UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO
CENTRO DE CIÊNCIA E TECNOLOGIA

PRÉ-PROJETO DE ENGENHARIA
DESENVOLVIMENTO DO SOFTWARE

"PetroPlanner"

Github: SimuladorDePlanejamentoDasAtividadesDoEstudanteEngenharia

DISCIPLINA LEP - LEP01348 : Introdução ao Projeto de Engenharia

Setor de Modelagem Matemática e Computacional

Versão 3:

AUTORES
Abigail Ribeiro Pinto Silva Guimarães
Maria Luiza Cornelio Ramos
Prof. André Duarte Bueno

MACAÉ - RJ
Maio - 2025

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Concepção

O presente projeto de engenharia tem como objetivo o desenvolvimento do PetroPlanner, um sistema computacional voltado para o planejamento acadêmico de estudantes de Engenharia de Petróleo. A ideia surgiu da necessidade recorrente de organizar a trajetória universitária, visualizar disciplinas cursadas e pendentes, acompanhar faltas, simular futuros períodos letivos e organizar horários semanais de forma visual, prática e intuitiva.

Durante a graduação, é comum que os alunos utilizem diversos métodos paralelos — como planilhas, anotações em papel ou ferramentas genéricas — para acompanhar seu progresso acadêmico. Tais métodos, embora úteis, são muitas vezes descentralizados, manuais e sujeitos a erros. O PetroPlanner busca resolver esses problemas através de uma aplicação personalizada, pensada especificamente para a grade curricular da Universidade Estadual do Norte Fluminense Darcy Ribeiro (UENF), mais precisamente da graduação oferecida no Laboratório de Engenharia e Exploração de Petróleo (LENEP).

A concepção do sistema levou em consideração os seguintes aspectos:

- Centralização das informações do aluno (nome, matrícula, curso, CRA, disciplinas);
- Classificação automática das disciplinas em andamento, aprovadas e pendentes, com base em um arquivo padrão .txt;
- Simulação de planejamento de semestres futuros, respeitando os pré-requisitos do curso;
- Visualização gráfica do quadro de horários, com possibilidade de adicionar atividades extracurriculares;
- Acompanhamento contínuo de faltas e notas, com alertas visuais sobre risco de reprovação;
- Interface amigável e acessível, construída com Qt, que visa facilitar o uso mesmo por alunos sem familiaridade com tecnologia.

A especificação do sistema inclui funcionalidades integradas que dialogam com arquivos externos e respondem a interações do usuário em tempo real. A modelagem foi feita com o auxílio de diagramas UML (casos de uso, classes, pacotes, sequência, estados, atividades), e a implementação utilizou C++ com Qt, respeitando os princípios da programação orientada a objetos.

O projeto é modular, expansível e pode futuramente ser adaptado para outras engenharias ou instituições, bastando alterar a estrutura curricular cadastrada.

1.1 Metodologia

A Figura 1.1 apresenta a metodologia a ser utilizada no desenvolvimento do sistema.

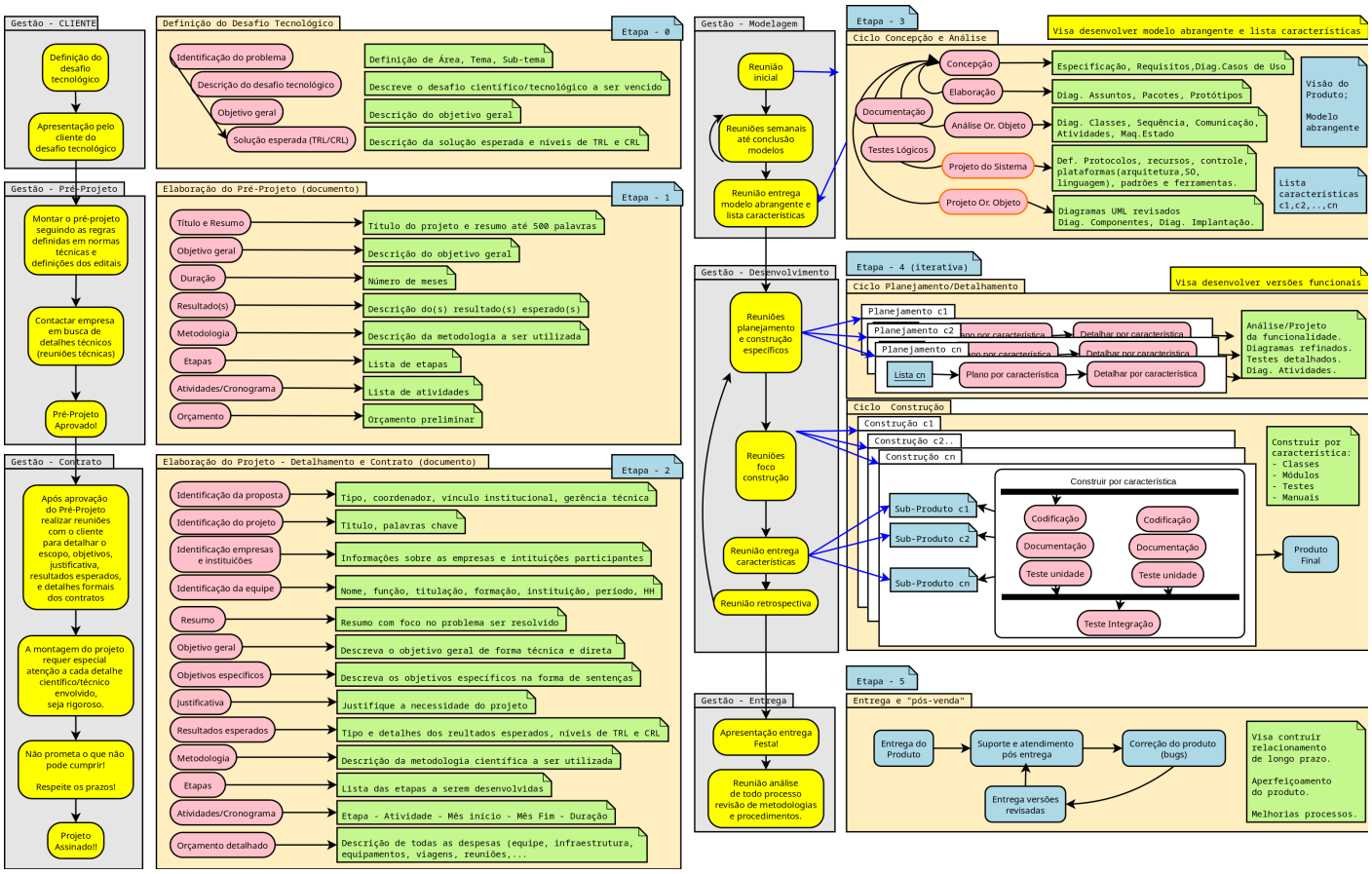

Figura 1.1: Metodologia utilizada no desenvolvimento do sistema

1.2 Nome do Sistema/Produto

Nome	PetroPlanner
Componentes principais	Menu lateral com ícones, telas de cadastro, campos de texto, caixas de seleção, tabelas interativas, botões de ação (Salvar, Editar, Cancelar).
Missão	Auxiliar estudantes de Engenharia de Petróleo no planejamento acadêmico, permitindo visualizar, organizar e simular sua trajetória universitária.

1.3 Especificação

O software PetroPlanner é uma ferramenta interativa desenvolvida para auxiliar estudantes de Engenharia de Petróleo no planejamento e acompanhamento de sua trajetória acadêmica, com base na

matriz curricular da UENF/LENEP.

O sistema permite ao aluno visualizar a grade curricular completa, classificada em disciplinas já cursadas, em andamento e a serem cursadas. A leitura dos dados é realizada a partir de um arquivo externo de texto, que contém as principais informações acadêmicas do estudante, como nome, matrícula, curso, CRA, período atual e histórico das disciplinas.

A interface permite que o aluno edite as disciplinas que está cursando, organize seus horários semanalmente (disciplinas e atividades extras) por meio de um quadro de horários interativo, e visualize alertas referentes ao número de faltas restantes em cada disciplina, com base na carga horária semanal.

O sistema também inclui um simulador de planejamento de semestres, no qual o estudante pode montar cenários futuros respeitando os pré-requisitos entre disciplinas e agrupando visualmente os componentes curriculares por semestre.

Além disso, é possível acompanhar individualmente cada disciplina, com detalhes como dias e horários de aula, trabalhos, provas e nível de dificuldade estimado. Toda a navegação é realizada por meio de uma interface gráfica simples e intuitiva, desenvolvida com a biblioteca Qt em C++.

