;
    s.setVolume(v);

```

```

s.DefinirCondicoes(t, p);

std::string arqSalmoura;
ListarArquivosTxt("Selecione o arquivo de ions da salmoura:",
    arqSalmoura);
std::ifstream arq("./dados/" + arqSalmoura);
if (!arq.is_open()) {
    std::cerr << "Erro ao abrir " << arqSalmoura << std::endl;
    return;
}

std::string linha;
while (std::getline(arq, linha)) {
    if (linha.empty() || linha[0] == '#') continue;
    std::string nomeIon;
    double mols;
    std::replace(linha.begin(), linha.end(), '\t', ' '); //
        Normaliza tabs para espaços
    std::istringstream iss(linha);
    if (iss >> nomeIon >> mols) {
        s.adicionarIon(nomeIon, mols);
    }
}
}

void CSimuladorPrecipitacao::CarregarSais(const std::string&
    caminhoArquivo) {
    std::ifstream arq("./dados/" + caminhoArquivo);
    if (!arq.is_open()) return;

    std::string linha;
    while (std::getline(arq, linha)) {
        if (linha.empty() || linha[0] == '#') continue;
        std::istringstream iss(linha);
        std::string nome, ion1str, ion2str;
        double kspRef, deltaH, tempRef;
        iss >> nome >> kspRef >> deltaH >> tempRef >> ion1str >> ion2str
            ;

        std::vector<CIon> ions;
        std::vector<int> coefs;
        for (const std::string& entrada : {ion1str, ion2str}) {
            size_t sep = entrada.find(":");
            if (sep == std::string::npos) continue;
            std::string nomeIon = entrada.substr(0, sep);
            int coef = std::stoi(entrada.substr(sep + 1));
            ions.push_back(tabelaIons.obterIon(nomeIon));
            coefs.push_back(coef);
        }
    }
}

```

```

        }
        saisDisponiveis.emplace_back(nome, kspRef, deltaH, tempRef, ions
            , coefs);
    }
}

void CSimuladorPrecipitacao::ListarArquivosTxt(const std::string& titulo
    , std::string& selecionado) const {
    std::vector<std::string> arquivos;
    std::cout << "\n" << titulo << std::endl;
    int index = 1;
    for (const auto& entry : fs::directory_iterator("./dados/")) {
        if (entry.is_regular_file() && entry.path().extension() == ".txt") {
            std::cout << "[" << index << "]" " " << entry.path().filename()
                .string() << std::endl;
            arquivos.push_back(entry.path().filename().string());
            ++index;
        }
    }
    if (arquivos.empty()) {
        std::cerr << "Nenhum arquivo .txt encontrado." << std::endl;
        exit(1);
    }
    int escolha = 0;
    do {
        std::cout << "Digite o numero do arquivo desejado: ";
        std::cin >> escolha;
    } while (escolha < 1 || escolha > static_cast<int>(arquivos.size()))
        ;

    selecionado = arquivos[escolha - 1];
}

void CSimuladorPrecipitacao::simular(const CMisturaSalmouras& mistura) {
    const auto& salmouras = mistura.getSalmouras();
    CPlotPrecipitacao plot(qualidade);

    // CASO MULTILOT DE BARRAS: mais de uma salmoura
    if (salmouras.size() > 1) {
        std::vector<std::string> nomesSalmouras;
        std::vector<std::vector<std::string>> todosNomesSais;
        std::vector<std::vector<double>> todosQs;
        std::vector<std::vector<double>> todosKsps;

        for (size_t idx = 0; idx < salmouras.size(); ++idx) {
            const auto& salmoura = salmouras[idx];
            std::unordered_map<std::string, double> conc = salmoura.

```

```

        getMapaIonsMols();
        double T = salmoura.getTemperatura();

        std::vector<std::string> nomesSais;
        std::vector<double> qs;
        std::vector<double> ksps;

        for (const auto& sal : saisDisponiveis) {
            nomesSais.push_back(sal.getName());
            qs.push_back(sal.calculateIonicProduct(conc));
            ksps.push_back(sal.KspCorrigido(T));
        }

        nomesSalmouras.push_back("Salmoura " + std::to_string(idx +
            1));
        todosNomesSais.push_back(nomesSais);
        todosQs.push_back(qs);
        todosKsps.push_back(ksps);
    }

    // AQUI ESTAVA O ERRO: você chamava o método errado
    plot.MultiplotQvsKspVsTemperatura(nomesSalmouras, todosNomesSais
        , todosQs, todosKsps);
    return;
}

// CASO PADRÃO: salmoura única
for (size_t idx = 0; idx < salmouras.size(); ++idx) {
    const auto& salmoura = salmouras[idx];
    std::cout << "\n==== Salmoura " << (idx + 1) << " ==== \n";

    std::unordered_map<std::string, double> concentracoes = salmoura
        .getMapaIonsMols();
    double temperatura = salmoura.getTemperatura();

    std::vector<std::string> nomes;
    std::vector<double> qs;
    std::vector<double> ksps;

    for (const auto& sal : saisDisponiveis) {
        double Q = sal.calculateIonicProduct(concentracoes);
        double K = sal.KspCorrigido(temperatura);
        nomes.push_back(sal.getName());
        qs.push_back(Q);
        ksps.push_back(K);

        std::cout << "Sal: " << sal.getName()
            << " | Q = " << Q

```

```

        << " | Ksp_corrigido = " << K
        << " | T(K) = " << temperatura << std::endl;

    if (Q > K) {
        std::cout << " " << sal.getName() << " - PRECIPITA" <<
            std::endl;
    } else {
        std::cout << " " << sal.getName() << " - estavel" <<
            std::endl;
    }
}

