Programação Avançada - LIC EIRP
2021/2022

Relatório da Fase 2
do Trabalho Prático

Nuno Almeida, 2019121655
Sandra Perdigão, 2019102697

Índice

1. Identificação

1.1. Fase a que diz respeito o documento

O documento diz respeito à **Fase 2**.

1.2. Turma Prática

A Turma Prática é a **P6**.

1.3. Grupo Atribuído

O grupo atribuído é o nº **22**.

1.4. Nome, número e endereço de email de cada elemento do grupo

- Nuno Almeida, 2019121655, nunofgsa@gmail.com / 2019121655@isec.pt
- Sandra Perdigão, 2019102697, sandraperdigao@hotmail.com / 2019102697@isec.pt

2. Introdução

Este trabalho prático consiste na implementação de uma aplicação de apoio ao processo de gestão de projetos e estágios do Departamento de Engenharia Informática e de Sistemas do ISEC. Relativamente à fase anterior, inclui algumas alterações significativas, como por exemplo uma interface gráfica. É possível guardar e continuar um estado anterior, assim como inserir e remover dados, fazer undo e redo nas atribuições de propostas e orientadores, entre outros.

Esta aplicação é constituída por cinco fases principais:

- Fase 1 - Configuração;
- Fase 2 - Opções de Candidatura;
- Fase 3 - Atribuição de Propostas;
- Fase 4 - Atribuição de Orientadores;
- Fase 5 - Consulta.

Consoante a fase em que se encontra o processo de gestão, a interface com o utilizador permite a introdução de diversos tipos de dados através de ficheiros CSV ou por inserção manual. Estes podem ser dados de alunos, dados de professores, dados de projetos/ estágios, e dados de candidaturas.

Com estes dados, a parte lógica da aplicação permite realizar algumas funcionalidades, tais como atribuir propostas aos alunos, consoante os dados das suas candidaturas, assim como atribuir orientadores às propostas.

A implementação desta aplicação teve por base os padrões apresentados nas aulas de Programação Avançada, tendo sido aplicado o padrão Finite State Machine (“máquina de estados finita”), para concretização da lógica de controle do processo de gestão dos projetos e estágios.

3. Organização Lógica

Segundo as indicações sobre a arquitetura do projeto, foram adicionados diversos packages de modo a estruturar logicamente o código. Estão organizados da seguinte forma:

- **log** - Regista informações gerais sobre o fluxo do programa. Permite que sejam apresentadas mensagens múltiplas ao utilizador quando vários eventos de erros surgem, como por exemplo na importação de ficheiros.
- **commands** - Suporta a implementação do padrão Command.
- **ui** - Interfaces de utilizador implementadas. Inclui a interface de texto, no package **text**, e interface gráfica no package **gui**.
- **gui** - Classes de suporte à GUI.
    - **UI** - Panes que representam estados da máquina de estados
    - **resources** - Recursos usados no trabalho, nomeadamente imagens, ficheiros CSS.
- **utils** - Classes de utilização geral ao programa. Incluem métodos úteis para exportação/ importação de ficheiros CSV, entre outros métodos gerais necessários ao bom funcionamento da aplicação.
- **model** - Inclui as classes que realmente implementam a aplicação e toda a lógica associada ao mesmo. Dentro deste package incluem-se outros dois: **data** e **fsm**.
- **data** - Contém as classes que constituem os dados necessários à aplicação. No interior deste package temos acesso a outros packages:
    - **applications** - Classes que contêm a lógica das candidaturas;
    - **attribution** - Classes que contêm a lógica das atribuições estudantes/propostas e propostas/orientadores;
    - **enums** - Classes que contêm valores finitos que diversos campos de dados podem tomar.
    - **flags** - Classe que contém a informação sobre o fecho das várias fases;
    - **proposal** - Classes que contêm a lógica das propostas;
    - **student** - Classes que contêm a lógica dos estudantes;
    - **teacher** - Classes que contêm a lógica dos professores;
- **fsm** - Classes associadas à implementação do padrão Máquina de Estados Finita.
    - **states** - Classes associadas à implementação do padrão Máquina de Estados.