As principais funcionalidades são distribuídas entre os seguintes módulos:

- Tela Inicial: ponto de entrada e resumo geral do aluno.
- Grade Curricular Completa: mostra todas as disciplinas do curso com status visual.
- Editor de Disciplinas em Curso: permite o aluno selecionar manualmente o que está cursando.
- Quadro de Horários: ferramenta de organização semanal com disciplinas e atividades.
- Acompanhamento de Disciplinas: visualização de detalhes e controle de faltas.
- Simulação de Planejamento: ambiente para montar semestres futuros conforme pré-requisitos.

O comportamento esperado do sistema é manter o histórico do aluno consistente com as alterações feitas na interface, garantindo que todas as informações estejam devidamente salvas no arquivo de entrada/saída (InformacoesAluno.txt), permitindo sua reutilização em futuras execuções.

1.4 Requisitos

1.4.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais

RF-01	O sistema deve ler os dados do aluno a partir de um arquivo texto (InformacoesAluno.txt) contendo nome, matrícula, curso, período, CRA e histórico de disciplinas.

RF-02	O sistema deve classificar as disciplinas em três categorias: Aprovadas, Em Curso e Não Cursadas.

RF-03	O sistema deve permitir a visualização de uma grade curricular completa, destacando as disciplinas já cursadas ou em andamento.

RF-04	O sistema deve possibilitar o planejamento de semestres futuros, respeitando os pré-requisitos entre disciplinas.

RF-05	O sistema deve exibir um quadro de horários interativo, onde o aluno possa organizar suas disciplinas em andamento e atividades extracurriculares.

RF-06	O sistema deve permitir o acompanhamento individual de disciplinas, exibindo faltas, dias e horários, trabalhos e provas.

RF-07	O sistema deve alertar visualmente sobre o limite de faltas permitido em cada disciplina.

RF-08	O sistema deve permitir que o aluno edite as disciplinas em curso, marcando quais está cursando no momento.

1.4.2 Requisitos não funcionais

RNF-01	O sistema deve possuir uma interface gráfica intuitiva, amigável e acessível a usuários com conhecimentos básicos de informática, utilizando botões autoexplicativos, cores diferenciadas e organização clara por seções.

RNF-02	O sistema deve ser compatível com sistemas operacionais que suportem aplicações Qt5 ou Qt6, como Windows e distribuições Linux.

RNF-03	Todas as informações acadêmicas do aluno (disciplinas cursadas, em andamento, atividades, faltas, horários) devem ser armazenadas e lidas a partir de um arquivo externo (InformacoesAluno.txt), garantindo persistência entre diferentes execuções do programa.

RNF-04	O sistema deve realizar a leitura do arquivo de entrada e o carregamento da interface principal em menos de 2 segundos, mesmo com uma grade curricular extensa.

RNF-05	A estrutura do código-fonte deve estar organizada em módulos e classes (ex: CAluno, CDisciplinas, TelaInicial), facilitando a manutenção, testes e futura expansão do sistema.

RNF-06	Os dados devem ser apresentados de forma visualmente compreensível, com destaque para disciplinas em andamento, alertas de faltas e agrupamento por período letivo ou por tipo de atividade.

RNF-07	O sistema deve garantir que informações críticas como pré-requisitos, horários e número de faltas não sejam perdidas ou corrompidas durante o uso, mesmo após alterações múltiplas feitas pelo usuário.

RNF-08	A interface deve se ajustar a diferentes tamanhos de tela, permitindo boa experiência de uso em monitores com diferentes resoluções.

1.5 Casos de Uso

A Tabela 1.1 mostra a descrição de um caso de uso.

Tabela 1.1: Caso de uso 1

Nome do caso de uso:	Acessar Módulo de Grade Curricular
Resumo/descrição:	Permite ao aluno visualizar sua grade curricular, disciplinas em andamento, e histórico de aprovações.
Etapas:	1. O aluno seleciona "Grade Curricular" no menu. 2. O sistema carrega as disciplinas (em andamento, aprovadas, pendentes). 3. O aluno filtra por período ou ciclo (opcional). 4. O sistema exibe a grade com cores indicativas (ex: vermelho para disciplinas com pré-requisitos não atendidos).
Cenários alternativos:	1. Dados não carregados: Sistema exibe mensagem de erro e sugere recarregar. 2. Disciplinas conflitantes: Sistema alerta sobre horários sobrepostos.

1.5.1 Diagrama de Casos de Uso: Simular Notas das Disciplinas em Andamento

Este diagrama ilustra como o Aluno interage com o Módulo de Acompanhamento de Disciplinas do sistema PetroPlanner. Ele detalha as funcionalidades que permitem ao aluno visualizar suas disciplinas em andamento, acessar os detalhes de notas e faltas de uma disciplina específica, simular notas para projeções, inserir notas reais quando recebidas, acompanhar a média final da disciplina, registrar e atualizar faltas, e finalmente, finalizar uma disciplina para que ela seja removida da lista de disciplinas em andamento.

1.5.2 Diagrama de Casos de Uso: Realizar Planejamento Semestral

Este diagrama descreve como o Aluno utiliza o Módulo de Simulação de Planejamento Semestral do sistema PetroPlanner. Ele mostra como o aluno acessa este módulo, visualiza as disciplinas que já cursou ou está cursando, explora as disciplinas disponíveis para os próximos semestres (considerando pré-requisitos), seleciona e marca as disciplinas que deseja cursar, avança a simulação semestre a semestre, e, por fim, salva o planejamento semestral simulado.

Diagrama de Casos de Uso: Montar Quadro de Horários

Este diagrama mostra todo o processo de como o aluno pode visualizar, planejar, e personalizar seu quadro de horários no semestre atual, utilizando tanto a interface gráfica quanto, opcionalmente, um arquivo de texto para a entrada de dados.

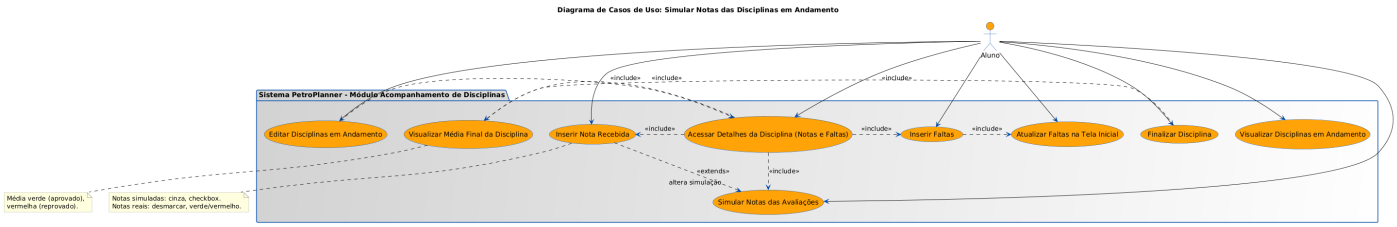

Figura 1.2: Diagrama de Casos de Uso: Simular Notas das Disciplinas em Andamento

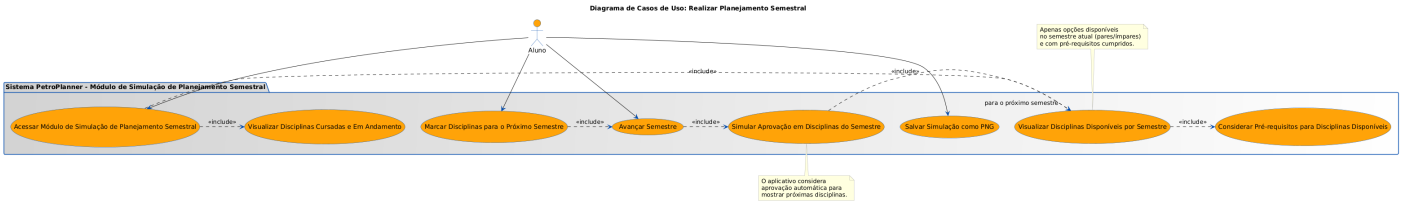

Figura 1.3: Diagrama de Casos de Uso: Realizar Planejamento Semestral

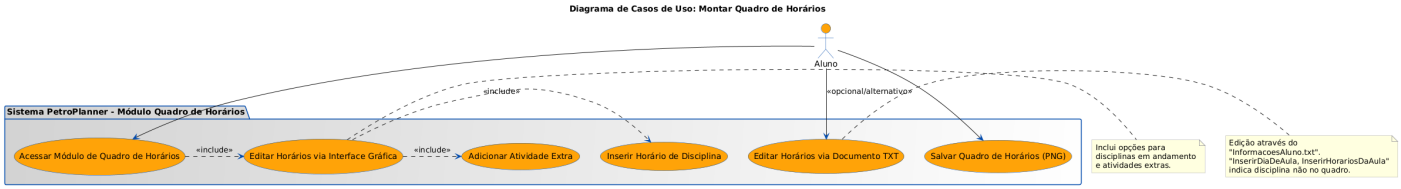

Figura 1.4: Diagrama de Casos de Uso: Montar Quadro de Horários

Capítulo 2

Elaboração

2.1 Análise de domínio

O domínio do PetroPlanner está inserido no contexto da formação acadêmica em Engenharia de Petróleo, mais especificamente no planejamento de disciplinas curriculares ao longo dos períodos letivos, com base em pré-requisitos, co-requisitos, carga horária e áreas de conhecimento (eixos temáticos).

