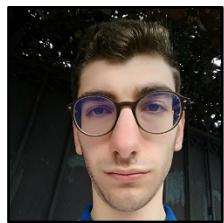


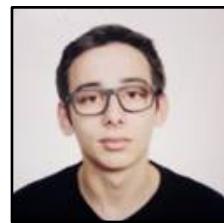
Trabalho Prático DSS

FASE II

Braga, 26 de dezembro de 2017



Francisco Oliveira



Raul Vilas Boas



Vitor Peixoto

A78416

A79617

A79175

Grupo 22

Mestrado Integrado em Engenharia Informática

Universidade do Minho

Introdução

Neste trabalho prático foi-nos pedida a criação de um sistema de gestão de horários. Essa aplicação deverá ser capaz de suportar o registo de Unidades Curriculares (UC's) e alunos, bem como a criação de turnos e respetivas trocas entre alunos. Temos de respeitar as três fases do projeto especificadas no enunciado do projeto e ainda o estatuto especial dos estudantes em regime de pós-laboral.

Este sistema deve ser robusto na medida em que respeita os horários dos docentes, a alocação de turnos e os horários dos estudantes.

Este projeto deve ser inicialmente planeado e modelado respeitando os princípios da programação orientada aos objetos. Para tal usamos como linguagem de modelação a *Unified Modelling Language (UML)*. A *UML* facilita a comunicação e análise, pelo que se torna essencial modelar o projeto a desenvolver, facilitando e orientando o processo de desenvolvimento do programa.

Iremos fazer a modelação necessária e adequada para o desenvolvimento do software, tendo em conta o conteúdo lecionado nas aulas teóricas.

Este projeto terá de estar suportado por uma base de dados relacional. Torna-se então necessário desenvolver um esquema lógico que se adapte às necessidades da aplicação, definidas pela modelação.

Finalmente é preciso desenvolver a aplicação em si, tendo em conta os requisitos dados pelo enunciado e modelados pelo *UML*, usando um ambiente de desenvolvimento integrado e uma linguagem adequada à programação orientada aos objetos, Java.

Índice

Abordagem inicial	Pág. 4
Inicialização	Pág. 5
Elaboração	Pág. 7
Modelo de Domínio	Pág. 7
Diagrama de <i>Use Case</i>	Pág. 8
Arquitetura	Pág. 9
Diagrama de classe	Pág. 10
Construção	Pág. 12
Diagrama de Máquina de Estado	Pág. 12
Diagramas de Sequência	Pág. 14
Base de dados do sistema	Pág. 17
Diagrama de instalação	Pág. 18
Desenvolvimento do software	Pág. 18
Arquitetura	Pág. 19
Conexão à base de dados	Pág. 19
Desenvolvimento gráfico	Pág. 20
Produção de código	Pág. 20
Funcionamento do sistema	Pág. 22
Transição	Pág. 30
Conclusão	Pág. 31
Apêndice	Pág. 33
Bibliografia	Pág. 44

Abordagem inicial

O desenvolvimento de um software deve ser planeado, com fases bem definidas e modeladas de modo a agilizar o desenvolvimento e evitar falhas sucessivas que ocorreriam caso adotássemos uma estratégia de *Code and Fix*. Deste modo podemos avaliar o progresso do sistema e prevenir erros futuros, correndo o risco de obter um produto mal desenvolvido ou que não respeite os requisitos definidos pelo cliente.

Tal como abordado nas aulas teóricas, o método mais correto para o desenvolvimento de um programa desta dimensão é o *Unified Process*. Este processo baseia o desenvolvimento do programa baseado em casos de uso (*use case*) e desenho da arquitetura do sistema. O seu processo apresenta quatro fases:

1. Inicialização
2. Elaboração
3. Construção
4. Transição

Cada fase pode contém várias iterações, dependendo da necessidade e complexidade do projeto.

O desenvolvimento baseado nos *use case* é bastante útil visto que o software é desenhado para satisfazer os requisitos em todas as fases do projeto, quer seja na conceção, quer na fase final de testes. O facto de assentar também na arquitetura do sistema é importante porque ajuda a obter um plano delineado do que desenvolver.

No caso específico deste projeto, o *Unified Process* revela-se o melhor método de desenvolvimento visto que não é um projeto de grande envergadura, no entanto é complexo o suficiente para ser necessário e útil a modelação deste. A utilização deste processo ganha ainda mais importância dados os seus requisitos bem definidos e a necessidade de criar um sistema capaz de os satisfazer eficaz e corretamente.

Inicialização

Nesta fase abordamos o que é que o enunciado nos pede. O que terá de ser desenvolvido, os utilizadores que irão interagir com o sistema e uma proposta de arquitetura do sistema.

Este projeto é de uma dimensão média pelo que podemos abordar esta fase do processo com uma única iteração num curto espaço de tempo. Para tal iremos iniciar o projeto com uma leitura minuciosa do enunciado e a escrita de uma visão geral sobre o que é necessário desenvolver para respeitar os requisitos do projeto:

Atores		Ações (Use Case)	Fases		
			1 ^a	2 ^a	3 ^a
Docente	Diretor de Curso	Login e Logout	X	X	X
		Registo de UC	X		
		Criação/edição/remoção de turnos	X	X	
		Registo de alunos	X		
		Alocar alunos em turnos	X	X	
	Diretor da UC	Mudar de fase	X	X	X
		Alocar alunos em turnos		X	X
		Trocar alunos entre turnos		X	X
		Remover aluno de um turno			X
		Definir capacidade de um turno		X	X
Estudante	Trabalhador Estudante	Marcar faltas (aluno perde turno se nº faltas > 25% aulas)			X
		Trocá-lo de turno especial (pode trocar sem precisar de outro aluno)		X	
		Trocá-lo de turno com outro aluno		X	
		Ver/aceitar/rejeitar troca		X	
		Consultar turnos	X	X	X

Podemos fazer então uma breve descrição dos intervenientes no sistema e respetivos casos de uso:

- **Docente:** unicamente responsável por lecionar os turnos e efetuar o registo de faltas dos alunos do seu turno.
- **Diretor de Curso:** utilizador responsável pela gestão do curso (registar UC's, alunos, turnos, etc.) na sua fase inicial. É um docente e pode também lecionar turnos de UC's. É o responsável por ditar as diferentes fases do sistema de gestão de turnos.
- **Diretor da UC:** utilizador responsável pela gestão da sua unidade curricular. Pode efetuar trocas de turnos, bem como alocar ou remover alunos e ditar a capacidade dos turnos da sua UC. É considerado um docente e tipicamente leciona alguns turnos da sua UC.
- **Estudante:** utilizador que se encontra inscrito num conjunto de turnos lecionados pelos docentes aos quais este assiste. Pode efetuar trocas de turnos com outros alunos livremente, desde que o outro aluno aceite. Pode também aceitar ou rejeitar pedidos de trocas de turnos de outros alunos e consultar os turnos onde se encontra inscrito.
- **Trabalhador estudante:** estudante que usufrui da vantagem de poder trocar de turnos livremente, sem necessitar de outro aluno para trocar, desde que o turno tenha capacidade suficiente. Para além desta vantagem, tem permissão para efetuar tudo o que um Estudante pode efetuar

Todos estes casos de uso devem ter assentes um conjunto de restrições para o bom funcionamento dos turnos. Restrições básicas como impedir que um aluno seja alocado num turno à mesma hora de um turno onde já está inscrito ou criar um turno prático à mesma hora de um turno teórico do mesmo ano. No entanto o enunciado pede-nos também flexibilidade no que toca à alocação dos alunos, visto que pode ser permitido um aluno ter duas teóricas à mesma hora, desde que sejam anos diferentes.

Elaboração

Chegamos agora à segunda fase do *Unified Process*. Nesta fase já registamos a maioria dos requisitos. No entanto, podem haver requisitos não referidos cuja adição seja importante ou até mesmo necessária.

Nesta fase é importante o papel da *UML*, na medida em que a modelação do Diagrama de *use case* e do Modelo de Domínio permitem a deteção de novos casos de uso e/ou especificações na arquitetura.

Modelo de Domínio

O Modelo de Domínio regista todas as entidades do Sistema através da análise dos requisitos e os relacionamentos entre elas. Estas entidades serão possíveis candidatas a classes do projeto.

Através da análise dos requisitos e dos *use case* recolhidos na fase anterior podemos elaborar o seguinte modelo numa fase inicial:

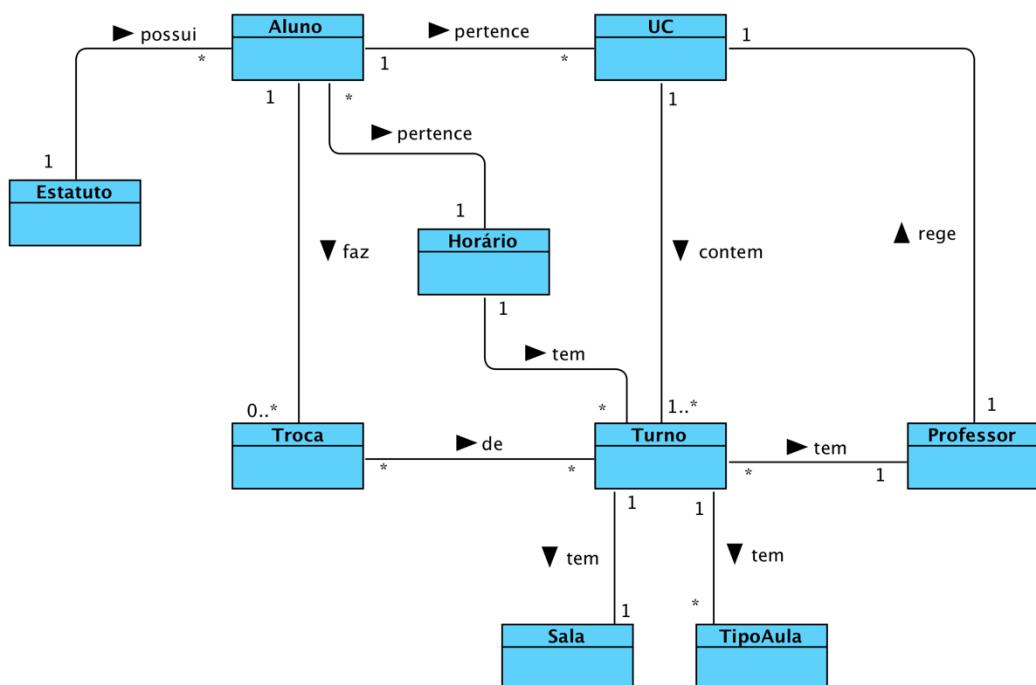


Fig. 1 – Primeira versão do modelo de domínio.

Este modelo apresenta o turno como uma entidade central, visto que de facto, o sistema irá basear-se na gestão dos mesmos. O aluno é definido por um estatuto (trabalhador-estudante ou normal) e tem atribuído a si um horário que contém vários turnos onde está inscrito. Sendo que pode efetuar trocas de turnos. Os turnos estão caracterizados pela sua UC, sala e tipo de aula (teórica, teórico-prática ou prática-laboratorial) e é lecionado por um professor que pode também rege uma UC.