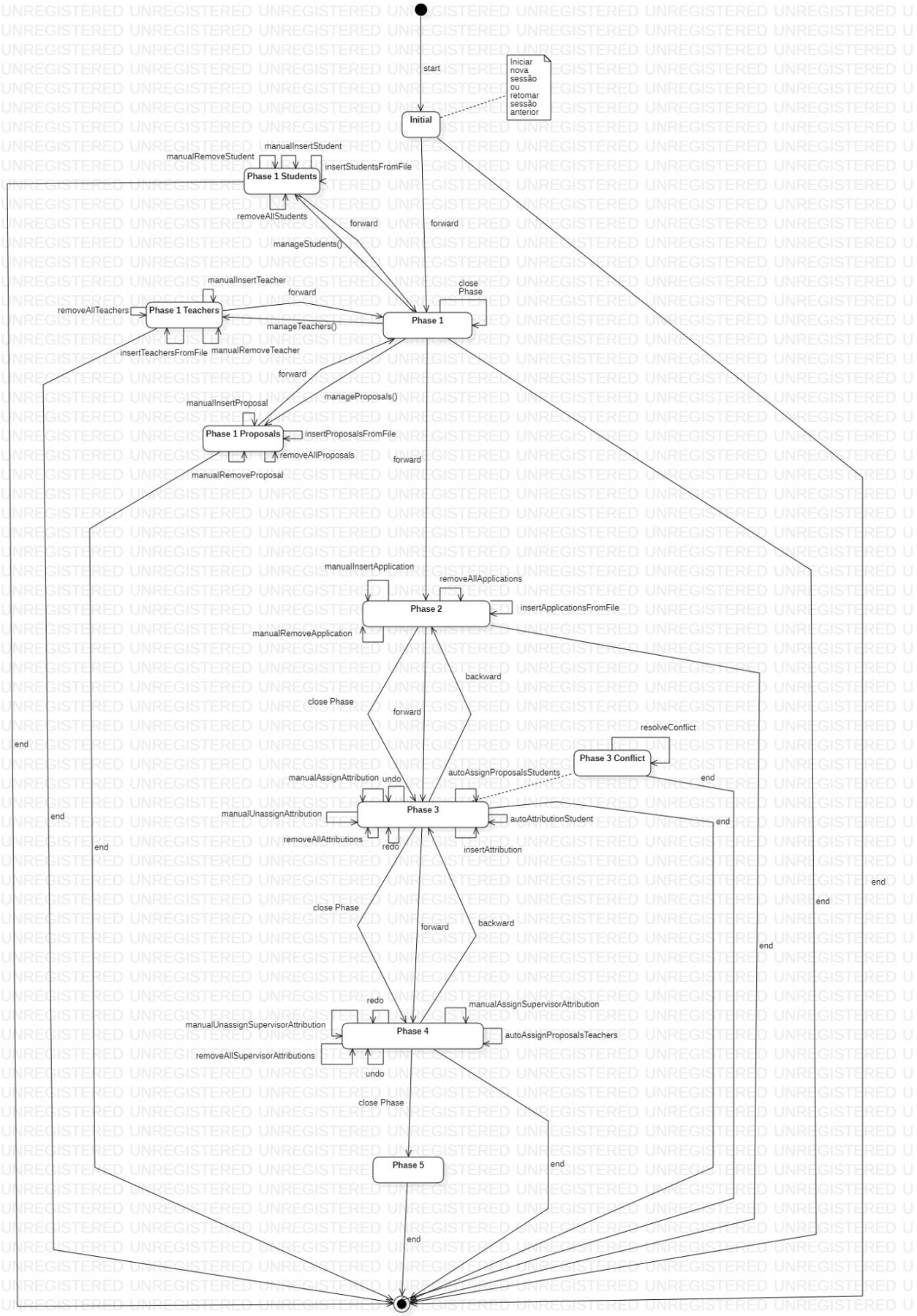
4. Padrões de Programação

4.1. Máquina de Estados Finita

Para a implementação da aplicação, a interação UI - aplicação é realizada pelo design pattern Máquina de Estados Finita. Assim, é possível aceder a informação essencial à UI e mudar o estado do programa, mantendo os princípios fulcrais de programação orientada a objetos, como o encapsulamento.