Definição do domínio:

O sistema atua como um planejador inteligente e visual, que auxilia estudantes na organização de seu percurso acadêmico dentro de cursos de Engenharia de Petróleo, respeitando as regras de matrícula e permitindo simulações de diferentes trajetórias.

Disciplinas relacionadas:

- Introdução a engenharia de petróleo
- Fundamentos da computação
- Calculo numérico
- Introdução ao projeto de engenharia
- Programação orientada a objeto em com C++

Questões espaciais/temporais:

- O sistema considera o semestre letivo (tempo), permitindo visualização por períodos.
- Não envolve local físico diretamente, mas o contexto é aplicado a instituições de ensino superior (ambiente acadêmico).

Inspirações e sistemas equivalentes:

- Sistemas de matrícula universitários (como SIGA, SIGAA, Moodle acadêmico);
- Softwares de simulação e planejamento de carreira;
- Ferramentas como o Trello (para organização visual) e planners curriculares de universidades.

2.2 Formulação teórica

O desenvolvimento do sistema PetroPlanner é fundamentado em conceitos clássicos da engenharia de software, da programação orientada a objetos (POO), e da modelagem de sistemas com UML (Unified Modeling Language). Estes elementos foram essenciais para garantir organização, modularidade, manutenibilidade e facilidade de uso da aplicação proposta.

Programação Orientada a Objetos (POO)

A estrutura do PetroPlanner foi desenvolvida utilizando a linguagem C++, aplicando os princípios da Programação Orientada a Objetos. Entre os conceitos utilizados, destacam-se:

- Encapsulamento: cada classe encapsula seus próprios dados e comportamentos, como as classes CAluno, CDisciplinas, QuadroDeHorarios, entre outras.
- Abstração: as classes representam abstrações reais do contexto acadêmico, como uma disciplina ou um aluno.
- Modularidade: cada módulo da aplicação é independente e interage por meio de interfaces bem definidas, facilitando a manutenção.
- Associação e Composição: A classe CAluno compõe vetores de objetos do tipo CDisciplinas, evidenciando a relação de composição entre aluno e suas disciplinas.

Modelagem UML

Para a modelagem do sistema, foram elaborados diversos diagramas UML, que permitiram a visualização estruturada dos componentes do projeto. Dentre os diagramas utilizados:

- Diagrama de Casos de Uso: identifica os principais atores e funcionalidades oferecidas pelo sistema.
- Diagrama de Classes: define a estrutura do sistema, seus atributos e métodos, bem como os relacionamentos entre classes.
- Diagramas de Sequência e Comunicação: ilustram o fluxo de mensagens entre os componentes do sistema ao longo da execução.
- Diagrama de Atividades: representa o fluxo de trabalho do usuário na interação com o sistema.
- Diagrama de Estado: mostra os estados possíveis das disciplinas (em curso, aprovadas, não cursadas).
- Diagramas de Componentes e Implantação: descrevem a organização física e lógica dos arquivos e estruturas que compõem o sistema.

Esses diagramas foram fundamentais para guiar a implementação e validar os requisitos levantados na fase de concepção.

Engenharia de Software

Durante a elaboração do sistema, foram considerados os princípios do ciclo de vida de software baseado em modelagem, implementação, validação e evolução. A modelagem dos dados, aliada à definição

clara de requisitos funcionais e não funcionais, permitiu o desenvolvimento de uma aplicação funcional, coerente com as necessidades do público-alvo.

Além disso, boas práticas de versionamento (como uso de Git), modularização, e documentação do código foram adotadas para garantir maior qualidade e escalabilidade ao projeto.

2.3 Identificação de pacotes – assuntos

- **Pacote: DadosAluno**

Descrição:

Este pacote é responsável por representar e gerenciar as informações acadêmicas do aluno, como nome, matrícula, curso, período, CRA e listas de disciplinas. Contém a classe CAluno, que encapsula os dados pessoais e acadêmicos do estudante.

Relação com outros pacotes:

É utilizado por quase todos os outros pacotes, pois fornece os dados base para construção de funcionalidades, visualizações e simulações. Atua como núcleo da aplicação.

- **Pacote: Disciplinas**

Descrição:

Este pacote representa a estrutura das disciplinas do curso, com todas as suas características (nome, carga horária, créditos, pré-requisitos, co-requisitos, etc.). Contém a classe CDisciplinas, além de funções auxiliares como getDisciplinasCurso().

Relação com outros pacotes:

Está associado diretamente ao pacote DadosAluno, pois suas instâncias compõem os vetores de disciplinas aprovadas, em curso e não cursadas. Também é usado no pacote Planejamento.

- **Pacote: GradeCurricular**

Descrição:

Este pacote é responsável por exibir visualmente toda a grade curricular do curso. Contém o módulo ModuloGradeCompleta, que apresenta ao usuário a disposição das disciplinas por período, com destaque para as que já foram cursadas, estão em andamento ou ainda não foram feitas.

Relação com outros pacotes:

Depende dos pacotes DadosAluno e Disciplinas para carregar corretamente as informações da grade e marcá-las com o status atual do aluno.

- **Pacote: Planejamento**

Descrição:

Este pacote reúne as funcionalidades relacionadas à projeção dos próximos períodos, com base nas disciplinas não cursadas e na verificação de pré-requisitos. Está representado principalmente pela classe SimulacaoPlanejamentoSEM.

Relação com outros pacotes:

Utiliza os dados de DadosAluno e Disciplinas para determinar quais matérias podem ser planejadas nos próximos semestres. Também se integra ao pacote Interface.

- **Pacote: QuadroDeHorarios**

Descrição:

Responsável por permitir que o aluno organize visualmente sua semana com as disciplinas em curso e atividades extras. Contém a classe QuadroDeHorarios, que monta e gerencia a tabela de horários.

Relação com outros pacotes:

Se baseia nos dados contidos em DadosAluno e nas propriedades de CDisciplinas. Interage também com Interface, pois depende de interações do usuário.

- **Pacote: Interface**

Descrição:

Reúne todas as janelas e interações visuais da aplicação, como TelaInicial, EditarDisciplinasEmCurso, botões, formulários e janelas auxiliares. Responsável pela experiência do usuário.

Relação com outros pacotes:

Interage com todos os outros pacotes, pois serve como ponte entre o usuário e a lógica da aplicação. É onde os dados são mostrados e manipulados de forma interativa.

2.4 Diagrama de pacotes – assuntos

Este diagrama representa a organização modular do sistema PetroPlanner, separando os componentes principais em pacotes lógicos. Os pacotes incluem a interface gráfica, as classes de modelo (como CAluno e CDisciplinas), os controladores responsáveis pela lógica de negócio e o módulo de persistência de dados. Essa separação facilita a manutenção, a reutilização de código e o entendimento da estrutura geral do sistema.

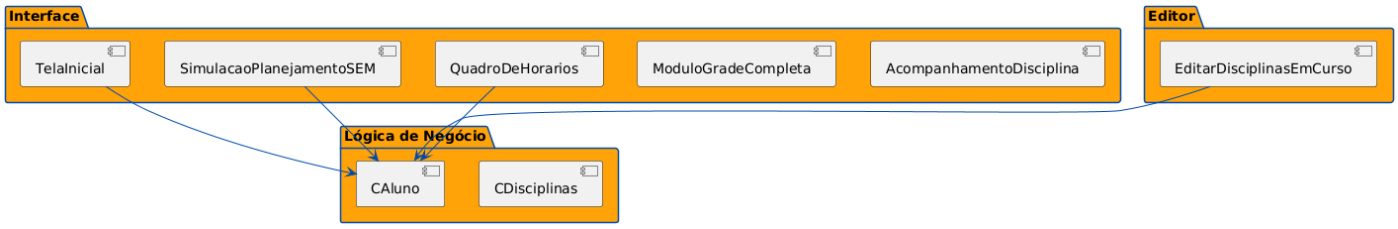

Figura 2.1: Diagrama de Pacotes

Capítulo 3

AOO – Análise Orientada a Objeto

A Análise Orientada a Objeto (AOO) corresponde à terceira etapa do desenvolvimento de um software, sendo aplicada após o levantamento de requisitos e definição dos módulos. No projeto Petro-Planner, essa análise permitiu mapear os principais elementos do domínio acadêmico de um estudante de Engenharia de Petróleo, como disciplinas, aluno, telas de interação e funcionalidades de apoio ao planejamento.