Porém numa fase posterior acabamos por efetuar alterações, tendo obtido uma nova versão do modelo de domínio, mais ajustada para os requisitos do sistema:

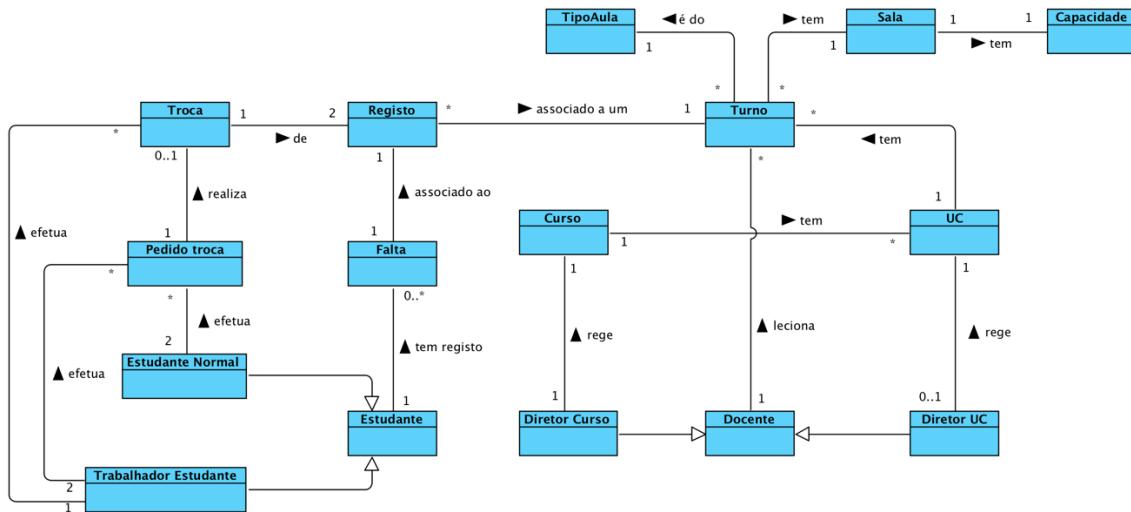


Fig. 2 – Versão final do modelo de domínio.

Este modelo é mais consistente e fiel aos requisitos. Foram adicionadas novas entidades (pedido troca, diretor UC, diretor curso) que melhoraram o modelo, auxiliando posteriormente no desenvolvimento do software e na escolha das respetivas classes. Este modelo também poderá ser importante para a criação de um modelo lógico para a base de dados relacional do projeto.

Diagrama de *Use Case*

Após termos os *use case* já definidos na primeira fase do ciclo de vida do *Unified Process* podemos avançar de imediato para a criação de um diagrama. Os *use case* definem a funcionalidade do sistema através dos seus requisitos e o diagrama facilita a interpretação dos mesmos, especificando todas as operações.

Os *use case* são o centro de todo o processo de desenvolvimento, no *Unified Process* por isso assumem um papel importantíssimo na modelação do projeto.

Assim, o diagrama obtido será o seguinte:

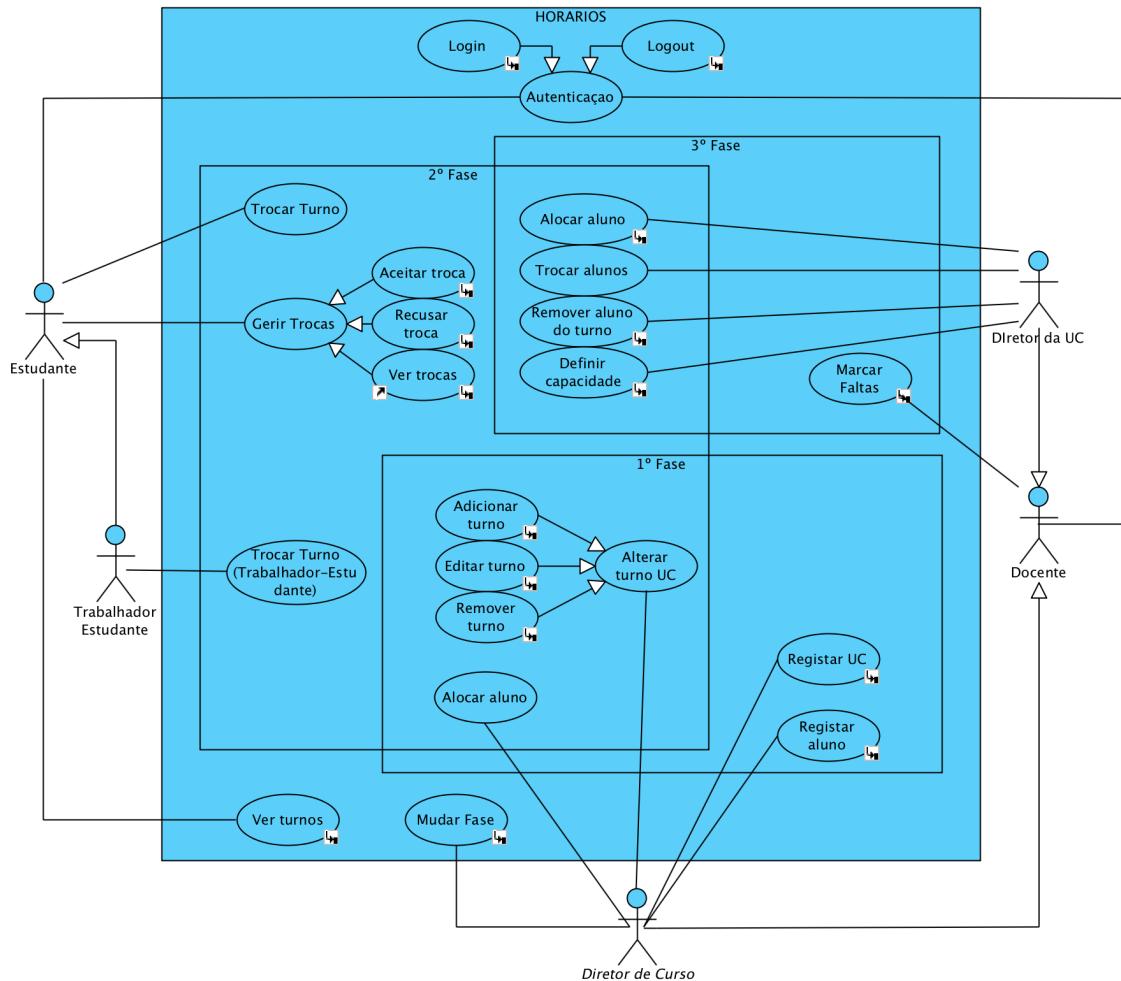


Fig. 3 – Versão final do diagrama de use case.

Arquitetura

Nesta fase é importante também definir a arquitetura da aplicação.

Optamos então por traçar uma arquitetura dividida por três camadas, cada uma delas dedicada ao seu propósito. Os dados irão estar aloados numa base de dados relacional (cujo desenvolvimento será abordado mais à frente) e serão consultados/ adicionados/ alterados/removidos pela camada de dados e os seus *Data Access Objects* (*DAO's*). Esta camada envia para a/recebe da camada de negócio, responsável pelo processamento dos dados (o código em si mesmo). A classe *Facade* é a responsável por ligar a camada de negócio e a de apresentação, no entanto, dada a pequena dimensão do projeto, grande parte dos métodos poderão ficar aloados nesta classe. A camada de apresentação é unicamente responsável por ler os dados inseridos pelos utilizadores e apresentar o *output* dos métodos, através do *Facade*.

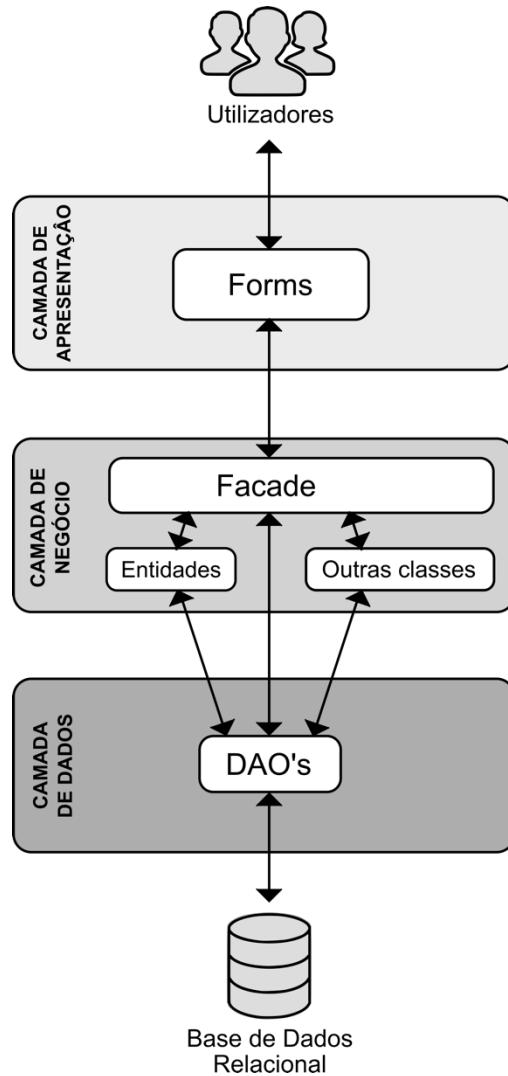
**Fig. 4 – Arquitetura do sistema.**

Diagrama de classe

O diagrama de classe tem bastante importância visto que orienta a estruturação do código, nomeadamente na construção das suas classes (podendo o código ser gerado automaticamente pelo *Visual Paradigm*).

Optamos por dividir o diagrama em duas pastas relativas às duas camadas onde a produção de código irá ter um papel mais importante: Negócio e Dados.

Grande parte das classes deriva do Modelo de Domínio, cujo objetivo é delinear os objetos que serão candidatos a classes. Alguns dos objetos declarados no Modelo de Domínio foram de facto anotados como classes no diagrama. Os objetos que não foram determinados como classes, foram definidos como atributos de classes.

Foi criada a classe *Facade*, que é a ligação entre a camada de apresentação e a camada de negócio, assim sendo, esta classe irá alojar todos os métodos relativos aos casos de uso (*use cases*).

Criamos as classes de acesso à base de dados, os *DAO's* e relacionamos com as suas respetivas classes na camada de negócio e com o *Facade*.

Segue-se em baixo o Diagrama de Classe construído:

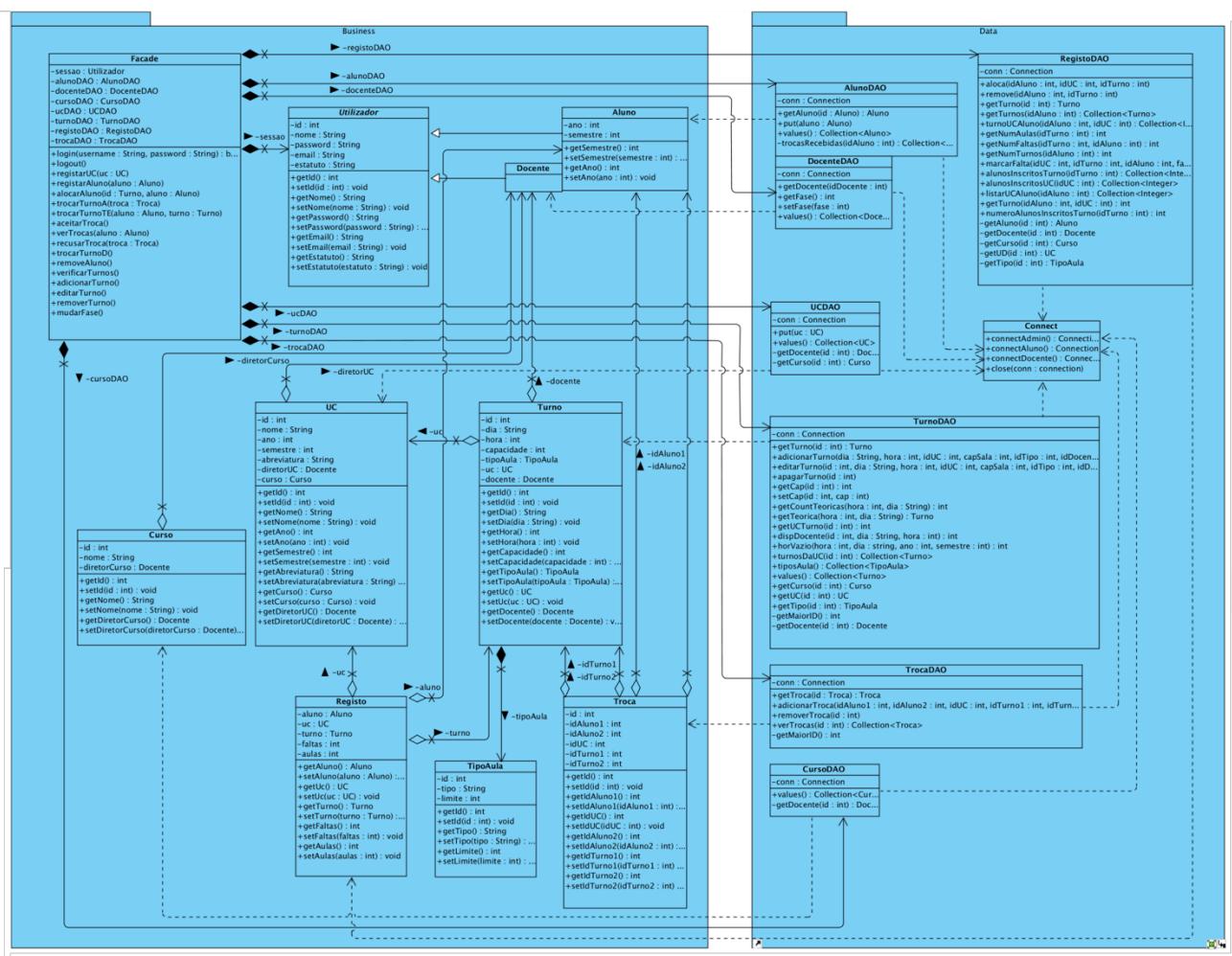


Fig. 5 – Diagrama de classe.

Construção

Esta é a fase mais demorada do projeto. Nesta fase é comum descrever as especificações dos *use case*, no entanto, é ainda mais importante o uso do UML no desenvolvimento de diagramas (diagramas de sequência, diagrama de estado de máquina e diagrama de instalação). É nesta fase também que se desenvolve o software.

O resultado final desta fase deverá ser um produto finalizado (software, design e respetivos modelos). O resultado poderá não ser uma versão final sem problemas. Eventuais problemas que possam aparecer serão resolvidos na seguinte fase, a Transição.

Diagramas de máquinas de estado

Os diagramas de máquinas de estado permitem modelar o comportamento global do sistema a desenvolver. São bastante úteis para o desenvolvimento da interface e comportamento da aplicação.

Como tal desenvolvemos os seguintes diagramas, referentes às três fases do ano letivo, definidas pelo diretor de curso:

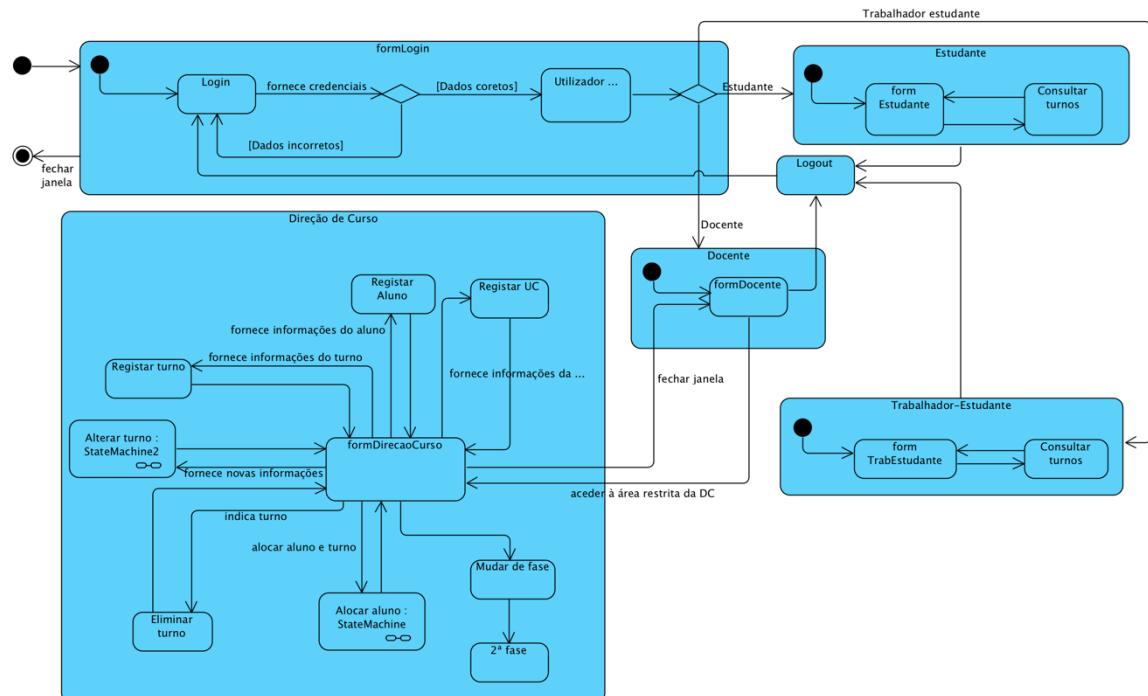
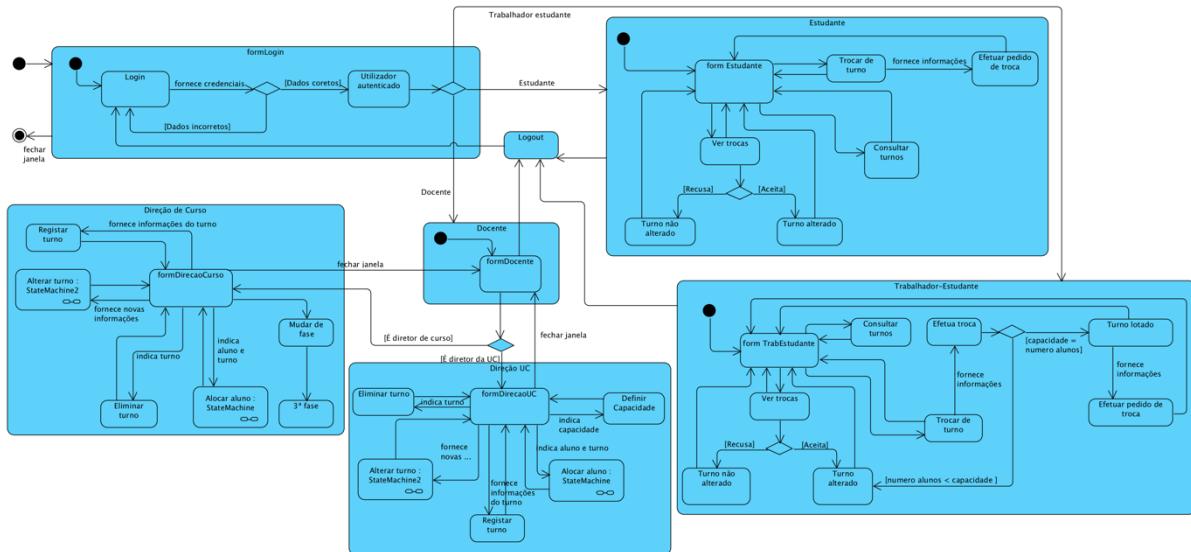
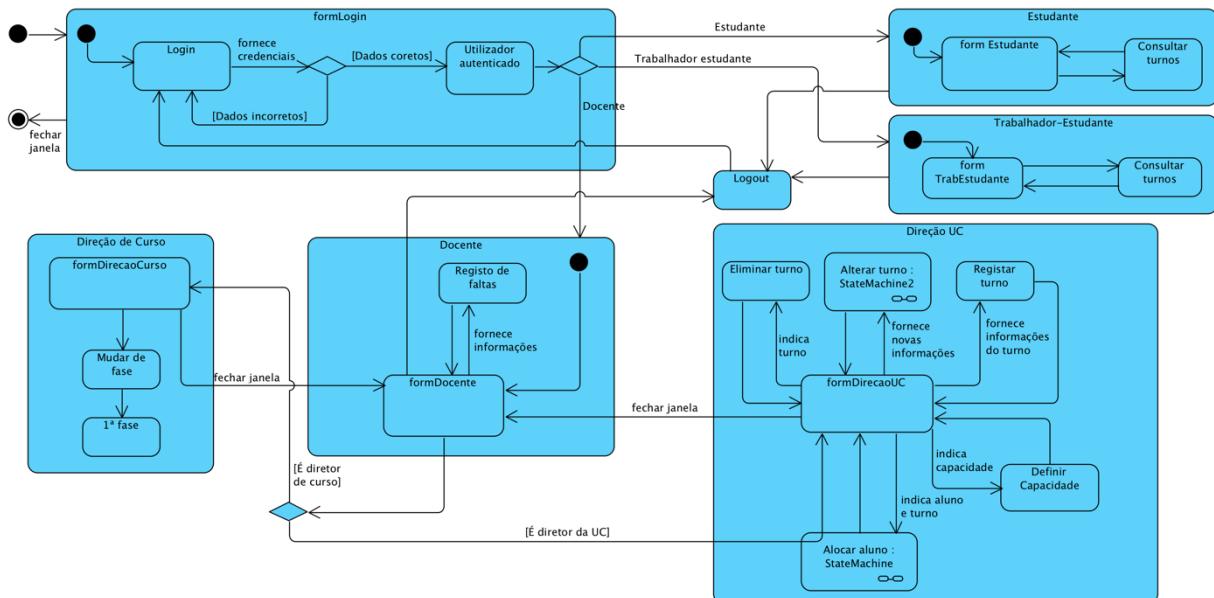


Fig. 6 – Diagrama de máquina de estado da 1^a fase.

Fig. 7 – Diagrama de máquina de estado da 2^a fase.Fig. 8 – Diagrama de máquina de estado da 3^a fase.

Não há uma maneira correta de justificar as decisões neste tipo de diagramas, visto que a nossa decisão é baseada naquilo que achamos que deve ser o comportamento do sistema, dado os requisitos. Comportamento esse especificado nos diagramas.

Foram também desenvolvidos dois subdiagramas de máquinas de estado, referentes ao estado “Alocar aluno” e “Alterar turno” que podem ser visualizados no Apêndice.

Diagramas de sequência

Os diagramas de sequência são utilizados para demonstrar a interação temporal entre objetos, através de mensagens, na execução de um *use case*.

Num processo onde os *use case* são um dos núcleos da modelação do desenvolvimento de software é importante desenvolver o fluxo da interação dos objetos em cada um dos *use case* de modo a facilitar posteriormente a fase de programar.

Para cada um dos *use case* do diagrama produzimos um diagrama de sequência sendo que iremos demonstrar e explicar de seguida alguns deles, no entanto poderão ser todos consultados no Apêndice.