plot.PlotQvsKsp(nomes, qs, ksps);

std::cout << "\nDigite o indice do sal para visualizar Q x Ksp(T
): ";
int escolha = 0;
std::cin >> escolha;
if (escolha >= 0 && escolha < static_cast<int>(saisDisponiveis.
size())) {
    const auto& salSelecionado = saisDisponiveis[escolha];
    double Qfixo = salSelecionado.calculateIonicProduct(
        concentracoes);
    std::vector<double> temps, kspsT;
    for (double T = temperatura - 20; T <= temperatura + 20; T
        += 1.0) {
        temps.push_back(T);
        kspsT.push_back(salSelecionado.KspCorrigido(T));
    }
    plot.PlotQvsKspVsTemperatura(salSelecionado.getName(), Qfixo
        , temps, kspsT);
}
}
}

```

Apresenta-se na listagem 7.13o arquivo com código da classe main.

Código 7.13: Arquivo de implementação da função main()

```

#include "CSimuladorPrecipitacao.h"

int main() {
    CSimuladorPrecipitacao simulador;
    simulador.executar();

    return 0;
}

```

Apresenta-se na listagem 7.14o arquivo com código da classe CPlotPrecipitacao.

Código 7.14: Arquivo de implementação da classe CPlotPrecipitacao

```

#include "CPlotPrecipitacao.h"
#include <fstream>
#include <sstream>
#include <cstdlib>
#include <iostream>

void CPlotPrecipitacao::PlotQvsKsp(const std::vector<std::string>& nomes
    ,
                                   const std::vector<double>& qs,
                                   const std::vector<double>& ksps) {
    std::ofstream file("q_vs_ksp.dat");
    if (!file.is_open()) return;
    for (size_t i = 0; i < nomes.size(); ++i) {
        file << i << "\t" << qs[i] << "\t" << ksps[i] << "\n";
    }
    file.close();

    std::ofstream gnu("plot_q_vs_ksp.gp");
    gnu << config.getTerminalSetting();
    gnu << "set logscale y\n";
    gnu << "set style data histograms\n";
    gnu << "set style fill solid border -1\n";
    gnu << "set boxwidth 0.9\n";
    gnu << "set xtics rotate by -45\n";
    gnu << "set xtics (";
    for (size_t i = 0; i < nomes.size(); ++i) {
        gnu << "\"" << nomes[i] << "\" " << i;
        if (i + 1 < nomes.size()) gnu << ", ";
    }
    gnu << ")\n";
    gnu << "plot 'q_vs_ksp.dat' using 2 title 'Q', '' using 3 title 'Ksp\n";
    gnu << "Corrigido'\n";
    gnu.close();

    std::system("gnuplot -persist plot_q_vs_ksp.gp");
}

void CPlotPrecipitacao::PlotQvsKspVsTemperatura(const std::string&
    nomeSal,
                                                    double qFixo,
                                                    const std::vector<double>&
                                                        temperaturas,
                                                    const std::vector<double>&
                                                        ksps) {
    std::ofstream file("ksp_vs_temp.dat");
    if (!file.is_open()) return;
    for (size_t i = 0; i < temperaturas.size(); ++i) {

```

```

        file << temperaturas[i] << "\t" << ksps[i] << "\t" << qFixo <<
            "\n";
    }
    file.close();

    std::ofstream gnu("plot_ksp_vs_temp.gp");
    gnu << config.getTerminalSetting();
    gnu << "set logscale y\n";
    gnu << "set xlabel 'Temperatura (K)'\n";
    gnu << "set ylabel 'Valor (mol^2/L^2)'\n";
    gnu << "plot 'ksp_vs_temp.dat' using 1:2 title 'Ksp Corrigido para "
        << nomeSal << " ' with lines, \\n";
    gnu << "      ' using 1:3 title 'Q fixo' with lines\n";
    gnu.close();

    std::system("gnuplot -persist plot_ksp_vs_temp.gp");
}

void CPlotPrecipitacao::MultiplotQvsKspVsTemperatura(
    const std::vector<std::string>& nomesSalmouras,
    const std::vector<std::vector<std::string>>& nomesSais,
    const std::vector<std::vector<double>>& todosQs,
    const std::vector<std::vector<double>>& todosKsps)
{
    std::vector<std::string> arquivosGerados;
    for (size_t i = 0; i < nomesSalmouras.size(); ++i) {
        std::string nomeArquivo = "salmoura_" + std::to_string(i + 1) +
            ".dat";
        arquivosGerados.push_back(nomeArquivo);
        std::ofstream dat(nomeArquivo);
        for (size_t j = 0; j < nomesSais[i].size(); ++j) {
            dat << nomesSais[i][j] << " " << todosQs[i][j] << " " <<
                todosKsps[i][j] << "\n";
        }
        dat.close();
    }

    std::ofstream gp("multiplot_barras.gp");
    gp << config.getTerminalSetting();
    gp << "set style data histograms\n";
    gp << "set style histogram cluster gap 1\n";
    gp << "set style fill solid border -1\n";
    gp << "set boxwidth 0.9\n";
    gp << "set logscale y\n";
    gp << "set yrange [1e-12:*]\n";
    gp << "set key outside\n";
    gp << "set multiplot layout 1," << nomesSalmouras.size() << " title
        'Q vs Ksp por Salmoura'\n";

```