A AOO tem como objetivo descrever o que deve ser feito pelo sistema, abstraindo-se da forma como será implementado. A partir dela, são definidos:

- os objetos que representam conceitos reais do problema;
- as classes que traduzem esses objetos para o software;
- os atributos e métodos que caracterizam cada classe;
- os relacionamentos existentes entre as classes (como associação, composição e dependência);
- e as interações entre os módulos que compõem o sistema.

3.1 Classes principais

O sistema identifica como classes fundamentais:

- CAluno: representa o estudante, com seus dados pessoais, CRA, período atual e disciplinas associadas.
- CDisciplinas: representa uma disciplina do curso, com nome, carga horária, pré-requisitos e demais atributos acadêmicos.
- TelaInicial: classe que centraliza a navegação do sistema, conectando o usuário às principais funcionalidades.
- ModuloGradeCompleta: exibe visualmente a grade curricular, destacando disciplinas conforme seu status.
- QuadroDeHorarios: permite a organização visual da semana do aluno, distribuindo aulas e atividades em uma tabela.

- SimulacaoPlanejamentoSEM: simula os próximos períodos letivos com base nas disciplinas ainda não cursadas.
- EditarDisciplinasEmCurso: interface para selecionar e ajustar as disciplinas que o aluno está cursando no momento.
- AcompanhamentoDisciplina: exibe os detalhes de uma disciplina específica, como notas, trabalhos, horários e faltas.

3.2 Relacionamentos identificados

A análise também evidenciou importantes relacionamentos entre as classes:

- A classe TelaInicial possui e manipula uma instância de CAluno, sendo o ponto central de consulta aos dados.
- CAluno mantém vetores com instâncias de CDisciplinas, classificando-as como em curso, aprovadas ou não cursadas.
- Módulos como ModuloGradeCompleta, QuadroDeHorarios e SimulacaoPlanejamentoSEM dependem dos dados da classe CAluno para funcionar corretamente.
- A interação entre classes de interface e dados é feita via composição ou agregação, com ponteiros que mantêm a comunicação entre elas.

3.3 Papéis das classes

Cada classe foi estruturada com responsabilidades bem definidas, respeitando os princípios de coesão e baixo acoplamento:

- Classes de domínio (CAluno, CDisciplinas): concentram os dados e regras do domínio acadêmico.
- Classes de interface (UI): promovem a interação com o usuário, atualizando a visualização e recebendo entradas.
- Classes de controle (como SimulacaoPlanejamentoSEM): atuam na lógica intermediária, processando dados e orquestrando funcionalidades.

3.4 Diagramas gerados

Como resultado da AOO, foram elaborados os seguintes diagramas UML:

- Diagrama de Classes: mostra as classes do sistema, seus atributos, métodos e relacionamentos.
- Diagrama de Pacotes: organiza os principais módulos e suas dependências.
- Diagramas de Sequência: ilustram cenários típicos de interação, como simular um semestre ou salvar o quadro de horários.
- Diagrama de Comunicação: destaca os objetos envolvidos em uma operação e como trocam mensagens.

- Diagrama de Estado: modela o comportamento de uma disciplina ao longo do tempo (não cursada, em curso, aprovada).
- Diagrama de Atividades: descreve o fluxo geral de ações desde o login do aluno até o uso das ferramentas.
- Diagrama de Componentes e de Implantação: representam a estrutura de arquivos do sistema e sua distribuição no ambiente computacional.

3.1 Diagramas de classes

Este diagrama representa a estrutura estática do sistema, mostrando as principais classes, seus atributos e métodos, além das relações entre elas. Por exemplo, a classe CAluno armazena os dados do estudante e interage com a classe CDisciplinas, que representa cada disciplina da grade curricular. As classes de interface como TelaInicial e SimulacaoPlanejamentoSEM utilizam essas informações para apresentar e manipular os dados na aplicação.

3.1.1 Dicionário de classes

Classe CDisciplinas

Descrição: Representa uma disciplina da grade curricular do curso de Engenharia de Petróleo.

Atributos principais:

- nome, periodo, creditos, cargaHoraria: dados básicos da disciplina.
- preRequisitos, coRequisitos: listas com os nomes das disciplinas exigidas como pré ou co-requisitos.
- eixoTematico, ciclo, nivelDificuldade: classificações curriculares para organização acadêmica.
- horaSemanalAula: indica a carga horária semanal da disciplina.
- diasHorarios: horários da disciplina no formato string para exibição e manipulação.

Métodos principais:

- getNome(), getHoras(), getCH(): acessores para nome, horas semanais e carga horária total.
- exibirInformacoes(): imprime no console os dados completos da disciplina.

Classe CAluno

Descrição:

Armazena todas as informações acadêmicas de um aluno.

Atributos principais:

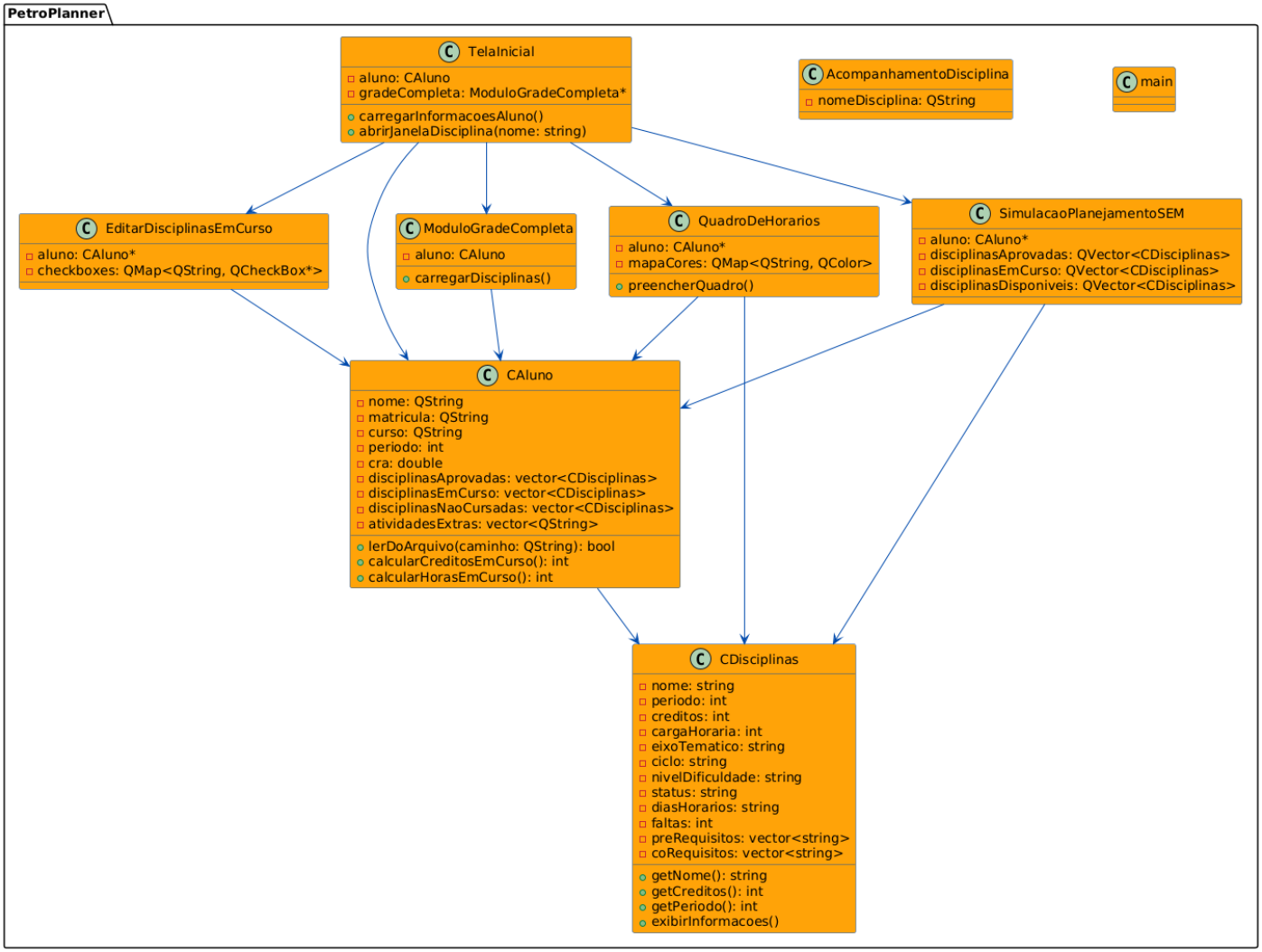

Figura 3.1: Diagrama de classes

- nome, matricula, curso, periodo, cra: dados pessoais e acadêmicos do aluno.
- disciplinasEmCurso, disciplinasAprovadas, disciplinasNaoCursadas: vetores que organizam as disciplinas de acordo com o status do aluno.
- atividadesExtras: lista com atividades não obrigatórias inseridas pelo aluno (como monitorias ou estudos livres).