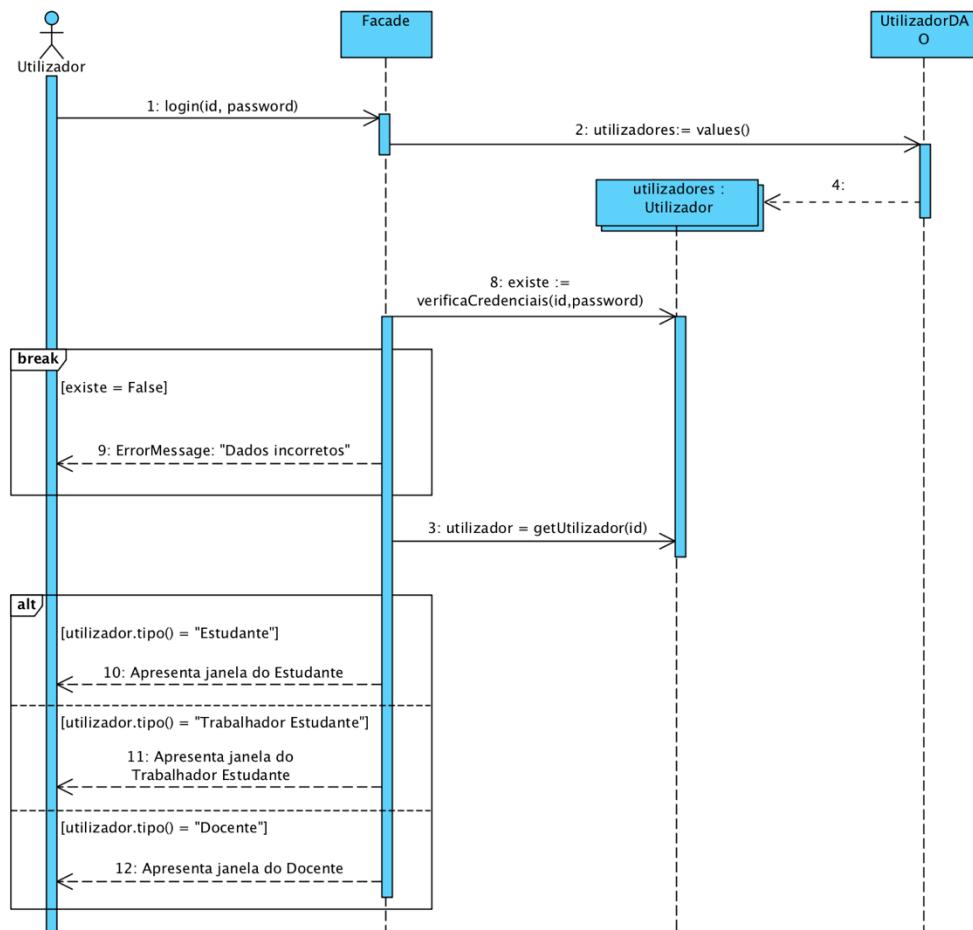


Fig. 9 – Diagrama de sequência “Login”.

Para o diagrama de sequência do *use case* “Login”, por exemplo, teremos um ator, que pode ser estudante ou docente, e que irá introduzir as suas credenciais. Estas serão enviadas para a camada de negócio, no *Facade*. Para verificar as credenciais iremos obter uma lista de utilizadores onde iremos verificar se os dados introduzidos existem nessa lista. A sequência termina imediatamente caso as credenciais não existam na lista de utilizadores. Caso estejam corretas, iremos abrir a janela correspondente ao utilizador introduzido (docente, estudante ou trabalhador-estudante).

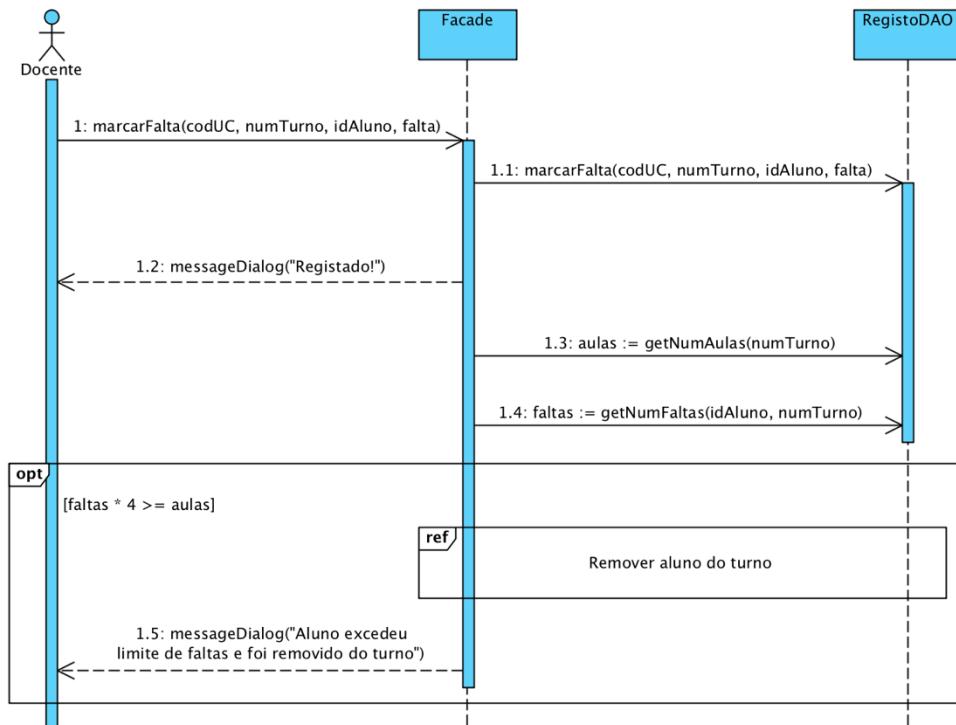


Fig. 10 – Diagrama de sequência “Marcar faltas”.

Este diagrama de sequência é referente ao *use case* de marcar faltas. Este *use case* será o responsável por satisfazer o requisito do sistema ser capaz de efetuar um registo de faltas e remover alunos do turno caso tenham mais de 25% de faltas ao turno.

Assim, o docente irá atribuir uma falta do seu turno, a um determinado aluno onde a variável “falta” será um binário onde 0 representa estar presente e 1 representa faltar. O *Facade* da camada de negócio recebe essa informação e regista a presença/falta na base de dados através do *RegistroDAO* e avisa o utilizador do sucesso da operação.

Após registar a presença/falta do aluno é verificado se ele ultrapassou o limite de faltas. Obtemos o número de aulas e de faltas do aluno nesse turno e verificamos a condição ($nFaltas * 4 \geq nAulas$). Caso a condição se verifique referenciamos o diagrama de sequência do *use case* “Remover aluno do turno” e avisamos o docente de que o aluno ultrapassou o limite de faltas e por isso foi removido do turno.

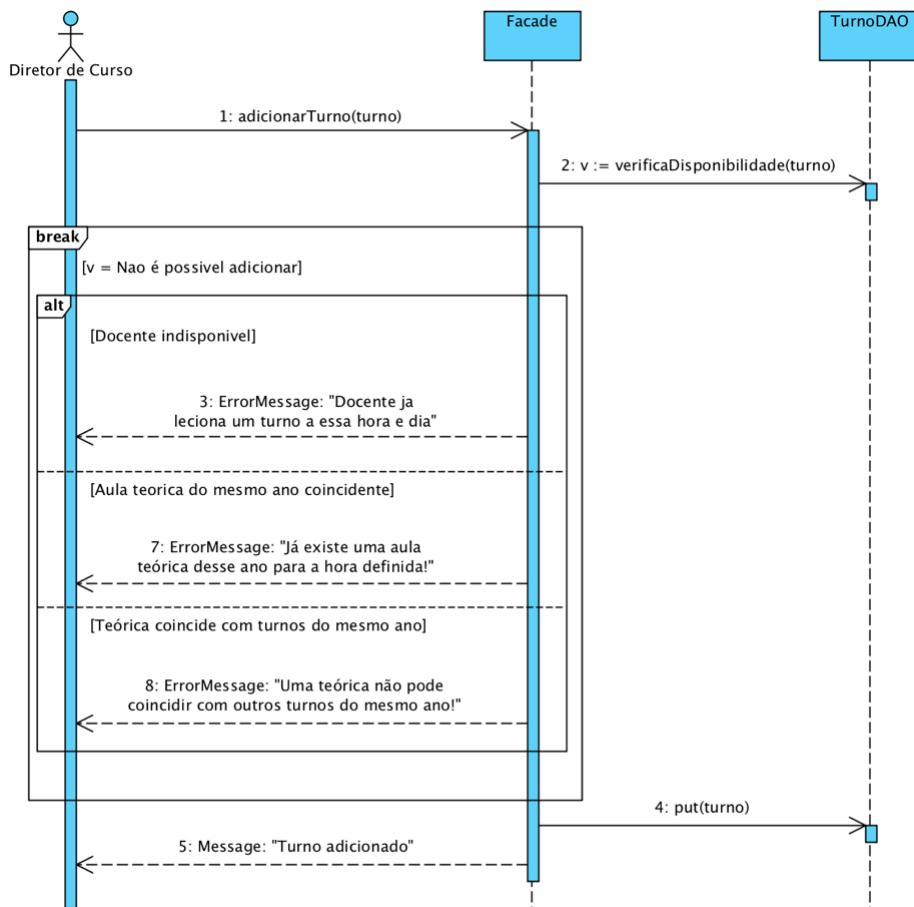


Fig. 11 – Diagrama de sequência “Adicionar turno”.

Este é o diagrama de sequência referente ao *use case* do diretor de curso adicionar um novo turno. O ator (diretor de curso) insere os dados do novo turno que serão entregues ao *Facade*.

Aqui é verificada a disponibilidade da adição do turno tendo em conta as informações do novo turno (dia, hora, docente, etc.). A sequência termina se o método de verificar disponibilidade devolver uma resposta negativa. Essa resposta negativa pode derivar de três restrições: o docente atribuído já leciona um turno na hora definida; já existe uma aula teórica desse ano para a hora definida, no caso de adicionarmos uma aula prática, ou, no caso de adicionarmos uma aula teórica, a hora definida coincidir com outros turnos do mesmo ano.

Caso nenhum destes casos se verifique, o turno é adicionado com sucesso à base de dados e é retribuída uma mensagem para o utilizador a confirmar a inserção.

Base de dados do sistema

Tendo em conta a modelação produzida até ao momento, é importante iniciar a construção de uma base de dados relacional que suporte o sistema. Essa base de dados deverá ser robusta e capaz de responder às *queries* derivadas dos requisitos do sistema.

Para o desenvolvimento da base de dados usamos o sistema gestor de base de dados *MySQL*, visto que já estamos familiarizados com o sistema nas aulas da unidade curricular de Bases de Dados.

Assim sendo, foi produzido o seguinte modelo lógico:

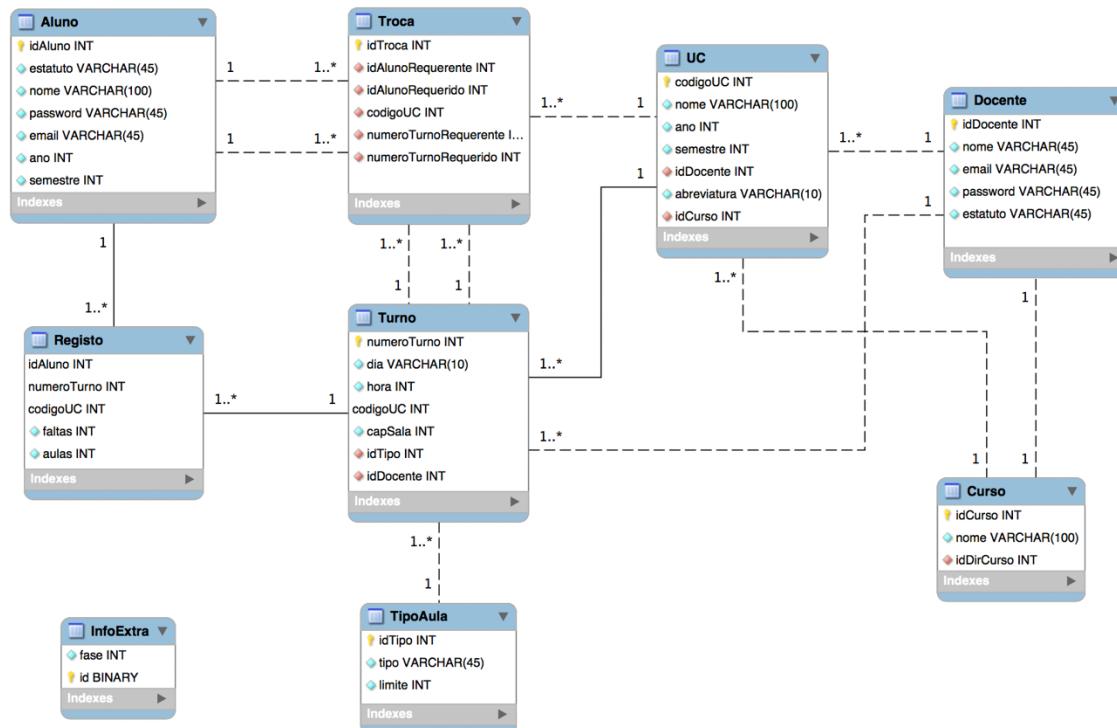


Fig. 12 – Modelo lógico da base de dados.