```

    for (size_t i = 0; i < nomesSalmouras.size(); ++i) {
        gp << "set title '" << nomesSalmouras[i] << "'\n";
        gp << "set xtics rotate by -45\n";
        gp << "set xtics font ',8'\n";
        gp << "plot '" << arquivosGerados[i] << "' using 2:xtic(1) title
            'Q' lt rgb 'blue', \\n";
        gp << "      '' using 3 title 'Ksp' lt rgb 'red'\n";
    }

    gp << "unset multiplot\n";
    gp.close();

    system("gnuplot -persist multiplot_barras.gp");
}

```

Apresenta-se na listagem 7.15 o arquivo de implementação da classe CTabelaPropriedadesIons.

Código 7.15: Arquivo de implementação da classe CTabelaPropriedadesIons

```

#include "CTabelaPropriedadesIons.h"
#include <fstream>
#include <sstream>
#include <iostream>

CTabelaPropriedadesIons::CTabelaPropriedadesIons() {}

void CTabelaPropriedadesIons::carregarDeArquivo(const std::string&
    caminho) {
    std::ifstream arquivo("./dados/" + caminho);
    if (!arquivo.is_open()) return;

    std::string linha;
    while (std::getline(arquivo, linha)) {
        if (linha.empty() || linha[0] == '#') continue;
        std::istringstream iss(linha);
        std::string nome;
        int carga;
        if (iss >> nome >> carga) {
            m_ions[nome] = CIon(nome, carga);
        }
    }
}

void CTabelaPropriedadesIons::salvarParaArquivo(const std::string&
    caminho) const {
    std::ofstream arquivo("./dados/" + caminho);
    for (const auto& [nome, ion] : m_ions) {

```



```
        arquivo << nome << "\\t" << ion.getCarga() << "\\n";
    }
}

void CTabelaPropriedadesIons::adicionarIon(const CIon& ion) {
    m_ions[ion.getNome()] = ion;
}

CIon CTabelaPropriedadesIons::obterIon(const std::string& nome) const {
    auto it = m_ions.find(nome);
    if (it != m_ions.end()) {
        return it->second;
    }
    std::cerr << "Íon não encontrado: " << nome << "\\n";
    return CIon();
}

const std::unordered_map<std::string, CIon>& CTabelaPropriedadesIons::
getMapa() const {
    return m_ions;
}
```

**Nota:**

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.

# Capítulo 8

## Exemplos de Uso

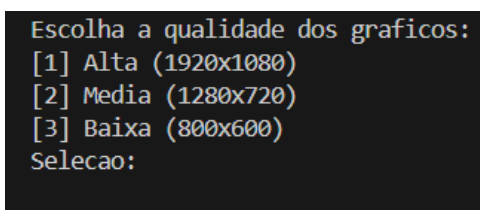
Todo projeto de engenharia passa por uma etapa de testes. Neste capítulo apresentamos alguns testes do software desenvolvido. Estes testes devem dar resposta aos diagramas de caso de uso inicialmente apresentados (diagramas de caso de uso geral e específicos).

### 8.1 Estrutura de Arquivos

Antes de iniciar, verifique se você tem os arquivos necessários na pasta ./dados/:

- Arquivos de íons: contêm os íons e suas concentrações para salmouras.
- Arquivos de sais: contêm os sais que podem precipitar, com seus dados de solubilidade.

Veja Figura 8.1.



```
Escolha a qualidade dos graficos:  
[1] Alta (1920x1080)  
[2] Media (1280x720)  
[3] Baixa (800x600)  
Selecao:
```

Figura 8.1: Início do Software

### 8.2 Selecionar o arquivo de íons disponíveis

Para que serve?

Esse arquivo contém os íons disponíveis na biblioteca, que podem ser utilizados na formação de sais (ex:  $\text{Ca}^{2+}$ ,  $\text{SO}_4^{2-}$ ).

O software listará os arquivos .txt da pasta ./dados/.

Ação do usuário: digite o número do arquivo de íons 4.

Veja Figura 8.2.

```

Selecione o arquivo de ions:
[1] salmoura_calcita.txt
[2] salmoura_exemplo.txt
[3] salmoura_todos_ions.txt
[4] tabela_ions.txt
[5] tabela_sais_tabs.txt
Digite o numero do arquivo desejado: 4

```

Figura 8.2: Documento Íon.

### 8.3 Selecionar o arquivo de sais disponíveis

Esse arquivo lista os sais que podem precipitar, com seus valores de Kps, entalpia de dissolução ( $\Delta P$ ) e íons constituintes.

Exemplo:

Nome	Ksp	DeltaP	Temperatura	Ions	Carga
Barita(BaSO <sub>4</sub> )	1.08e-10	-131000	298.15	Ba <sup>2+</sup>	+2

Veja Figura 1.3 8.3.