Métodos principais:

- lerDoArquivo(caminho): carrega os dados do aluno a partir do arquivo InformacoesAluno.txt.
- calcularHorasEmCurso(): soma as horas semanais das disciplinas em andamento.
- calcularCHTotalCursada(): soma a carga horária total das disciplinas já aprovadas.

Classe TelaInicial

Descrição:

Interface principal da aplicação. É o ponto de entrada para todas as funcionalidades oferecidas ao usuário.

Atributos principais:

- aluno: instância da classe CAluno, contendo todos os dados do estudante logado.
- gradeCompleta: ponteiro para a janela ModuloGradeCompleta.
- telaSimulacao: ponteiro para a janela de simulação de planejamento (SimulacaoPlanejamentoSEM).

Métodos principais:

- carregarInformacoesAluno(): atualiza a interface com os dados extraídos do arquivo.
- PreencherDisciplinasEmCurso(): exibe botões com as disciplinas em andamento.
- abrirJanelaDisciplina(nomeDisciplina): abre a janela de acompanhamento daquela disciplina.
- abrirTelaSimulacao(), abrirQuadroDeHorarios(), abrirEditorDeDisciplinas(): métodos de navegação entre módulos.

Classe ModuloGradeCompleta

Descrição:

Interface visual responsável por exibir toda a grade curricular do curso, destacando disciplinas aprovadas, em curso e não cursadas.

Relação com CAluno:

- Recebe um ponteiro ou instância de CAluno para montar a grade com base nos dados do estudante.

Métodos principais:

- carregarDisciplinas(): percorre o vetor de disciplinas e monta visualmente a grade, utilizando cores ou estilos diferentes conforme o status da disciplina.

Classe EditarDisciplinasEmCurso

Descrição:

Interface que permite ao usuário adicionar ou remover disciplinas que está cursando no momento.
Atributos principais:

- aluno: ponteiro para o aluno atual.
- checkboxes: mapa entre o nome das disciplinas e os checkboxes que representam o estado de seleção.

Métodos principais:

- on_buttonBox_accepted(): salva as alterações feitas pelo usuário no arquivo InformacoesAluno.txt.
- on_buttonBox_rejected(): descarta mudanças e fecha a interface.

Classe QuadroDeHorarios

Descrição:

- Tela que permite ao usuário visualizar, editar e exportar seu quadro semanal de horários.

Atributos principais:

- aluno: ponteiro para o aluno atual.
- mapaCores: mapeia disciplinas/atividades para cores fixas.
- coresDisponiveis: lista de cores pré-definidas.
- modoEdicaoAtivo: controla o modo de edição.

Métodos principais:

- preencherQuadro(): popula a grade visual com as disciplinas e atividades do aluno.
- aoClicarSalvar(): atualiza o quadro e grava os novos horários no arquivo.
- salvarQuadroComoImagem(): exporta o quadro como imagem .png.

Classe SimulacaoPlanejamentoSEM

Descrição:

Interface que permite simular futuros períodos do curso, testando combinações de disciplinas elegíveis.
Atributos principais:

- aluno: ponteiro para o aluno atual.
- semestres: vetor com as simulações feitas.
- disciplinasDisponiveis, disciplinasAprovadas, disciplinasEmCurso: vetores que filtram as disciplinas segundo critérios acadêmicos.

Métodos principais:

- avancarSemestre(): cria um novo bloco visual com disciplinas possíveis para o próximo período.
- salvarComoImagem(): exporta a simulação feita como imagem.
- preRequisitosOk(disciplina): verifica se os pré-requisitos de uma disciplina foram cumpridos.

Classe AcompanhamentoDisciplina

Descrição:

Janela de detalhes que exibe o progresso em uma disciplina específica.
Atributos principais:

- ui: componente de interface gerado pelo Qt Designer.
- nomeDisciplina: nome da disciplina selecionada.

Métodos principais:

- Construtor e destrutor para iniciar e liberar a interface corretamente.
- Métodos de exibição (como atualização de nota, faltas, observações), caso existam.

3.2 Diagrama de Sequência – Eventos e mensagens gerais

Este diagrama representa o fluxo de execução típico do sistema durante a interação do usuário com a interface principal. Ele descreve como os objetos trocam mensagens e em que ordem isso ocorre. A sequência inicia com ações do usuário, como clicar em botões, e segue com chamadas a métodos das classes para carregar, processar e exibir informações sobre disciplinas, faltas e simulações.

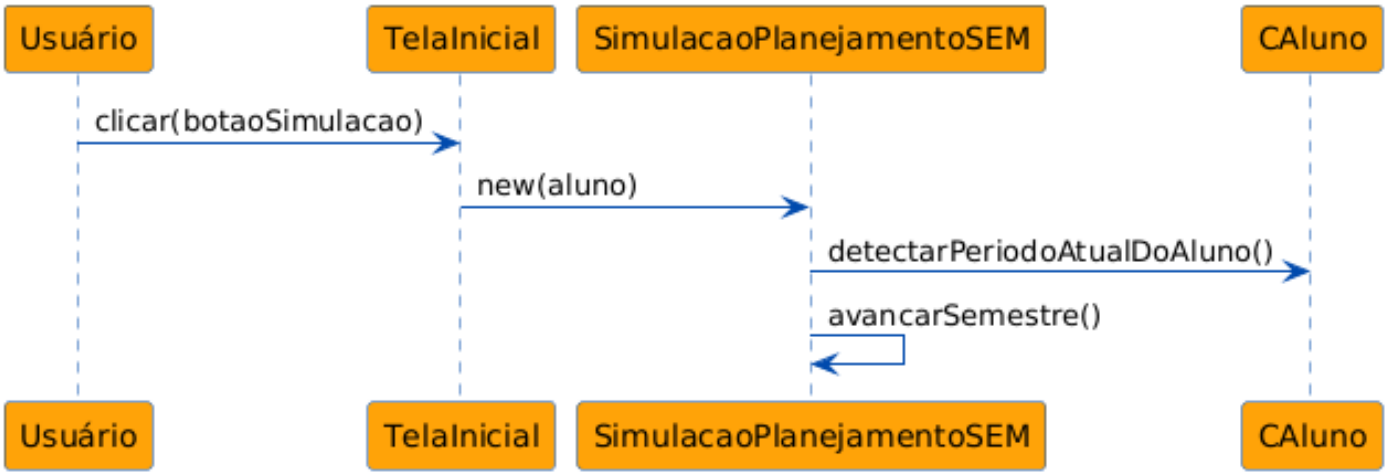

Figura 3.2: Diagrama de seqüência - eventos e mensagens

3.2.1 Diagrama de Sequência Específico – Selecionar disciplina na simulação

Este diagrama representa um cenário específico de interação: a seleção de disciplinas durante a simulação de planejamento. Ele detalha como o sistema responde à ação do usuário ao escolher uma disciplina, atualizando a lista de disciplinas simuladas, criando botões visuais e verificando os pré-requisitos. É útil para demonstrar a lógica de um caso de uso pontual e importante no sistema.

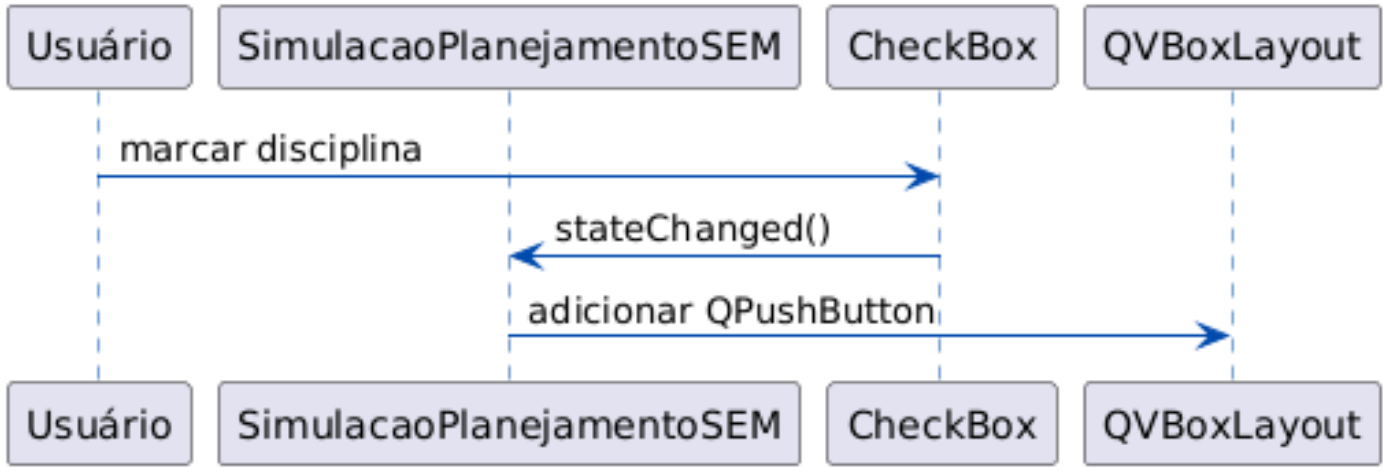

Figura 3.3: Diagrama de sequência específico

3.3 Diagrama de Comunicação – Colaboração entre classes

Este diagrama representa a interação entre os objetos durante a execução de funcionalidades, com ênfase na troca de mensagens. Ao contrário do diagrama de sequência, que foca na ordem, este mostra quem colabora com quem. Ele evidencia, por exemplo, como a tela de simulação (SimulacaoPlanejamentoSEM) comunica-se com o objeto CAluno e como este acessa as informações das disciplinas.