Abaixo encontra-se o diagrama de estados que serve de base à implementação. Definiram-se os estados, tendo como princípio os momentos em que é obrigatório o input do utilizador, ou apenas por uma questão de compreensão de acontecimentos por parte do utilizador (como, por exemplo, a transição das diversas fases). O esquema abaixo é autoexplicativo, no entanto, as condições de mudança de estado não estão representadas nele. São as seguintes:

- A transição **forward** entre o estado *Initial* e *Phase 1* ocorre quando o utilizador opta por iniciar uma nova sessão. Caso pretenda retomar uma sessão anterior, o estado onde a aplicação inicia é o estado em que ficou na última sessão.
- A transição **forward** entre o estado *Phase 1* e *Phase 2* ocorre se, para cada ramo, o número total de propostas for igual ou superior ao número de alunos. O método **closePhase** no estado Phase 1 mantém ativo este estado, mas não permite futuras alterações e mantém a possibilidade de consultar os dados.
- A transição **closePhase** entre o estado *Phase 2* e *Phase 3* apenas é permitida caso a *Phase 1* já esteja fechada. Depois de fechada a *Phase 2*, não são permitidas futuras alterações e mantém-se a possibilidade de consultar os dados. A transição **forward** é permitida independentemente do fecho da fase.
- A transição **closePhase** entre o estado *Phase 3* e *Phase 4* apenas é permitida caso todos os alunos com candidaturas submetidas possuam projeto atribuído. Depois de fechada a *Phase 3*, não são permitidas futuras alterações e mantém-se a possibilidade de consultar os dados. A transição **forward** é permitida independentemente do fecho da fase.
- A transição **closePhase** entre o estado *Phase 4* e *Phase 5* é permitida sem restrições. Depois de fechada a *Phase 4*, no estado *Phase 5* não é possível regressar a qualquer uma das fases anteriores.


start
Iniciar nova sessão ou retomar sessão anterior
Initial
manualInsertStudent
manualRemoveStudent
insertStudentsFromFile
Phase 1 Students
removeAllStudents
forward
forward
manageStudents()
manualInsertTeacher
forward
close Phase
removeAllTeachers
Phase 1 Teachers
manageTeachers()
Phase 1
insertTeachersFromFile
manualRemoveTeacher
forward
manageProposals()
manualInsertProposal
Phase 1 Proposals
insertProposalsFromFile
removeAllProposals
forward
manualRemoveProposal
manualInsertApplication
removeAllApplications
Phase 2
insertApplicationsFromFile
manualRemoveApplication
backward
close Phase
forward
resolveConflict
Phase 3 Conflict
end
manualAssignAttribution
undo
autoAssignProposalsStudents
manualUnassignAttribution
Phase 3
autoAttributionStudent
removeAllAttributions
redo
insertAttribution
close Phase
forward
backward
redo
manualAssignSupervisorAttribution
manualUnassignSupervisorAttribution
Phase 4
autoAssignProposalsTeachers
removeAllSupervisorAttributions
undo
close Phase
Phase 5
end

4.2. Factory

O padrão Factory abstrai a criação de certos objetos, simplificando o código onde for usado. Assim, disponibiliza um método que permite a criação de uma instância de uma classe entre um conjunto de várias possibilidades. No presente trabalho usamos duas vezes o padrão Factory: na criação dos estados, devolvendo o estado desejado mediante o valor da enum **StateDataEnum**; e na criação das propostas na package data, tendo em conta a proposta selecionada.

4.3. Mediator

O padrão Mediator reduz o acoplamento entre os componentes de uma aplicação, permitindo a comunicação indireta através de um objeto “Mediador”. No presente trabalho usámos este padrão através das classes Context e PoEManager, ou seja, a classe PoEManager é utilizada para gerir a classe Context.