```

Selecione o arquivo de sais:
[1] salmoura_calcita.txt
[2] salmoura_exemplo.txt
[3] salmoura_todos_ions.txt
[4] tabela_ions.txt
[5] tabela_sais_tabs.txt
Digite o numero do arquivo desejado: 3

```

Figura 8.3: Documento Salmoura.

### 8.4 Informar quantas salmouras serão misturadas

Ação do usuário: digite a quantidade de salmouras (ex: 1,2 ).

Veja Figura 1.4. 8.4.

### 8.5 Inserir dados do cenário

Para cada salmoura deve-se inserir as condições termodinâmica, o usuário deve informar:

- Volume (L): volume total da salmoura.
- Temperatura (K): temperatura em Kelvin.
- Pressão (atm): pressão da salmoura.

Veja Figura 1.5. 8.5.

## 8.6 Cálculo da Precipitação

O programa calcula:

- As concentrações finais de cada íon.
- A temperatura média da mistura.
- O produto iônico  $Q$  e compara com o  $K_{sp}$  de cada sal.

Resultado:

- Se  $Q > K_{sp}$ : o sal precipita.
- Se  $Q < K_{sp}$ : o sal permanece dissolvido.

Veja Figura 1.6. 8.6.

## 8.7 Visualização gráfica (Q vs Ksp)

O programa plota:

- Um gráfico com o valor de  $Q$   $K_{sp}$  para cada sal.

Veja Figura 1.7. 8.7.

Ação do usuário:

- Digitar o índice do sal desejado para ver o gráfico com variação de temperatura.

Veja Figura 1.8. 8.8.

Ação do usuário:

Digitar o índice do sal desejado para ver o gráfico com variação de temperatura.

- Um gráfico adicional opcional ( $Q$  fixo vs  $K_{sp}$  em função da temperatura), para o sal selecionado.

Veja Figura 1.9. 8.9.

## 8.8 Estrutura de Arquivos

Antes de iniciar, verifique se você tem os arquivos necessários na pasta ./dados/:

- Arquivos de íons: contêm os íons e suas concentrações para salmouras.
- Arquivos de sais: contêm os sais que podem precipitar, com seus dados de solubilidade.

```

Selecione o arquivo de sais:
[1] salmoura_calcita.txt
[2] salmoura_exemplo.txt
[3] salmoura_todos_ions.txt
[4] tabela_ions.txt
[5] tabela_sais_tabs.txt
Digite o numero do arquivo desejado: 3
Quantas salmouras deseja adicionar? 1

```

Figura 8.4: Quantidade de salmoura adicionada.

```

--- Salmoura 1 ---
Volume (L): 4
Temperatura (K): 700
Pressao (atm): 1000

```

Figura 8.5: Condições termodinâmica

```

==== Resultados ====
Sal: Barita(BaSO4) | Q = 8e-05 | Ksp_corrigido = 4.32186e-25 | T(K) = 800
Barita(BaSO4) - PRECIPITA
Sal: Calcita(CaCO3) | Q = 0.0001 | Ksp_corrigido = 3.18321e-19 | T(K) = 800
Calcita(CaCO3) - PRECIPITA
Sal: Gipsita(CaSO4) | Q = 0.0001 | Ksp_corrigido = 1.33777e-12 | T(K) = 800
Gipsita(CaSO4) - PRECIPITA
Sal: Halita(NaCl) | Q = 0.000225 | Ksp_corrigido = 0.00211301 | T(K) = 800
Halita(NaCl) - estavel
Sal: Magnesita(MgCO3) | Q = 4e-05 | Ksp_corrigido = 4.30687e-18 | T(K) = 800
Magnesita(MgCO3) - PRECIPITA
Sal: Celestita(SrSO4) | Q = 5e-05 | Ksp_corrigido = 7.71941e-22 | T(K) = 800
Celestita(SrSO4) - PRECIPITA
Sal: Fluorita(CaF2) | Q = 9e-08 | Ksp_corrigido = 8.1146e-28 | T(K) = 800
Fluorita(CaF2) - PRECIPITA
Sal: Barrilha(Na2CO3) | Q = 2.25e-06 | Ksp_corrigido = 1.03952e-05 | T(K) = 800
Barrilha(Na2CO3) - estavel

```

Figura 8.6: Resultado Precipitação

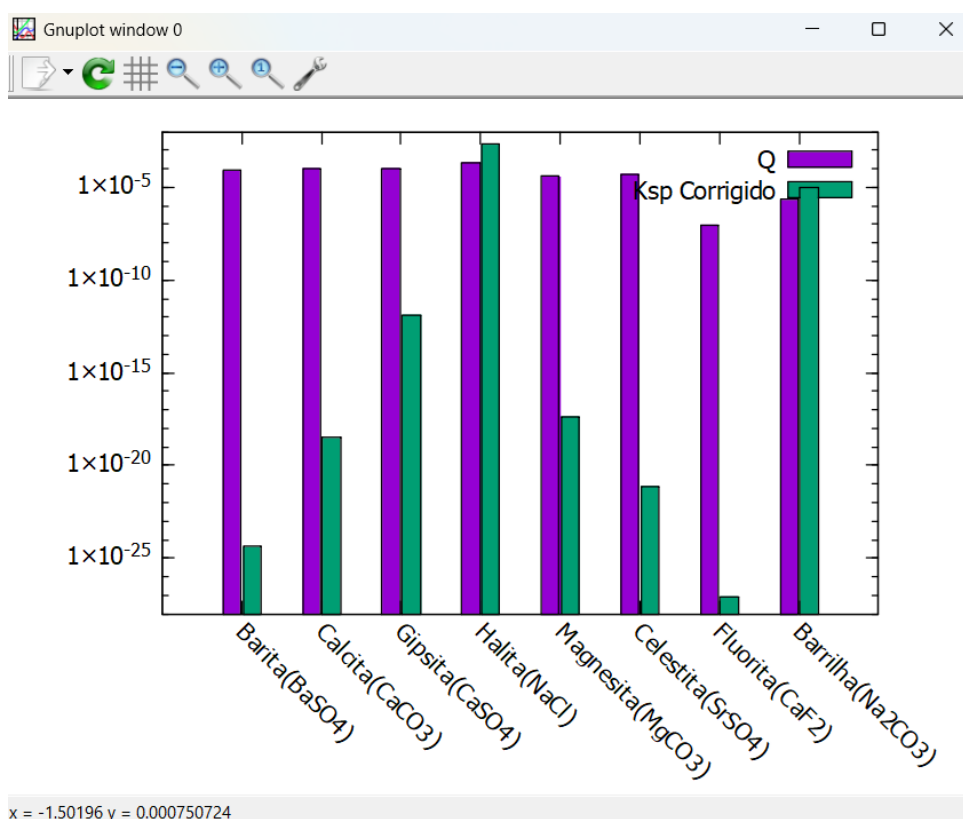


Figura 8.7: Resultado Precipitação Gráfico