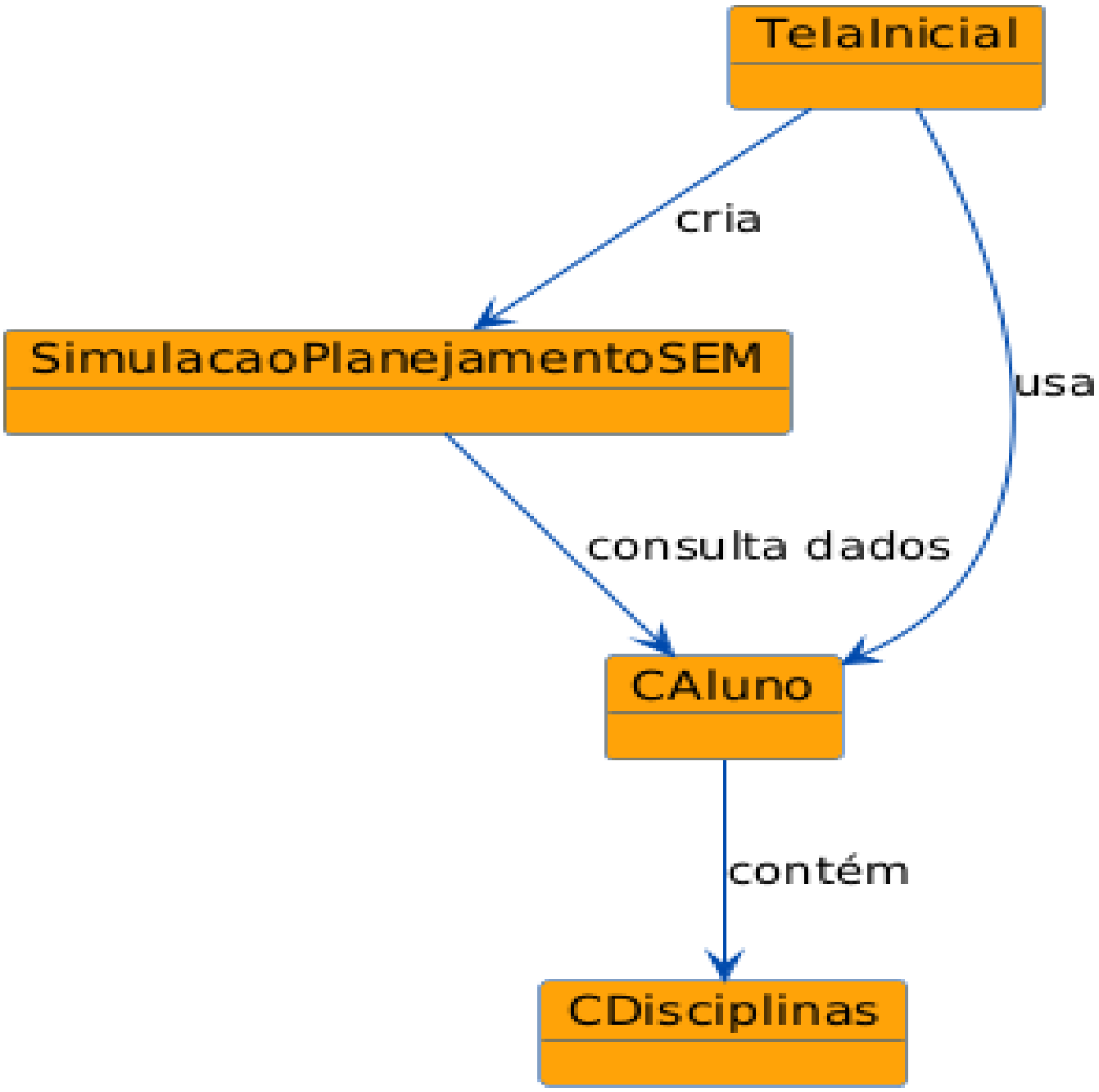

Figura 3.4: Diagrama de Comunicação – Colaboração entre classes

3.4 Diagrama de Estados – Estados da disciplina na simulação

Este diagrama representa o ciclo de vida de uma disciplina dentro do sistema, de acordo com a situação do aluno. Uma disciplina pode estar nos estados “Não Cursada”, “Em Curso” ou “Aprovada”, e as transições entre esses estados ocorrem conforme os dados do aluno são atualizados ou modificados pelo usuário. Esse diagrama é útil para representar regras de negócio relacionadas ao progresso acadêmico.

Figura 3.5: Diagrama de máquina de estado

3.5 Diagrama de Atividades – Fluxo da tela de simulação

Este diagrama representa o fluxo de atividades do usuário na tela de simulação de planejamento. Ele mostra as decisões possíveis, como selecionar disciplinas elegíveis, avançar para o próximo semestre ou cancelar a ação. Essa representação ajuda a visualizar o comportamento do sistema sob a perspectiva do usuário e garante que todas as possibilidades estão cobertas no design.

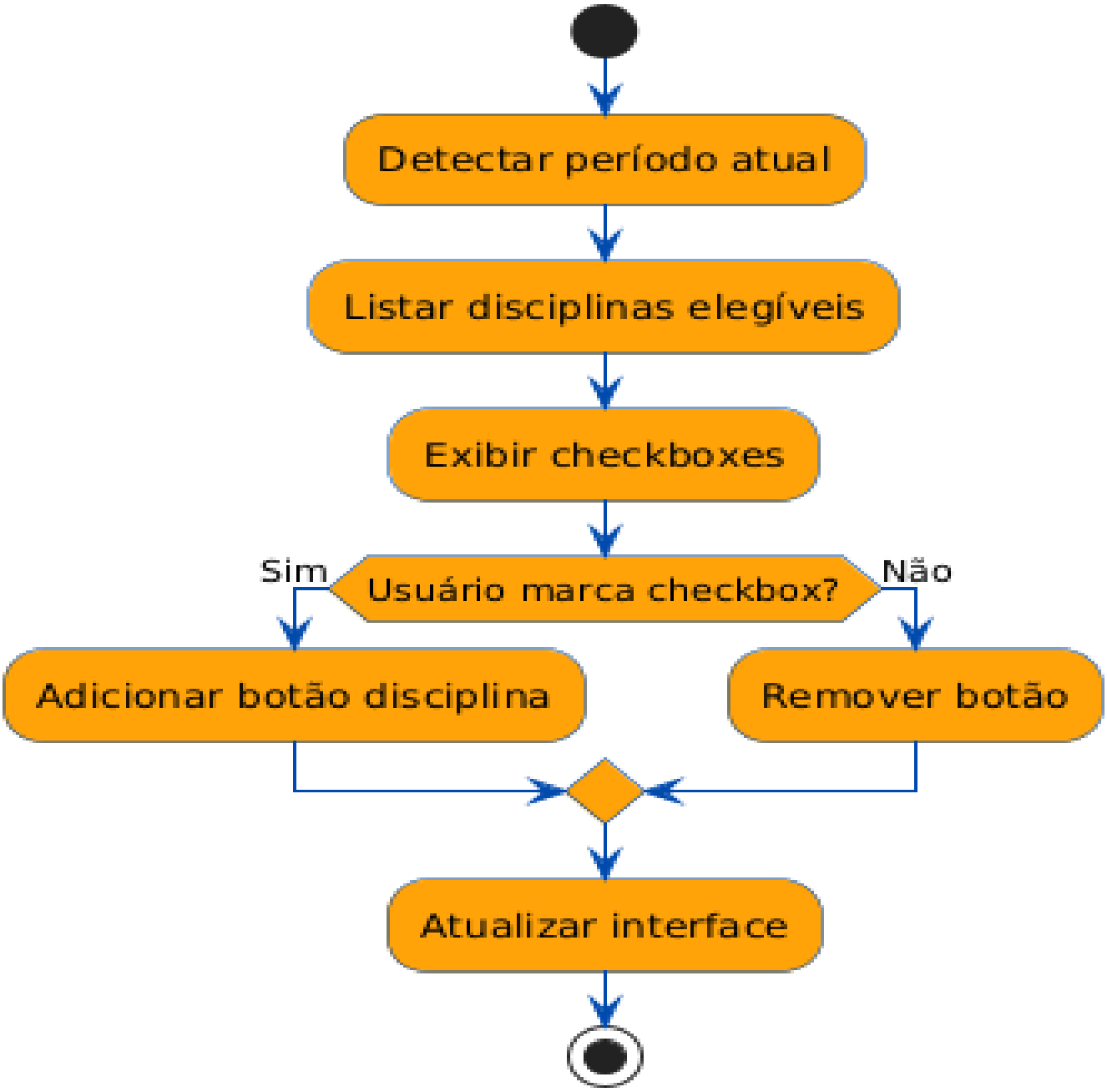

Figura 3.6: Diagrama de atividades

Capítulo 4

Projeto

4.1 Projeto do Sistema

O projeto do sistema PetroPlanner foi estruturado com base na análise orientada a objeto e nas necessidades práticas dos estudantes de Engenharia de Petróleo. O sistema foi concebido como uma aplicação desktop autônoma, utilizando C++ com o framework Qt para a interface gráfica e manipulação de dados.