4.4. Singleton

O padrão Singleton permite que só exista uma instância de uma determinada classe, tendo assim como maior vantagem que possa ser utilizada a única instância dessa classe em qualquer parte do código, devolvendo uma única referência através do método getInstance. Assim, a título de exemplo, é possível utilizar funcionalidades de classes hierarquicamente superiores em classes mais profundas na estrutura do código. Este padrão foi utilizado no nosso trabalho para registar eventos internos da aplicação através do objeto Logger, o que é bastante útil para ajudar a manter o controlo de eventos, apresentar uma mensagem no ecrã, ajudar a localizar a raiz de um problema da aplicação, entre outros. Também utilizámos este padrão para gravar o estado da aplicação, pois uma vez que estamos a usar um PoEManager, que é único durante todo o decorrer do programa, e necessitamos da sua referência no StateInitial no caso de o utilizador optar por carregar um estado anterior.

4.5. Facade

O padrão Facade permite que seja colocada uma classe intermediária que servirá de ponto de acesso às funcionalidades disponibilizadas por outras classes. A classe Facade deve fornecer métodos que permitam redirecionar as execuções pretendidas para os objetos internos dos packages que se pretende que tenham visibilidade limitada. Este padrão foi utilizado no nosso trabalho através da classe DataManager, que contém métodos que, consoante o que é solicitado, permitem redirecionar para as classes de dados pretendidas (StudentData, TeacherData, etc.) ao mesmo tempo que opera funcionalidades mais complexas que requerem acesso a vários tipos de dados.

4.6. Command

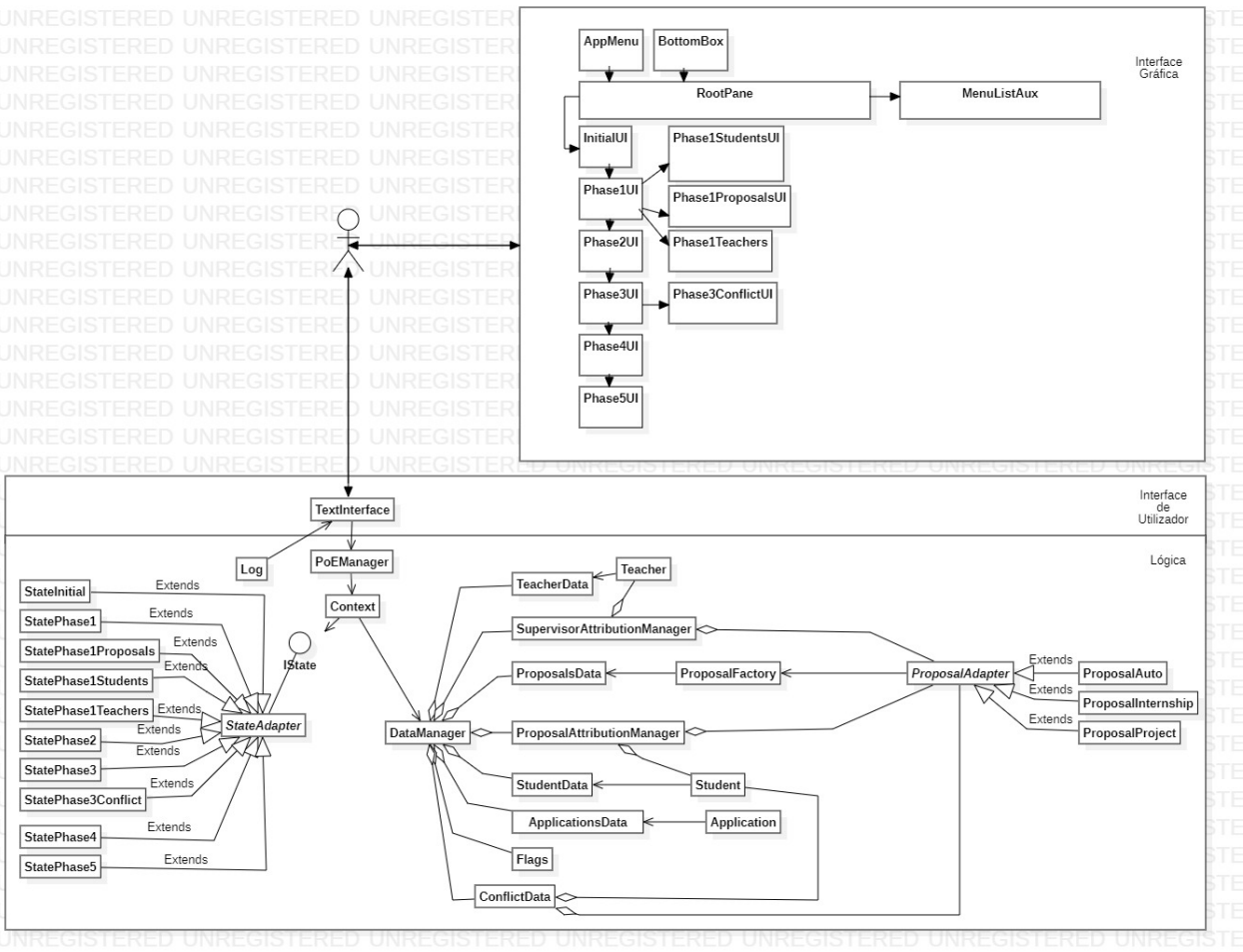
Com o objetivo de implementar a funcionalidade de undo e redo, foi usado o padrão Command. Associado ao Command, há um histórico das últimas ações efetuadas no que diz respeito a atribuições de propostas a estudantes (Fase 3), e de atribuições de propostas a orientadores, já associadas a estudantes (Fase 4).