```

==== Resultados ====
Sal: Barita(BaSO4) | Q = 8e-05 | Ksp_corrigido = 4.32186e-25 | T(K) = 800
Barita(BaSO4) - PRECIPITA
Sal: Calcita(CaCO3) | Q = 0.0001 | Ksp_corrigido = 3.18321e-19 | T(K) = 800
Calcita(CaCO3) - PRECIPITA
Sal: Gipsita(CaSO4) | Q = 0.0001 | Ksp_corrigido = 1.33777e-12 | T(K) = 800
Gipsita(CaSO4) - PRECIPITA
Sal: Halita(NaCl) | Q = 0.000225 | Ksp_corrigido = 0.00211301 | T(K) = 800
Halita(NaCl) - estavel
Sal: Magnesita(MgCO3) | Q = 4e-05 | Ksp_corrigido = 4.30687e-18 | T(K) = 800
Magnesita(MgCO3) - PRECIPITA
Sal: Celestita(SrSO4) | Q = 5e-05 | Ksp_corrigido = 7.71941e-22 | T(K) = 800
Celestita(SrSO4) - PRECIPITA
Sal: Fluorita(CaF2) | Q = 9e-08 | Ksp_corrigido = 8.1146e-28 | T(K) = 800
Fluorita(CaF2) - PRECIPITA
Sal: Barrilha(Na2CO3) | Q = 2.25e-06 | Ksp_corrigido = 1.03952e-05 | T(K) = 800
Barrilha(Na2CO3) - estavel

Digite o indice do sal para visualizar Q x Ksp(T):

```

Figura 8.8: Condições que o sal se precipita

Veja Figura 8.10.

```
Escolha a qualidade dos graficos:
[1] Alta (1920x1080)
[2] Media (1280x720)
[3] Baixa (800x600)
Selecão:
```

Figura 8.10: Início do Software

## 8.9 Selecionar o arquivo de íons disponíveis

Para que serve?

Esse arquivo contém os íons disponíveis na biblioteca, que podem ser utilizados na formação de sais (ex:  $\text{Ca}^{2+}$ ,  $\text{SO}_4^{2-}$ ).

O software listará os arquivos .txt da pasta ./dados/.

Ação do usuário: digite o número do arquivo de íons 4.

Veja Figura 8.11.

```
Selecione o arquivo de ions:
[1] salmoura_calcita.txt
[2] salmoura_exemplo.txt
[3] salmoura_todos_ions.txt
[4] tabela_ions.txt
[5] tabela_sais_tabs.txt
Digite o numero do arquivo desejado: 4
```

Figura 8.11: Documento Íon.

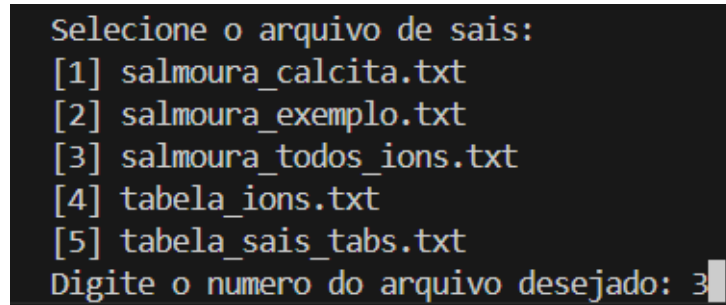
## 8.10 Selecionar o arquivo de sais disponíveis

Esse arquivo lista os sais que podem precipitar, com seus valores de Kps, entalpia de dissolução (deltaP) e íons constituintes.

Exemplo:

Nome	Ksp	DeltaP	Temperatura	Ions	Carga
Barita( $\text{BaSO}_4$ )	1.08e-10	-131000	298.15	$\text{Ba}^{2+}$	+2

Veja Figura 1.3 8.12.