Este modelo foi produzido tendo em conta os requisitos iniciais, adaptando-se corretamente ao diagrama de classe e restante modelação produzida.

Todas as entidades do modelo são facilmente relacionáveis com o modelo de domínio e diagrama de classe produzido, no entanto queremos abordar duas entidades que podem necessitar de explicação: “InfoExtra” e “Registro”.

A entidade “Registro” foi criada para relacionar as inscrições dos alunos nos seus diferentes turnos. “Turno” e “Aluno” são entidades distintas e não relacionadas que descrevem unicamente os turnos das várias UC’s e os alunos existentes. A inscrição do aluno num determinado turno será registada na entidade “Registro”, onde iremos alojar o ID do aluno e o número do turno onde está inscrito. Aproveitamos também e usamos esta entidade para alojar o registo de faltas do aluno no turno.

A entidade “InfoExtra” não está relacionada com a gestão dos horários. É uma entidade com uma única linha (ID=1) que existe unicamente para guardar a fase definida pelo diretor de curso.

Após confirmar que o modelo era o adequado para as necessidades, produzimos o modelo físico através de um comando existente no *MySQL Workbench*. O script gerado automaticamente encontra-se no ficheiro “Script.sql”.

De modo a podermos verificar o bom funcionamento do software produzimos também um povoamento da base de dados com alunos e docentes registados e com as UC's e turnos que docentes irão lecionar/reger, tendo sempre o cuidado de organizar os turnos num horário correto (sem sobreposições de teóricas, horários de docentes, etc.). Este povoamento pode ser consultado no ficheiro “Population.sql”.

Podemos também definir como um mecanismo de segurança a criação de utilizadores. Neste caso podemos definir três utilizadores distintos: Administrador, Docente e Aluno. Cada utilizador tem as permissões adequadas aos casos de uso definidos, sendo que o Administrador tem controlo total sobre a base de dados. A criação dos utilizadores está disponível para consulta no ficheiro “Users.sql”.

Diagrama de instalação

O diagrama de instalação auxilia na leitura da arquitetura física do sistema e dos componentes software do sistema que compõem as suas camadas.

O diagrama de instalação construído para o nosso projeto foi o seguinte:

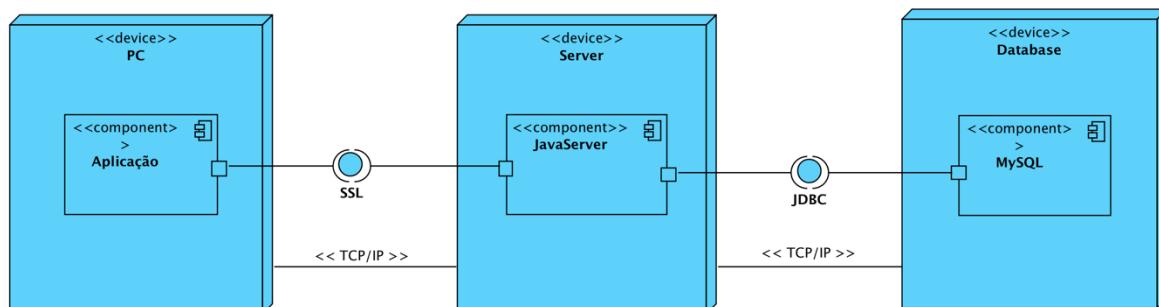


Fig. 13 – Modelo lógico da base de dados.

Desenvolvimento do software

Para começar o desenvolvimento da aplicação é importante escolher um IDE (*Integrated Development Environment*) que seja adequado à realização do projeto e onde o ambiente de desenvolvimento nos seja familiar. Por isso escolhemos o *NetBeans*, visto que já o tínhamos usado anteriormente na UC.

1. Arquitetura

Seguimos a estrutura definida na fase anterior do *Unified Process*. Assim sendo, iremos ter o código estruturado por três camadas:

- Camada de apresentação: é a camada em contacto com o utilizador, que aloja os formulários (interface do programa) e lê e escreve informações.
- Camada de negócio: é a camada que recebe as informações das suas camadas adjacentes e a processa.
- Camada de dados: camada responsável por ler e registar informação da base de dados e a enviar/receber da sua camada superior.

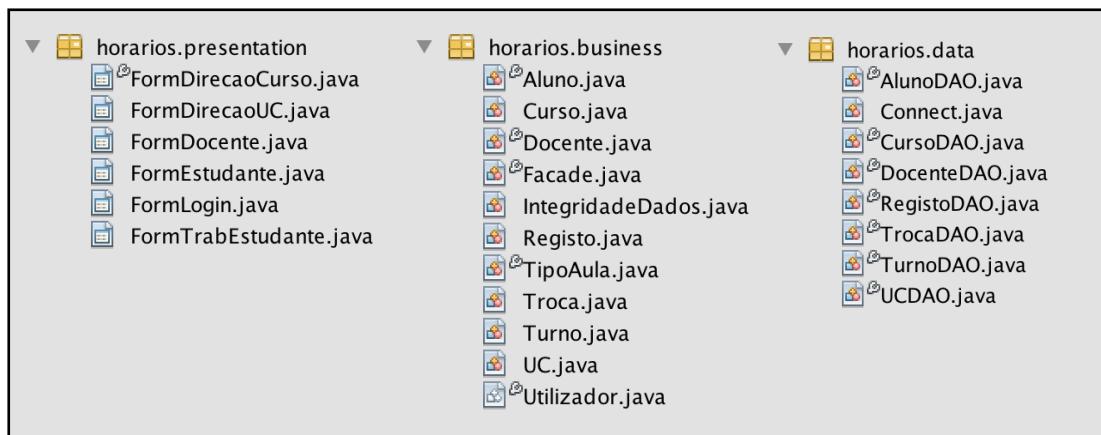


Fig. 14 – Organização das camadas.

2. Conexão à base de dados

Primeiro temos de conectar o IDE à base de dados, usando o Java Database Connectivity (JDBC). O NetBeans permite efetuar facilmente uma ligação ao *localhost* do *MySQL*. No entanto temos de criar uma classe “Connect” que irá ligar o software em si à base de dados.

Para respeitar os mecanismos de segurança definidos na criação da base de dados, criamos três conexões possíveis à base de dados, relativos aos três utilizadores definidos.

```
/** Estabelece ligação à base de dados ...6 lines */
public static Connection connectAdmin() throws SQLException, ClassNotFoundException {
    Class.forName("com.mysql.jdbc.Driver");
    //cliente deve fechar conexão!
    return DriverManager.getConnection("jdbc:mysql://" + URL + "/" + DB + "?user=" + USERNAMEADMIN + "&password=" + PASSWORDADMIN);
}
public static Connection connectAluno() throws SQLException, ClassNotFoundException {
    Class.forName("com.mysql.jdbc.Driver");
    //cliente deve fechar conexão!
    return DriverManager.getConnection("jdbc:mysql://" + URL + "/" + DB + "?user=" + USERNAMEALUNO + "&password=" + PASSWORDALUNO);
}
public static Connection connectDocente() throws SQLException, ClassNotFoundException {
    Class.forName("com.mysql.jdbc.Driver");
    //cliente deve fechar conexão!
    return DriverManager.getConnection("jdbc:mysql://" + URL + "/" + DB + "?user=" + USERNAMEDOCENTE + "&password=" + PASSWORDDOCENTE);
}
```

Fig. 15 – Método que estabelece a ligação à base de dados.

3. Desenvolvimento gráfico

Estabelecida a ligação à base de dados é necessário construir o ambiente gráfico da aplicação baseado na proposta de interface desenhada na primeira fase deste projeto.

Optamos por dar importância às *Combo Boxes* ao invés de *Text Fields*, deste modo conseguimos obter uma integridade de dados maior, visto que o utilizador será obrigado a selecionar um dos dados fornecidos pela *Combo* e não poderá por valores que não existem ou que não fazem sentido.

Os formulários criados foram sofrendo modificações ao longo do desenvolvimento do código, de modo a se adaptar a necessidade que fossem aparecendo, pelo que iremos demonstrar a interface.

4. Produção de código

Começamos por criar no *Facade* da camada de negócio os métodos referentes aos *use case* existentes. Vamos explicar, como exemplo, o método de registar a falta de um aluno.

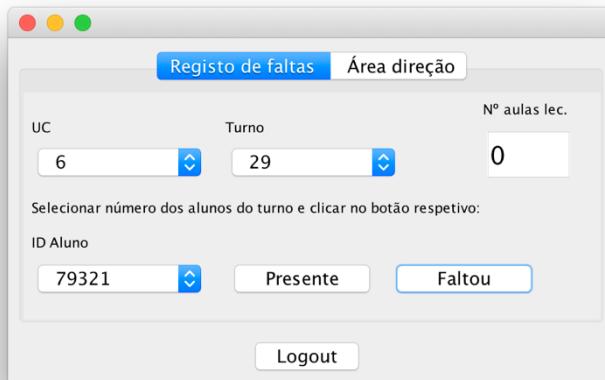


Fig. 16 – Formulário do registo de faltas.

```
private void buttonFaltouADActionPerformed(java.awt.event.ActionEvent evt) {
    int codUC = Integer.parseInt(comboCodAD.getSelectedItem().toString());
    int numTurno = Integer.parseInt(comboTurnoAD.getSelectedItem().toString());
    int idAluno = Integer.parseInt(comboAlunoAD.getSelectedItem().toString());

    facade.marcarFalta(codUC, numTurno, idAluno, 1);
    comboAlunoAD.removeItemAt(comboAlunoAD.getSelectedIndex());
}
```

Fig. 17 – Botão “faltou” acionado.

Neste caso, quando o botão “faltou” é acionado é lida a informação das *Combo Boxes*, é efetuado o método “marcarFalta” do *Facade* e removemos esse aluno da *Combo* de alunos do formulário.

No *Facade*, o método “marcarFalta” recebe os dados fornecidos no formulário e um inteiro “falta” que é 1 caso o aluno tenha faltado, ou 0 caso o aluno tenha estado presente. Neste caso, o botão acionado é o de falta, logo o método recebe 1.

Invocamos o método “marcarFalta” do *RegistoDAO* que irá registar os dados na base de dados e vamos verificar se o aluno excedeu o limite de faltas ao turno.

Obtemos o número de aulas lecionadas do turno e de faltas do aluno ao turno e efetuamos a condição de verificação. Caso tenha excedido o limite ($faltas * 4 \geq aulas$) invocamos o método “removeAluno”, presente também no *Facade* visto que se trata de um *use case*, e enviamos uma mensagem para o utilizador a informar que o aluno foi removido.

```
public void marcarFalta(int codUC, int numTurno, int idAluno, int falta){
    registoDAO.marcarFalta(codUC, numTurno, idAluno, falta);

    int aulas = registoDAO.getNumAulas(numTurno);
    int faltas = registoDAO.getNumFaltas(numTurno, idAluno);

    if(faltas*4 >= aulas) {
        try {
            removeAluno(idAluno, numTurno);
        } catch (ClassNotFoundException | SQLException ex) {
            Logger.getLogger(FormDocente.class.getName()).log(Level.SEVERE, null, ex);
        }
        ImageIcon icon = new ImageIcon(getClass().getClassLoader().getResource("resources/warning.png"));
        JOptionPane.showMessageDialog(null, "Aluno excedeu limite de faltas"
            + "e foi removido do turno!", "Aviso", JOptionPane.INFORMATION_MESSAGE, icon);
    }
}
```

Fig. 18 – Método “marcarFalta” do *Facade*.