1. Protocolos

   - Entre elementos externos: A aplicação não depende de comunicação com dispositivos externos ou APIs. O foco está na interação direta com o usuário por meio da interface gráfica.
   - Entre elementos internos: Os módulos se comunicam por meio de chamadas de métodos diretas, utilizando objetos e ponteiros. O sistema de sinais e slots do Qt também é amplamente utilizado para eventos da interface.
   - Interface das classes: As APIs internas são definidas pelos arquivos de cabeçalho C++ (.h), que especificam os métodos públicos de cada classe.
   - Formato de arquivos: As informações do aluno são salvas em arquivos .txt, com estrutura delimitada por ponto e vírgula (;), facilitando leitura e edição.

2. Recursos

   - Uso de memória: O gerenciamento é feito pelo Qt, que possui hierarquia de objetos parent-child, e pelo uso consciente de new/delete e containers da STL.
   - Banco de dados: Não foi usado um banco de dados tradicional. Os dados são persistidos em arquivos de texto, mas a estrutura pode ser adaptada futuramente para uso de SQLite.
   - Armazenamento: Todo o conteúdo é armazenado localmente no computador do usuário. A portabilidade é garantida pelos arquivos simples e leves.

3. Controle

- Baseado em eventos: O sistema responde a eventos do usuário (cliques, seleções), utilizando o sistema de event loop do Qt.
- Condições extremas: São tratadas por meio de verificações de entrada (como número de faltas, notas fora do padrão) e mensagens de alerta.
- Concorrência: Como o sistema é leve e voltado para tarefas rápidas, a concorrência ainda não foi necessária, mas o Qt oferece suporte a QThread para futuras otimizações.
- Escalas de tempo: As interações são imediatas. Alterações em dados são salvas logo após ações do usuário (ex: clicar em "Salvar").

4. Plataformas

- Arquitetura: O projeto adota uma estrutura próxima ao padrão MVC: dados (modelos), telas (views) e funções de controle (slots e métodos).
- Plataformas suportadas: Qt permite compatibilidade com Windows, Linux e macOS.
- Bibliotecas utilizadas:
- Qt: Interface gráfica, gerenciamento de eventos, arquivos.
- STL: Containers (vector, map, etc.).
- IDE: Qt Creator, voltado para projetos em C++/Qt.

5. Padrões de projeto

Foram seguidos princípios básicos de projeto orientado a objeto:

- Modularidade: cada classe tem uma responsabilidade bem definida.
- Baixo acoplamento e alta coesão: a comunicação entre os módulos se dá por meio de interfaces diretas ou sinais.
- Padrões usados:
  - MVC: organização entre interface, lógica e dados.
  - Observer (via sinais e slots): resposta a eventos na interface.
  - Singleton (implícito para classes que representam o aluno ativo).

4.2 Projeto Orientado a Objeto – POO

O projeto orientado a objeto levou em consideração os recursos da linguagem C++ e do framework Qt. As classes definidas na análise foram detalhadas com os métodos, atributos e estruturas necessárias para implementação real.

Efeitos do projeto no modelo estrutural

- Diagramas de pacotes foram atualizados para incluir subsistemas como GUI, lógica, e persistência.
- Diagramas de classes foram refinados com atributos como QPushButton*, QVBoxLayout* e métodos como on_botaoVerGradeCompleta_clicked().

Efeitos do projeto no modelo dinâmico

- Diagramas de sequência e comunicação foram atualizados para refletir chamadas de métodos reais entre objetos da interface (ex: botão de salvar → método que grava arquivo).
- Diagrama de atividades agora inclui verificações de dados, simulações e salvamento.
- Máquinas de estado modelam, por exemplo, o ciclo de vida de uma disciplina (não cursada → em curso → aprovada).

Efeitos do projeto nos atributos

- Novos atributos: ponteiros para elementos da interface (ex: QLabel* labelCRA) e flags internos (bool modoEdicaoAtivo).

Efeitos do projeto nos métodos

- Novos métodos: slots de eventos do Qt (ex: void aoClicarSalvar()), métodos de controle da interface e salvamento de dados em arquivos.

Efeitos do projeto nas heranças

- Herança: usada com Qt, onde classes como TelaInicial herdam de QMainWindow.

Efeitos do projeto nas associações

- Associações: implementadas via ponteiros e containers como QVector<CDisciplinas>.

Efeitos do projeto nas otimizações

- Desempenho: as telas foram desenhadas para responder rapidamente, mesmo com muitos dados.
- Organização: classes auxiliares foram criadas para manter o código limpo.
- Escalabilidade: estrutura preparada para expansão futura (ex: suporte a banco de dados ou exportação de relatórios).

4.3 Diagrama de Componentes

Este diagrama representa os módulos físicos do sistema e suas dependências. Cada componente encapsula uma parte funcional, como o módulo de leitura de arquivos, o módulo de simulação ou o acompanhamento de disciplinas. Ele é importante para entender como o sistema é dividido em unidades reutilizáveis e independentes.