```
Selecione o arquivo de sais:
[1] salmoura_calcita.txt
[2] salmoura_exemplo.txt
[3] salmoura_todos_ions.txt
[4] tabela_ions.txt
[5] tabela_sais_tabs.txt
Digite o numero do arquivo desejado: 3
```

Figura 8.12: Documento Salmoura.

## 8.11 Informar quantas salmouras serão misturadas

Ação do usuário: digite a quantidade de salmouras (ex: 1,2 ).

Veja Figura 1.4. 8.13.

## 8.12 Inserir dados do cenário

Para cada salmoura deve-se inserir as condições termodinâmica, o usuário deve informar:

- Volume (L): volume total da salmoura.
- Temperatura (K): temperatura em Kelvin.
- Pressão (atm): pressão da salmoura.

Veja Figura 1.5. 8.14.

## 8.13 Calculo da Precipitação

O programa calcula:

- As concentrações finais de cada íon.
- A temperatura média da mistura.
- O produto iônico  $Q$  e compara com o  $K_{sp}$  de cada sal.

Resultado:

- Se  $Q > K_{sp}$ : o sal precipita.
- Se  $Q < K_{sp}$ : o sal permanece dissolvido.

Veja Figura 1.6. 8.15.



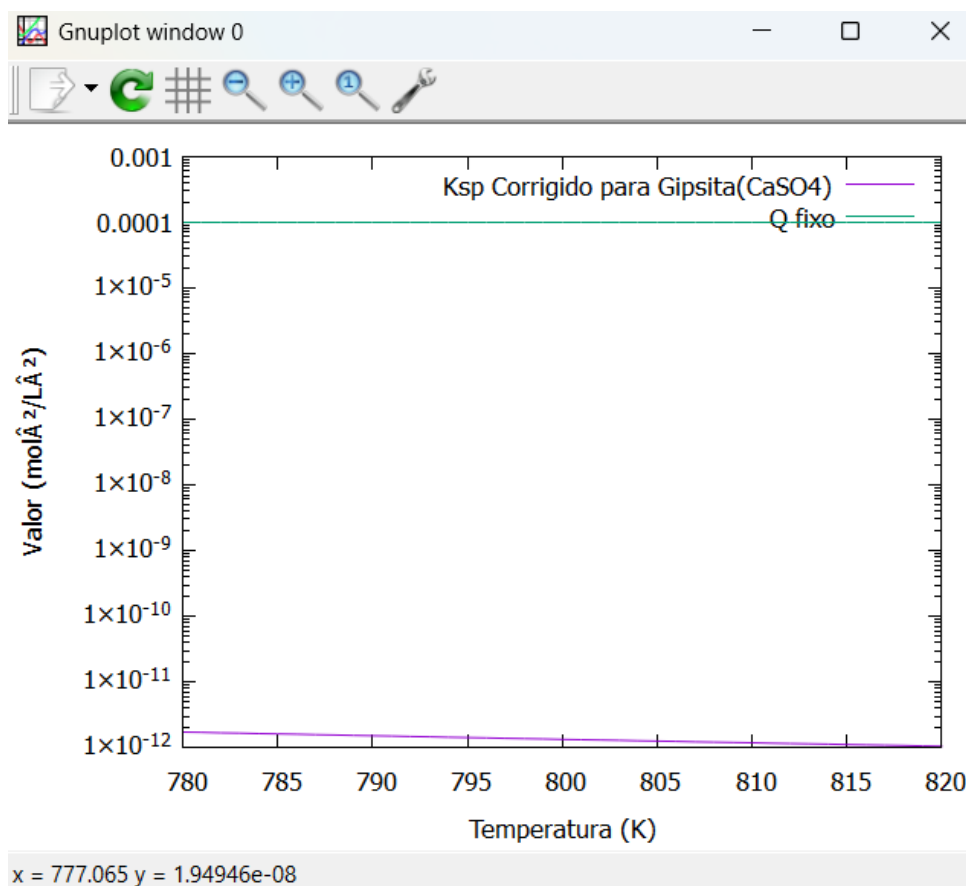


Figura 8.9: Sal específico precipitado

```

Selecione o arquivo de sais:
[1] salmoura_calcita.txt
[2] salmoura_exemplo.txt
[3] salmoura_todos_ions.txt
[4] tabela_ions.txt
[5] tabela_sais_tabs.txt
Digite o numero do arquivo desejado: 3
Quantas salmouras deseja adicionar? 1

```

Figura 8.13: Quantidade de salmoura adicionada.

```

--- Salmoura 1 ---
Volume (L): 4
Temperatura (K): 700
Pressao (atm): 1000

```

Figura 8.14: Condições termodinâmica

## 8.14 Visualização gráfica (Q vs Ksp)

O programa plota:

- Um gráfico com o valor de  $Q$   $K_{sp}$  para cada sal.

Veja Figura 1.7. 8.16.

Ação do usuário:

- Digitar o índice do sal desejado para ver o gráfico com variação de temperatura.

Veja Figura 1.8. 8.17.

Ação do usuário:

Digitar o índice do sal desejado para ver o gráfico com variação de temperatura.

- Um gráfico adicional opcional ( $Q$  fixo vs  $K_{sp}$  em função da temperatura), para o sal selecionado.

Veja Figura 1.9. 8.18.