Vamos explicar agora o funcionamento da “marcarFalta” do *RegistoDAO*.

Este método recebe as mesmas variáveis que a “marcarFalta” do *Facade*. Inicialmente estabelece uma ligação à base de dados através da ligação com o utilizador “docente”, para respeitar as suas permissões. É criada a *query* onde adicionamos a falta do aluno no turno na tabela “Registo” e executamos a *query* (*executeUpdate*). Apanhamos as devidas exceções, caso existam, e fechamos a conexão.

```
public void marcarFalta(int codUC, int numTurno, int idAluno, int falta){
    try {
        conn = Connect.connectDocente();
        PreparedStatement stm = conn.prepareStatement("UPDATE Registo SET aulas=aulas+1, faltas=faltas+(?)"
            + "WHERE numeroTurno=(?) AND idAluno=(?)" + "AND codigoUC=(?)", Statement.RETURN_GENERATED_KEYS);
        stm.setInt(1, falta);
        stm.setInt(2, numTurno);
        stm.setInt(3, idAluno);
        stm.setInt(4, codUC);
        stm.executeUpdate();
        ResultSet rs = stm.getGeneratedKeys();
    } catch (ClassNotFoundException | SQLException e) {
        throw new NullPointerException(e.getMessage());
    } finally {
        Connect.close(conn);
    }
}
```

Fig. 19 – Método “marcarFalta” do *RegistoDAO*.

Este é o funcionamento típico de um método do *Facade*.

Todos os métodos definidos no *Facade* (*use case*) foram desenvolvidos seguindo esta linha de produção de código. Temos ainda alguns métodos na camada de apresentação cujo objetivo é manter informação atualizada e adequada nas *Combo Boxes*, por exemplo: ao selecionar uma UC apresentar numa *Combo* apenas os turnos dessa UC.

Funcionamento do sistema

O funcionamento do sistema é facilmente perceptível através do diagrama de máquinas de estado que demonstram o funcionamento do sistema e as operações que podem ser efetuadas em cada estado da aplicação.

Para ter a aplicação disponível deve usar o *MySQL Workbench* e executar os três *scripts* presentes na pasta “Database” pela seguinte ordem: *script.sql* -> *users.sql* -> *population.sql*. A base de dados já está disponível e povoada. Para executar agora a aplicação, basta abrir o ficheiro *Horarios.jar* presente na diretoria “Código/dist/”.

Qualquer utilizador que inicie a aplicação depara-se com um sistema de autenticação. Para entrar no programa tem de introduzir as suas credenciais e será reencaminhado para o formulário respetivo: estudante, trabalhador-estudante ou docente.

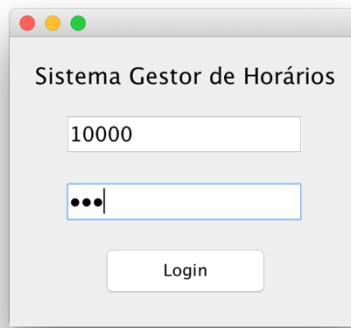


Fig. 20 – Login.

Neste caso, o ID 10000 é o do diretor de curso, logo irá abrir a janela do docente, onde este poderá registar as presenças do turno. Para tal, tem de selecionar a UC, o turno e o aluno para registrar a presença ou a falta. O aluno é removido automaticamente do turno caso exceda o limite de faltas. No canto superior direito é dada a informação do número de aulas lecionadas a esse turno, até ao momento.



Fig. 21 – Docente.

No outro separador da janela do docente temos os botões das áreas de acesso restrito aos diretores de UC e curso. O botão só deixa abrir a janela dos diretores se estes tiverem registados de facto como diretores de UC ou curso. Neste caso a sessão iniciada é a do diretor de curso, logo teremos acesso à janela da direção de curso.

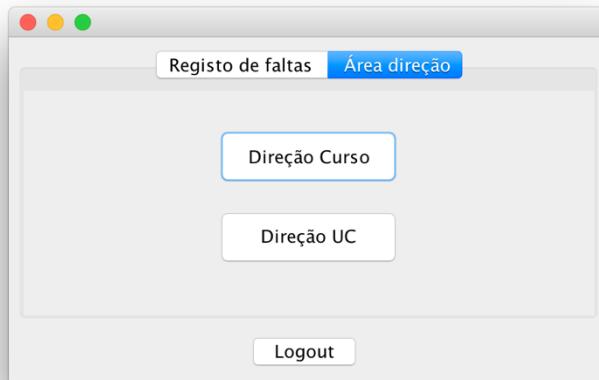


Fig. 22 – Docente.



Fig. 23 – Diretor de curso.



Fig. 24 – Diretor de curso.

Neste caso ao abrir a janela do diretor de curso, grande parte dos separadores encontram-se bloqueados isto devido às restrições das fases do ano letivo, definidas nos requisitos. No entanto, se mudarmos da 3^a para a 1^a fase, esses separadores desbloqueiam-se. Aqui é permitido registrar UC's, alunos, alocar alunos em turnos e adicionar, editar e remover turnos.

The screenshot shows the 'Turnos' tab selected in the 'Registrar UC' window. The interface includes fields for 'Turno' (with a dropdown menu showing '15'), 'Dia' (set to 'Quarta'), 'Hora' (set to '11'), and an 'Eliminar' button. Below this, there are fields for 'Docente' (set to '10890'), 'Capacidade' (set to '20'), and 'Tipo' (set to '3'). An 'Guardar' button is also present. At the bottom, there are fields for 'Código UC' (dropdown menu showing '2'), 'Dia', 'Hora', and 'Tipo' (dropdown menu showing '3'). Below these, there are fields for 'Docente' (dropdown menu showing '10890') and 'Capacidade'. A 'Registrar novo' button is located at the bottom right.

Fig. 25 – Diretor de curso.

O separador dos Turnos, por exemplo, na parte superior permite editar as informações de um turno (basta selecionar um turno da *combo* e editar as informações na caixa) ou eliminar esse turno selecionado. Na parte inferior é-nos permitido registrar um novo turno.

Caso um diretor de uma UC quisesse aceder à sua área restrita encontraria esta janela. Nela é capaz de alocar um aluno num turno, alterar a capacidade ou trocar alunos entre turnos.



Fig. 26 – Diretor de UC.

Esta é a interface para os docentes.

Caso iniciássemos sessão com um estudante, esta seria a janela que apareceria:

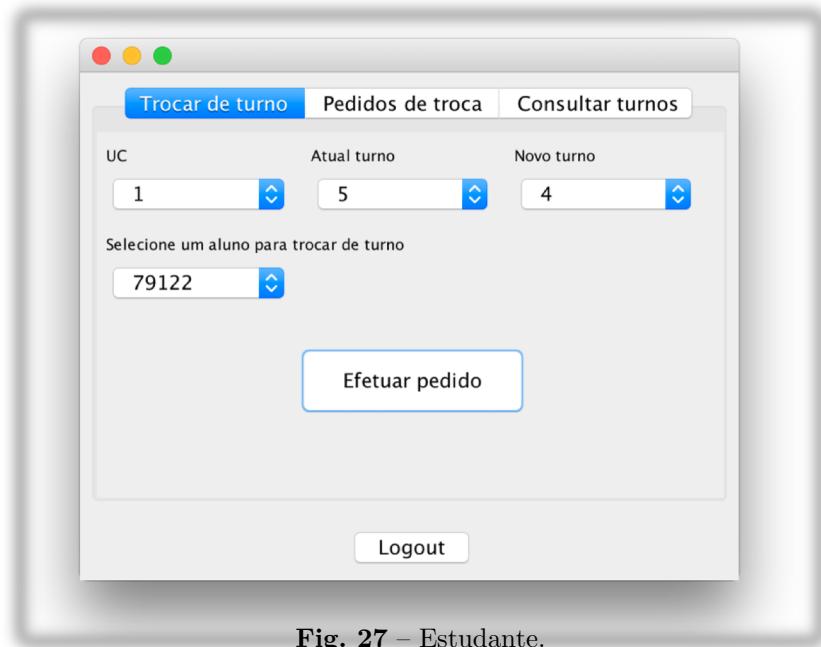


Fig. 27 – Estudante.

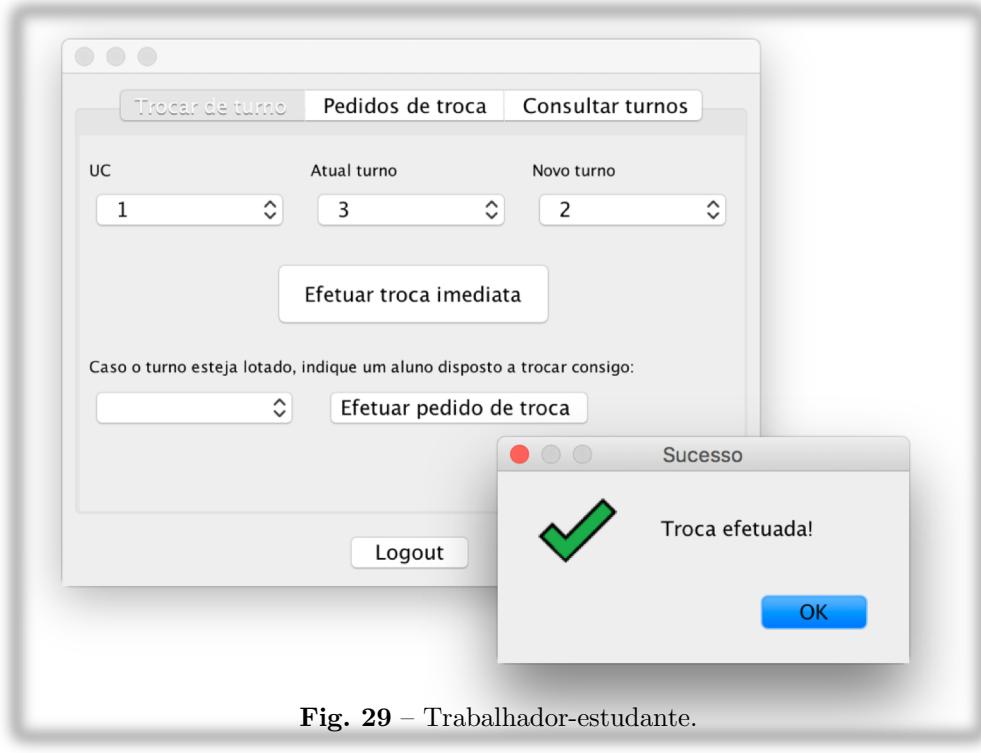
Aqui o estudante tem três separadores, onde pode efetuar pedidos de troca de turno, visualizar e aceitar/recusar pedidos de troca de outros alunos e consultar os turnos onde está inscrito.

UC	Tipo	Turno	% Faltas	Dia	Hora
BD	Prática	12	0	Segunda	9
SD	Prática	28	0	Terça	9
SD	Teórica	26	0	Sexta	9
RC	Teórica	21	0	Terça	14
MNONL	Teórica	16	0	Quinta	11
MDIO	Teórico-...	5	0	Quarta	11
DSS	Prática	8	0	Segunda	14
RC	Prática	23	0	Segunda	11
CD	Teórica	36	0	Quinta	14
CD	Teórico-...	38	0	Quarta	9
MDIO	Teórica	1	0	Terça	11
DSS	Teórica	6	0	Quinta	9
MNONL	Teórico-...	18	0	Quinta	16
BD	Teórica	11	0	Sexta	11

Logout

Fig. 28 – Estudante.