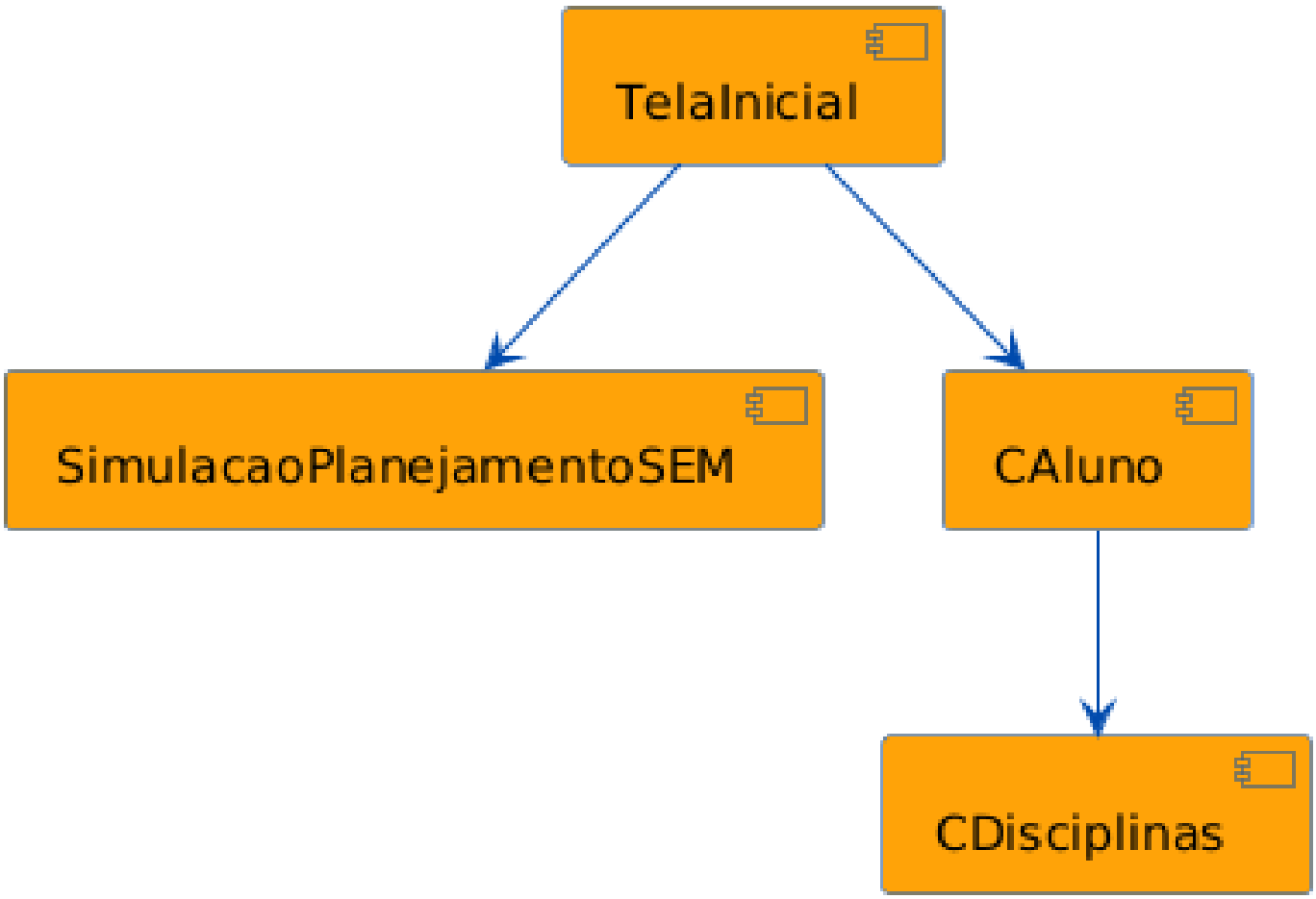

Figura 4.1: Diagrama de componentes

4.4 Diagrama de Implantação

Este diagrama representa a arquitetura física de execução do sistema. Mostra onde os componentes do sistema estão implantados, como no ambiente local (computador pessoal), com o uso de Qt para a interface gráfica e o arquivo InformacoesAluno.txt como banco de dados local. Ele ajuda a visualizar como o sistema será utilizado no mundo real.

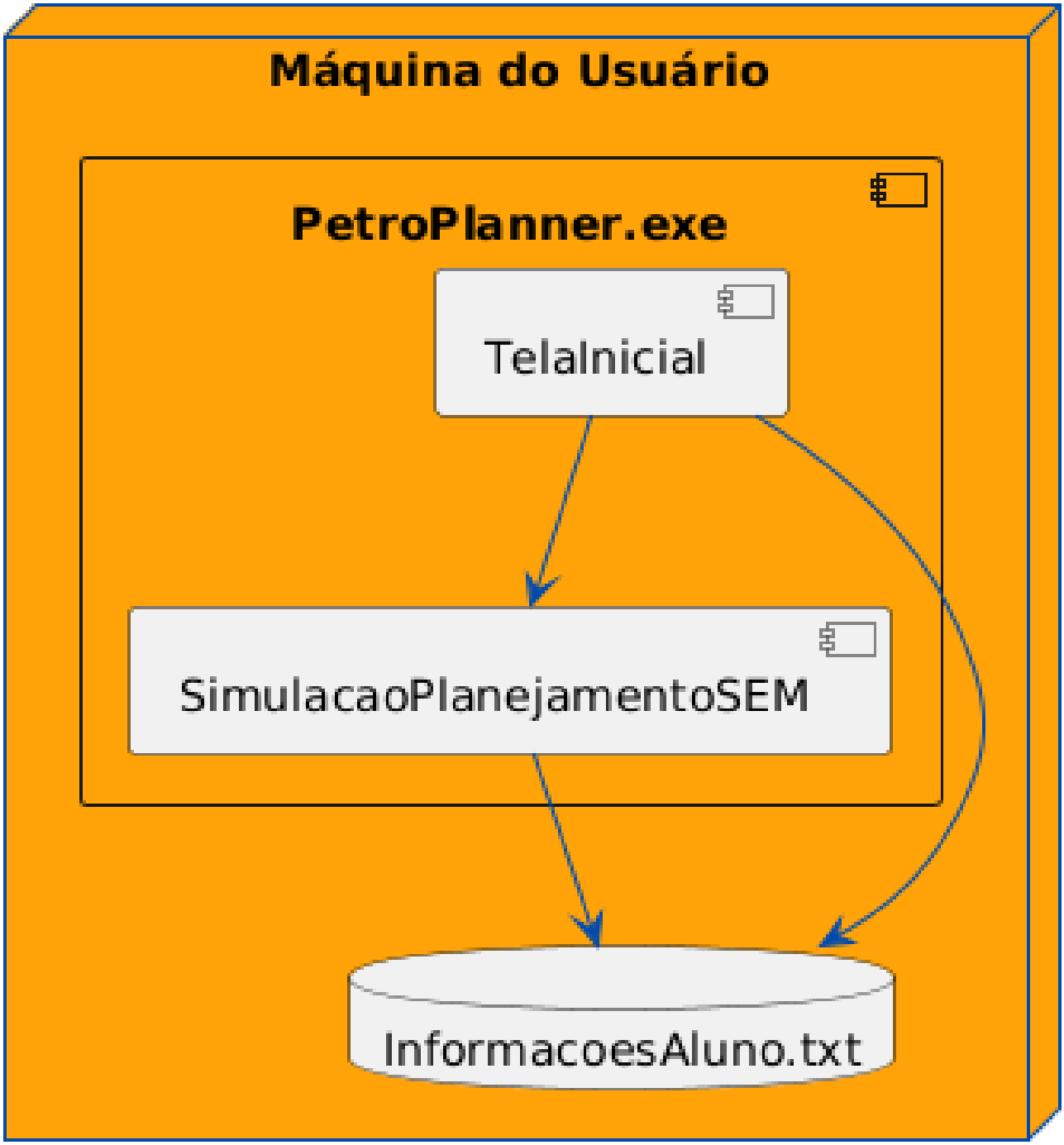

Figura 4.2: Diagrama de implantação

Capítulo 5

Lista das Características/*Features*

Neste capítulo apresenta-se a lista de funcionalidades (features) implementadas no sistema Petro-Planner, estruturadas com base na abordagem de análise e desenvolvimento orientado a objetos, com entregas incrementais.

As funcionalidades foram organizadas em versões correspondentes aos ciclos de desenvolvimento (v0.1, v0.3, v0.7), refletindo a maturação gradual do sistema e o avanço da equipe na construção dos módulos planejados.

Cada feature foi pensada para ser desenvolvida em um único ciclo, permitindo progresso contínuo, revisões constantes e validação prática em cada etapa.

5.1 Lista de características <<features>>

Versão v0.1 – Núcleo de Dados e Estrutura Inicial

Nesta fase, o foco esteve na definição das classes centrais, na estrutura de dados e na leitura automatizada das informações do aluno. Essas funcionalidades representam a base de todo o sistema.

Features v0.1:

- Leitura de arquivos .txt contendo os dados do aluno.
- Armazenamento organizado das disciplinas aprovadas, em curso e não cursadas.
- Implementação da classe CAluno, responsável por representar o aluno no sistema.
- Implementação da classe CDisciplinas, representando a estrutura de uma disciplina.
- Implementação do método lerDoArquivo(), que interpreta e estrutura os dados salvos.
- Cálculo automático da carga horária cursada via calcularHorasEmCurso() e calcularCHTotalCursada().

Versão v0.2 – Integração dos Dados com a Interface Gráfica

Nesta fase, iniciou-se a construção da interface gráfica utilizando Qt, conectando os dados do aluno e das disciplinas com os elementos visuais do sistema. Com isso, o usuário passou a interagir diretamente com suas informações acadêmicas organizadas.

Funcionalidades v0.2:

- Implementação da TelaInicial com exibição de informações do aluno (nome, matrícula, CRA etc.).
- Conexão entre as estruturas de dados do aluno e disciplinas e a interface gráfica.
- Implementação da janela ModuloGradeCompleta, exibindo a grade curricular completa do curso.
- Uso de cores distintas para indicar o status das disciplinas (aprovadas, em curso, reprovadas, não cursadas).
- Criação da barra de progresso de carga horária semanal (limite de 16h).
- Implementação do método carregarInformacoesAluno() para exibir os dados do aluno na interface.
- Testes básicos de navegação entre telas com botões funcionais.
- Leitura e exibição dinâmica da grade curricular a partir dos dados das disciplinas do curso.

Versão v0.3 – Interatividade Avançada e Módulos Auxiliares

Na fase final, o sistema recebeu melhorias significativas em usabilidade e inteligência de simulação, com janelas específicas para acompanhar desempenho, planejar semestres e visualizar o quadro de horários semanal.

Features v0.3:

- Implementação da janela AcompanhamentoDisciplina, com simulação de notas e cálculo de média final.
- Cálculo do que o aluno precisa tirar nas próximas avaliações para ser aprovado.
- Implementação do método abrirJanelaDisciplina() na TelaInicial, com ligação entre as disciplinas da grade e suas janelas detalhadas.
- Implementação do módulo SimulacaoPlanejamentoSEM, simulando planejamento de semestres futuros.
- Checagem automática de pré-requisitos ao simular disciplinas elegíveis.
- Implementação do QuadroDeHorarios, preenchido com base nas disciplinas em andamento e atividades extras.
- Armazenamento e leitura automatizada dos horários no arquivo InformacoesAluno.txt.
- Testes com diferentes arquivos de entrada simulando perfis variados de alunos (CRA, disciplinas, horários).

Referências Bibliográficas

Índice Remissivo