```

==== Resultados ====
Sal: Barita(BaSO4) | Q = 8e-05 | Ksp_corrigido = 4.32186e-25 | T(K) = 800
      Barita(BaSO4) - PRECIPITA
Sal: Calcita(CaCO3) | Q = 0.0001 | Ksp_corrigido = 3.18321e-19 | T(K) = 800
      Calcita(CaCO3) - PRECIPITA
Sal: Gipsita(CaSO4) | Q = 0.0001 | Ksp_corrigido = 1.33777e-12 | T(K) = 800
      Gipsita(CaSO4) - PRECIPITA
Sal: Halita(NaCl) | Q = 0.000225 | Ksp_corrigido = 0.00211301 | T(K) = 800
      Halita(NaCl) - estavel
Sal: Magnesita(MgCO3) | Q = 4e-05 | Ksp_corrigido = 4.30687e-18 | T(K) = 800
      Magnesita(MgCO3) - PRECIPITA
Sal: Celestita(SrSO4) | Q = 5e-05 | Ksp_corrigido = 7.71941e-22 | T(K) = 800
      Celestita(SrSO4) - PRECIPITA
Sal: Fluorita(CaF2) | Q = 9e-08 | Ksp_corrigido = 8.1146e-28 | T(K) = 800
      Fluorita(CaF2) - PRECIPITA
Sal: Barrilha(Na2CO3) | Q = 2.25e-06 | Ksp_corrigido = 1.03952e-05 | T(K) = 800
      Barrilha(Na2CO3) - estavel

```

Figura 8.15: Resultado Precipitação

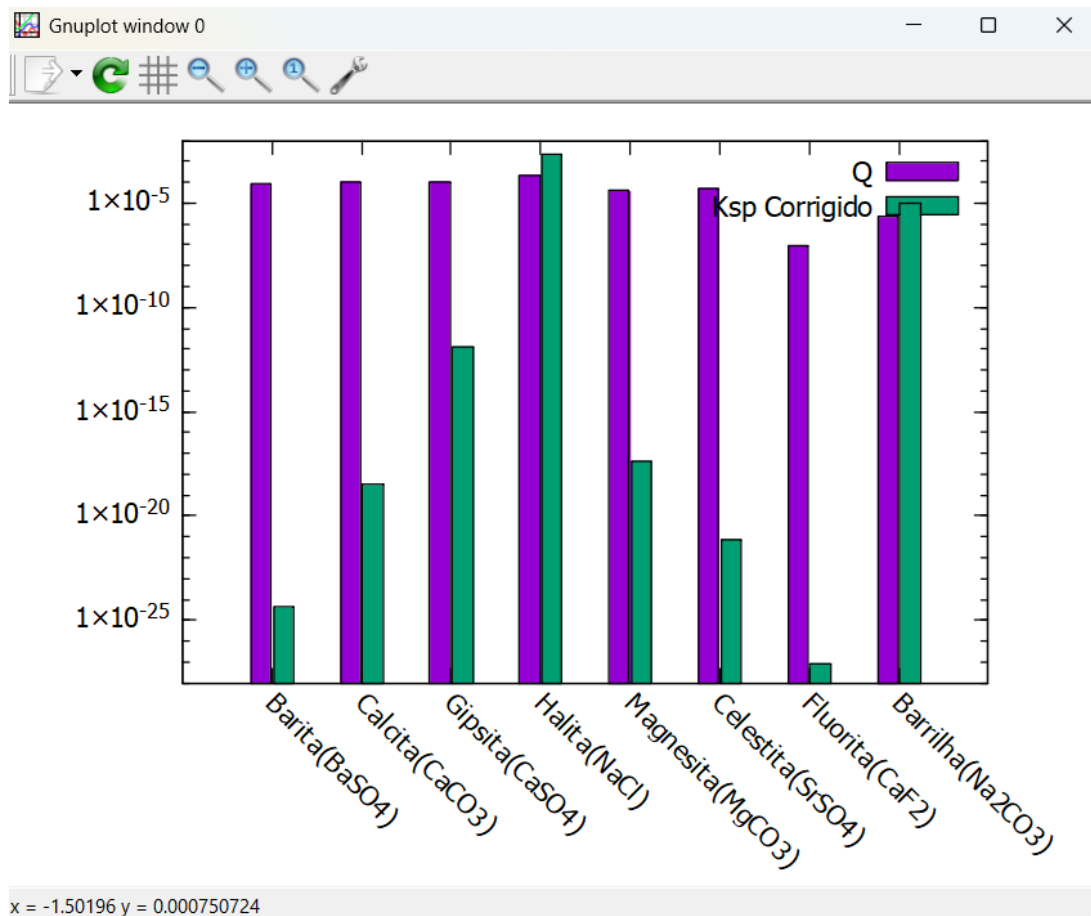


Figura 8.16: Resultado Precipitação Gráfico

```

===== Resultados =====
Sal: Barita(BaSO4) | Q = 8e-05 | Ksp_corrigido = 4.32186e-25 | T(K) = 800
      Barita(BaSO4) - PRECIPITA
Sal: Calcita(CaCO3) | Q = 0.0001 | Ksp_corrigido = 3.18321e-19 | T(K) = 800
      Calcita(CaCO3) - PRECIPITA
Sal: Gipsita(CaSO4) | Q = 0.0001 | Ksp_corrigido = 1.33777e-12 | T(K) = 800
      Gipsita(CaSO4) - PRECIPITA
Sal: Halita(NaCl) | Q = 0.000225 | Ksp_corrigido = 0.00211301 | T(K) = 800
      Halita(NaCl) - estavel
Sal: Magnesita(MgCO3) | Q = 4e-05 | Ksp_corrigido = 4.30687e-18 | T(K) = 800
      Magnesita(MgCO3) - PRECIPITA
Sal: Celestita(SrSO4) | Q = 5e-05 | Ksp_corrigido = 7.71941e-22 | T(K) = 800
      Celestita(SrSO4) - PRECIPITA
Sal: Fluorita(CaF2) | Q = 9e-08 | Ksp_corrigido = 8.1146e-28 | T(K) = 800
      Fluorita(CaF2) - PRECIPITA
Sal: Barrilha(Na2CO3) | Q = 2.25e-06 | Ksp_corrigido = 1.03952e-05 | T(K) = 800
      Barrilha(Na2CO3) - estavel