Caso iniciássemos sessão com um trabalhador-estudante, a janela seria idêntica à do estudante em regime normal, exceto uns detalhes do separador “Trocar de turno”, visto que o trabalhador-estudante pode trocar de turno sem precisar de outro estudante.

**Fig. 29 – Trabalhador-estudante.**

Neste caso, o trabalhador-estudante efetuou com sucesso uma troca do turno 3 para o turno 2, da UC com o ID 1 (MDIO).

No entanto, pode efetuar uma troca com um aluno também:

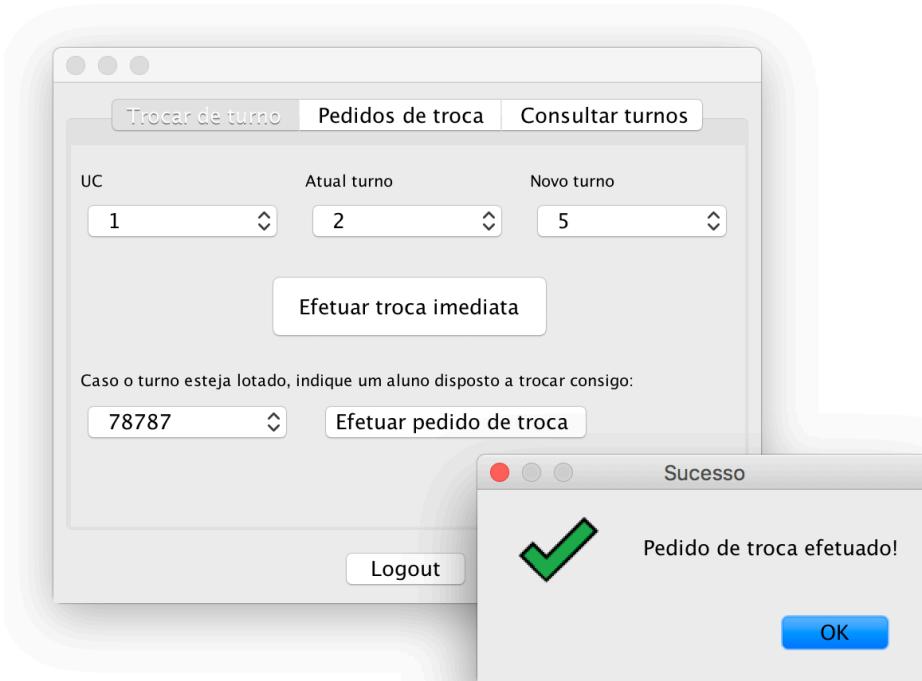


Fig. 30 – Trabalhador-estudante.

O trabalhador-estudante enviou um pedido de troca para o aluno com o ID 78787. Assim, se o aluno com o ID 78787 iniciar sessão e for ao separador “Pedidos de troca” irá ter lá este pedido de troca:

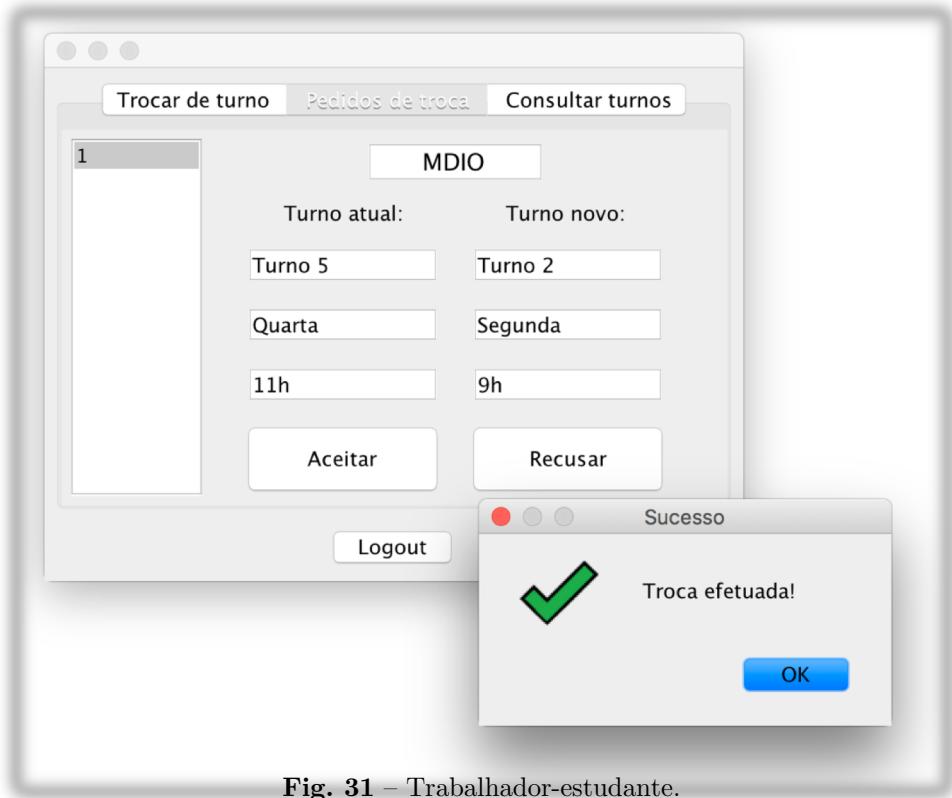


Fig. 31 – Trabalhador-estudante.

Como seria de esperar, a troca aparece no separador. O aluno aceitou a troca. Se for agora visualizar os turnos onde se encontra inscrito, a informação encontra-se atualizada, encontrando-se agora no turno 2 da UC MDIO.

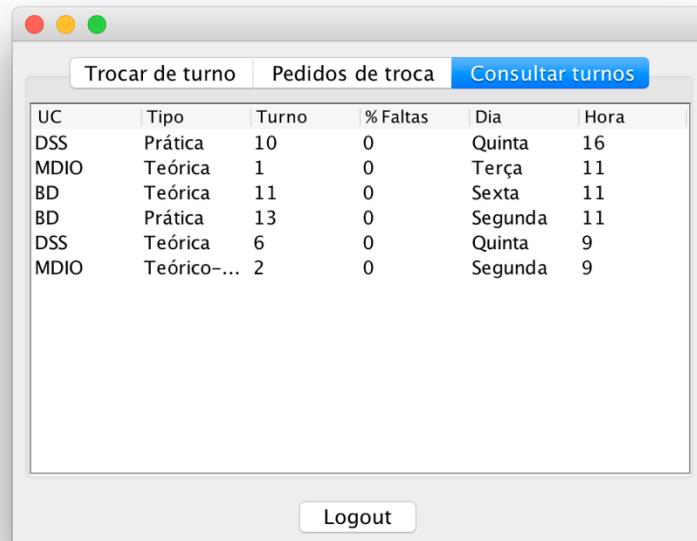


Fig. 32 – Trabalhador-estudante.

A interface avisa também quando é impossível fazer uma troca, ou quando é impossível registar um turno devido a variadas restrições (definidas nos requisitos do problema):



Fig. 33 – Avisos.

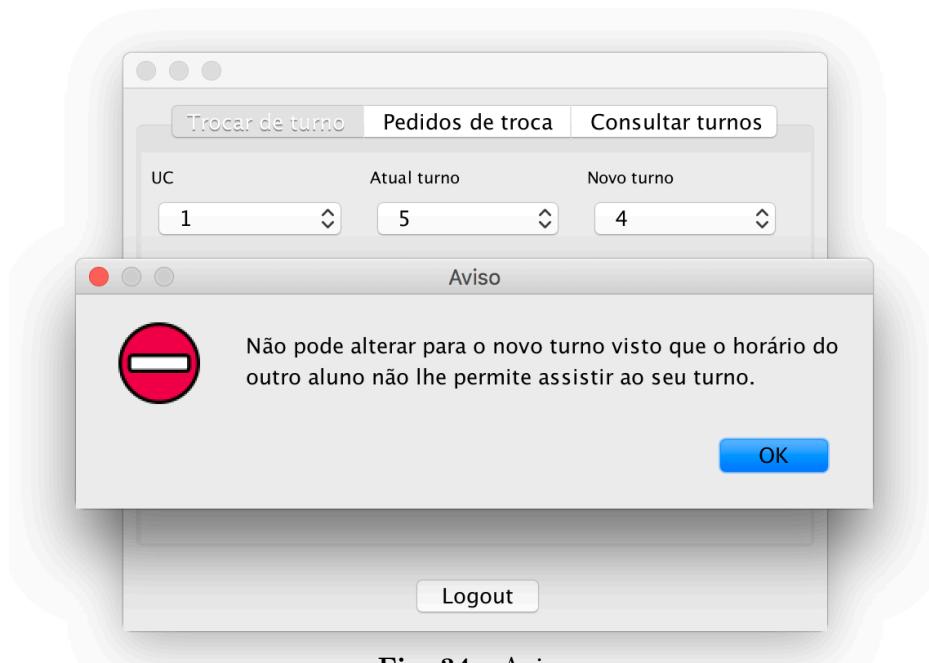


Fig. 34 – Avisos.

Transição

Todo o código foi submetido a vários testes, sendo que não foi encontrado nenhum erro visto que todos os erros encontrados foram sendo corrigidos ao longo da construção do projeto. Assim sendo, podemos confirmar que temos um sistema 100% funcional.

Conclusão

Concluída a última fase do *Unified Process*, podemos considerar finalizada a realização deste projeto e analisar os aspectos positivos e negativos que retiramos da realização do trabalho.

A abordagem à realização do trabalho foi no nosso ponto de vista a mais adequada pois é aquela que se adequa melhor às dimensões do projeto. Este método (*Unified Process*) baseia-se na estruturação arquitetural do sistema e nos casos de uso (*use case*) definidos pelos requisitos iniciais.

Na primeira fase, Inicialização, recolhemos os requisitos fornecidos pelo enunciado e elaboramos uma tabela onde registamos e organizamos os mesmos.

Na seguinte fase, Elaboração, desenvolvemos o Modelo de Domínio que serviu como elaboração dos objetos candidatos a classe e definição inicial da estruturação do sistema. Desenvolvemos também um Diagrama de *Use Case*, aplicando os casos de uso recolhidos na fase anterior. Este diagrama será a base do desenvolvimento de toda a aplicação, uma vez que esta vai ser construída à volta dos requisitos. Foi desenhada uma arquitetura estrutural do sistema e ainda um Diagrama de classe, com as classes definidas, métodos e atributos. Este diagrama pode ser gerar código automaticamente, servindo assim como ponto inicial para o desenvolvimento da aplicação.

Na terceira fase, Construção, modelamos mais diagramas essenciais para o desenvolvimento do software, construímos a base de dados que irá suportar o sistema e iniciamos a produção de código do mesmo.

Na quarta e última fase, Transição, verificamos a ausência de erros do resultado final, obtendo um produto 100% finalizado.

Com este trabalho podemos concluir que a modelação é uma excelente ferramenta no auxílio do desenvolvimento de software de média a grande dimensão visto que permite haver um faseamento do desenvolvimento, o que facilita a correção de erros em qualquer momento, sem implicar grandes custos. A modelação também permite a deteção de erros numa fase inicial, facilitando a sua correção.