5. Classes

Algumas das classes mais importantes da nossa implementação são as seguintes:

- Classe **TextInterface** - Classe que trata de toda a interação com o utilizador.
- Classe **CommandManager** - Permite a implementação do Pattern Command.
- Classe **Context** - É composta por um objeto do tipo **StateAdapter** e por um objeto **DataManager**, controlando a interação com ambos.
- Classe **PoEManager** - É composta por um Contexto, e tem função de gerir a UI. É um Singleton.
- Classe **StateAdapter** - Representa, de forma abstrata, os estados possíveis da máquina de estados. É extendida pelas classes dos estados possíveis da aplicação.
- Classe **DataManager** - É a classe que gere os dados que são posteriormente devolvidos à interface com o utilizador, para serem mostrados ou apresentar alguma escolha ao utilizador. Engloba uma grande parte da exportação de informação. Esta classe é composta por instâncias de classes agregadoras de dados, na sua maioria baseadas em HashMaps, que permitem a segurança da inserção controlada de elementos.
- Classe **Flags** - Contém informação relativamente ao fecho das várias fases.
- Classe **Log** - Classe que auxilia toda a aplicação a apresentar informação ao utilizador e a registar eventos relevantes de funcionamento da aplicação.
- Classe **Configs** - Permite ao utilizador se pretende lançar a interface de texto ou a interface gráfica.

Nota: As classes utilitárias e a enum não se encontram no esquema, devido a serem públicas e acessíveis durante todo o programa.


Interface Gráfica
AppMenu
BottomBox
RootPane
MenuListAux
InitialUI
Phase1StudentsUI
Phase1UI
Phase1ProposalsUI
Phase1Teachers
Phase2UI
Phase3UI
Phase3ConflictUI
Phase4UI
Phase5UI
Interface de Utilizador
TextInterface
Lógica
Log
PoEManager
Context
IState
StateInitial
StatePhase1
StatePhase1Proposals
StatePhase1Students
StatePhase1Teachers
StatePhase2
StatePhase3
StatePhase3Conflict
StatePhase4
StatePhase5
Extends
StateAdapter
DataManager
TeacherData
Teacher
SupervisorAttributionManager
ProposalsData
ProposalFactory
ProposalAdapter
ProposalAttributionManager
StudentData
Student
ApplicationsData
Application
Flags
ConflictData
ProposalAuto
ProposalInternship
ProposalProject

6. Funcionalidades Implementadas

Componente do Trabalho	Realizado	Não realizado
Inclusão das operações Undo e Redo	x	
Implementação da aplicação com uma interface com o utilizador (IU) em modo gráfico (JavaFX).	x	
Introdução e gestão de dados realizada de forma interactiva na IU (mantendo-se a possibilidade de importar os dados a partir de ficheiros CSV).	x	
Apresentação de gráficos de resumo do processo	x	
Painel de resumo da entidade que se encontra a ser gerida em cada momento	x	
Código devidamente comentado usando JavaDoc		x
Edição de dados		x