Digite o indice do sal para visualizar Q x Ksp(T):

```

Figura 8.17: Condições que o sal se precipita

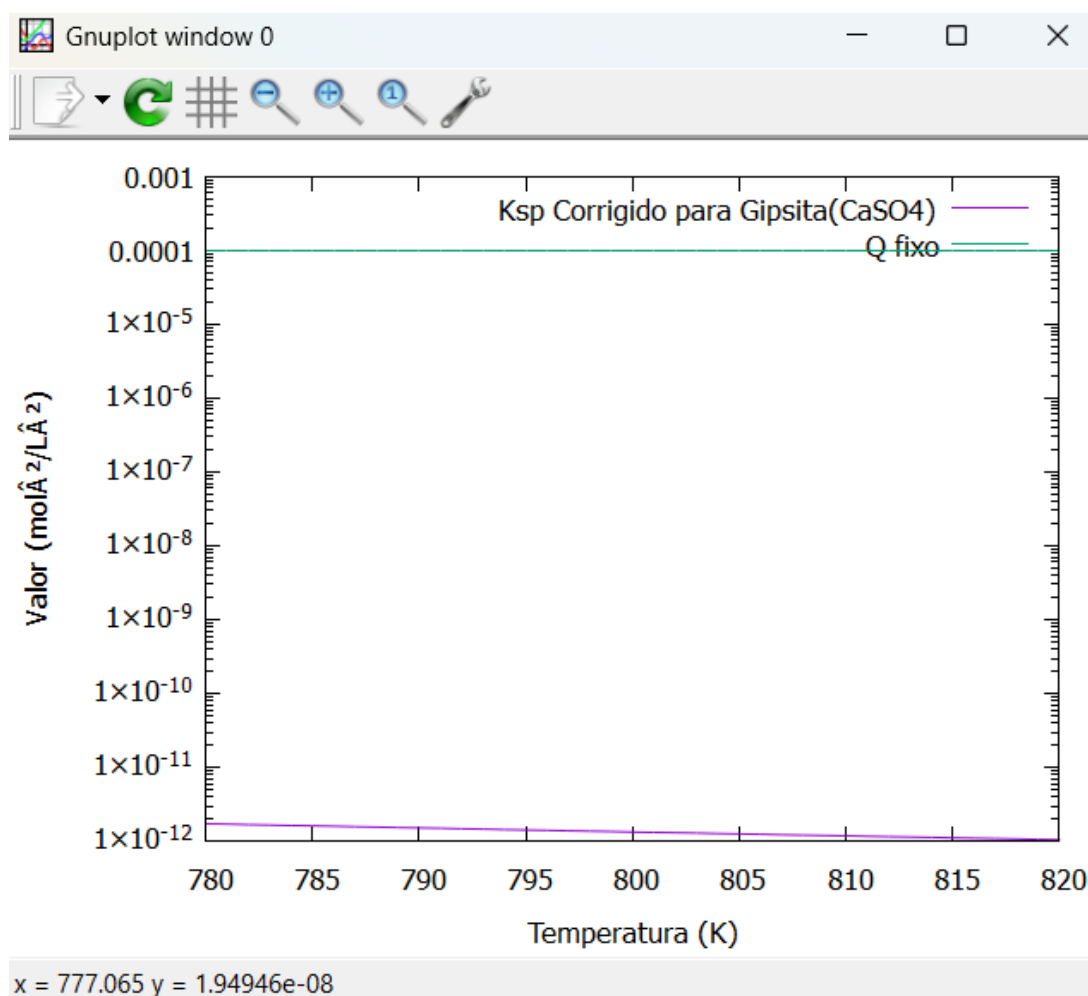


Figura 8.18: Sal específico precipitado

## Capítulo 9

# Documentação para o Desenvolvedor

Todo projeto de engenharia precisa ser bem documentado. Neste sentido, apresenta-se neste capítulo documentações extras para o desenvolvedor. Ou seja, instruções para pessoas que venham a dar continuidade a este projeto de engenharia.

**Nota:** O manual do usuário é apresentado em um documento separado. Veja diretório "doc/ManualDoUsuario".

### 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`.
- Biblioteca CGnuplot; os arquivos para acesso a biblioteca CGnuplot devem estar no diretório com os códigos do software;
- O software `gnuplot`, disponível no endereço <http://www.gnuplot.info/>, deve estar instalado. É possível que haja necessidade de setar o caminho para execução do `gnuplot`.
- .
- .

### 9.2 Como gerar a documentação usando doxygen

A documentação do código do software deve ser feita usando o padrão JAVADOC, conforme apresentada no Capítulo - Documentação, do livro texto da disciplina. Depois de documentar o código, use o software `doxygen` para gerar a documentação do desenvolvedor no formato html. O software `doxygen` lê os arquivos com os códigos (\*.h e \*.cpp) e gera uma documentação muito útil e de fácil navegação no formato html.

- Veja informações sobre uso do formato JAVADOC em:
  - <http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html>
- Veja informações sobre o software `doxygen` em
  - <http://www.stack.nl/~dimitri/doxygen/>

Passos para gerar a documentação usando o `doxygen`.

- Documente o código usando o formato JAVADOC. Um bom exemplo de código documentado é apresentado nos arquivos da biblioteca CGnuplot, abra os arquivos `CGnuplot.h` e `CGnuplot.cpp` no editor de texto e veja como o código foi documentado.
- Abra um terminal.
- Vá para o diretório onde está o código.

```
cd /caminho/para/seu/codigo
```

- Peça para o `doxygen` gerar o arquivo de definições (arquivo que diz para o `doxygen` como deve ser a documentação).

```
doxygen -g
```

- Peça para o `doxygen` gerar a documentação.

```
doxygen
```

- Verifique a documentação gerada abrindo o arquivo `html/index.html`.

```
firefox html/index.html
```

ou

```
chrome html/index.html
```

Apresenta-se a seguir algumas imagens com as telas das saídas geradas pelo software `doxygen`.

#### **Nota:**

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.



# Referências Bibliográficas

- [Blaha and Rumbaugh, 2006] Blaha, M. and Rumbaugh, J. (2006). *Modelagem e Projetos Baseados em Objetos com UML 2*. Campus, Rio de Janeiro. 18
- [Rumbaugh et al., 1994] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1994). *Modelagem e Projetos Baseados em Objetos*. Edit. Campus, Rio de Janeiro. 18