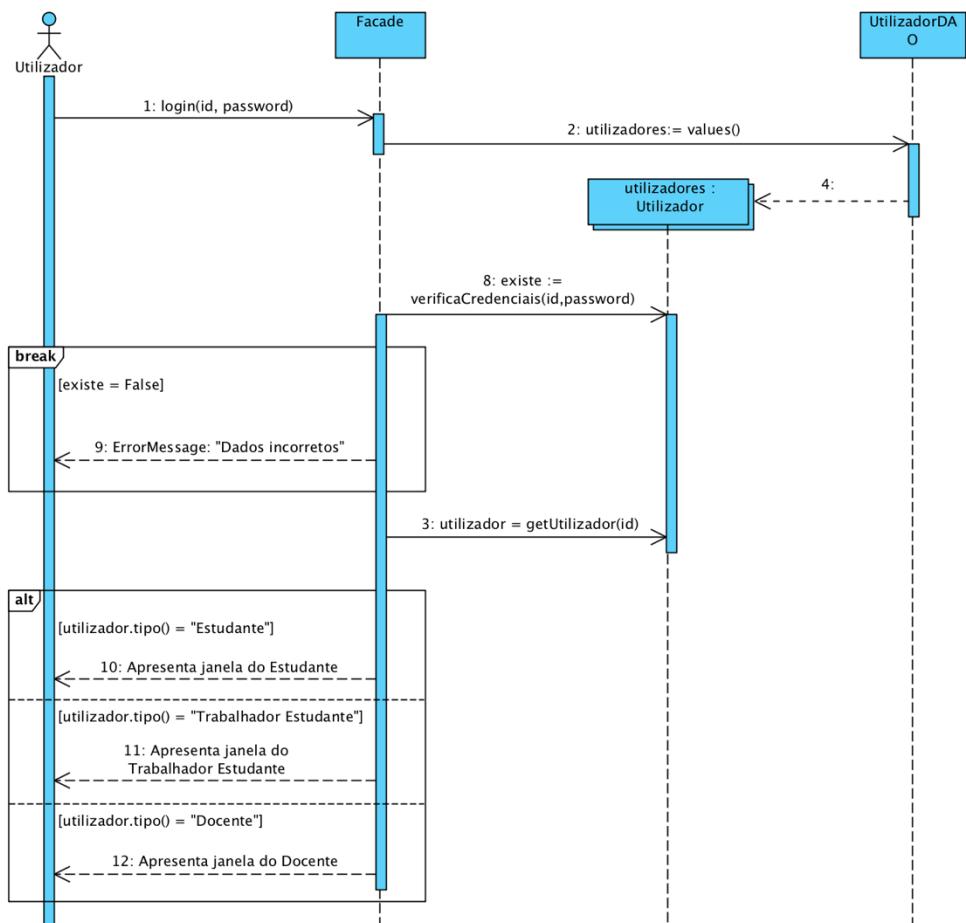
O nosso desempenho neste trabalho é positivo, no nosso ponto de vista, uma vez que conseguimos obter um resultado final completamente modelado, 100% funcional e de acordo com os requisitos, embora não faça a alocação inicial automática dos turnos.

Consideramos a nossa maior dificuldade, a criação dos diagramas e modelos, visto que há maneiras diferentes de modelar e todas podem estar corretas, o que nos atrapalhou um bocado e nos levou a fazer algumas modificações recorrentes em diagramas (especialmente os de sequência).

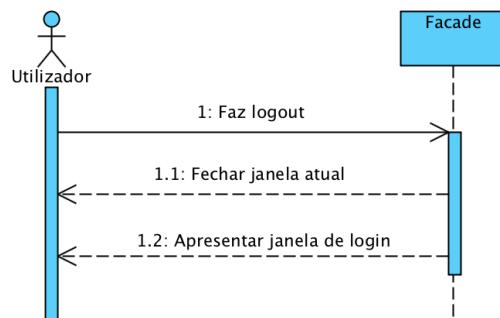
Apêndice

Diagramas de sequência

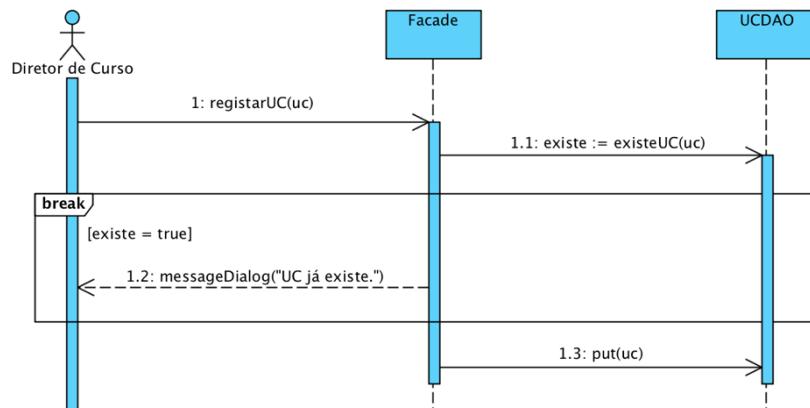
1. Login



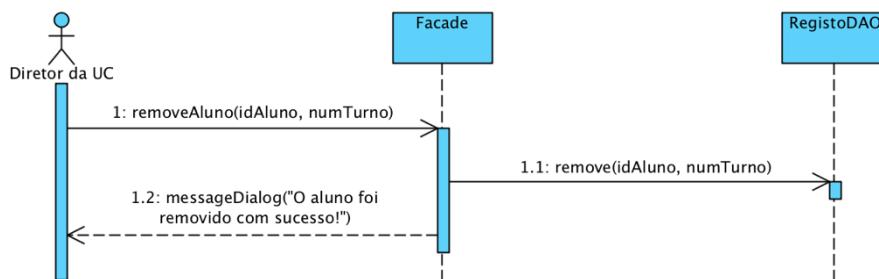
2. Logout



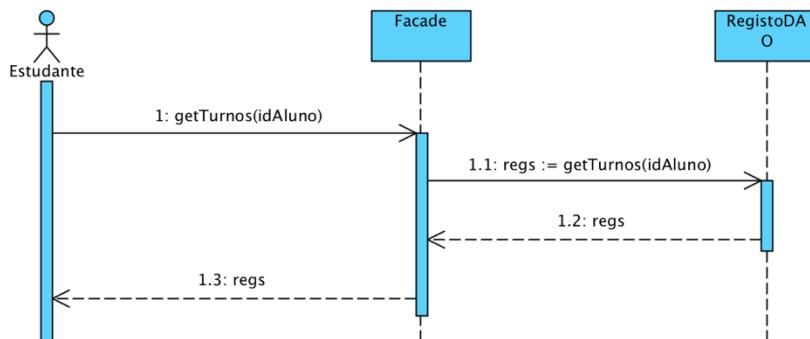
3. Registar UC



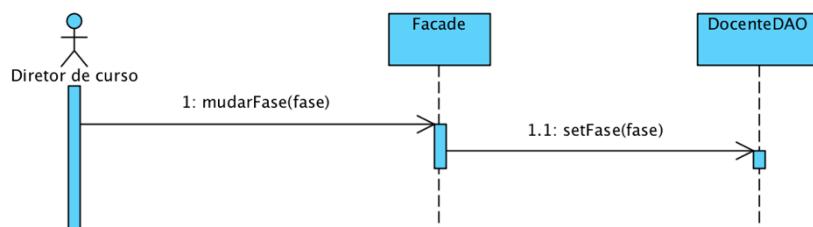
4. Remover aluno do turno



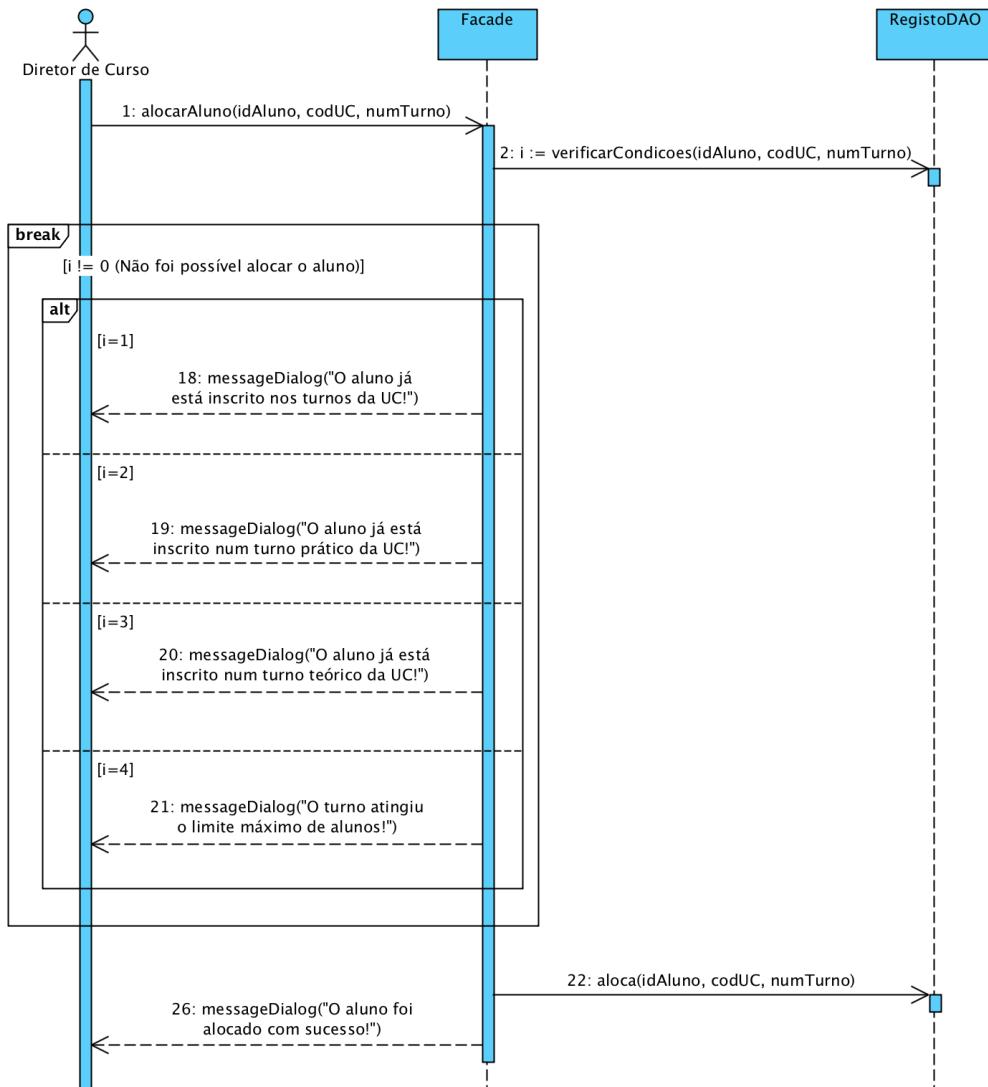
5. Ver turnos



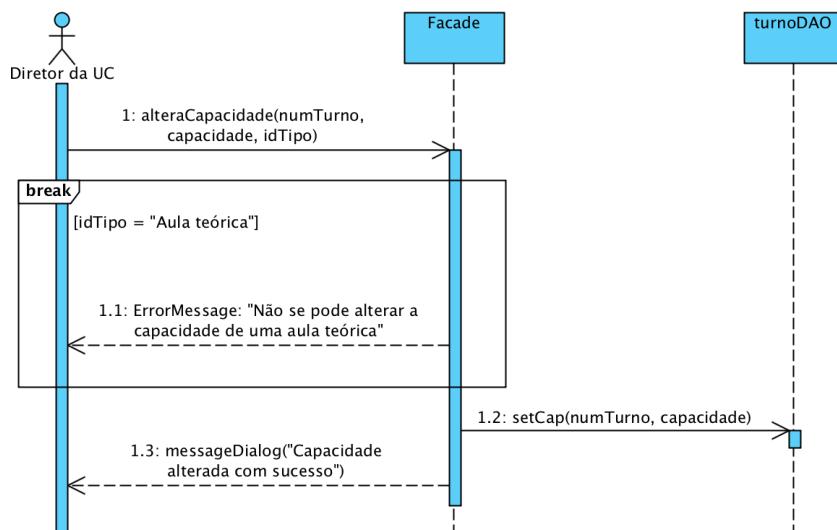
6. Mudar de fase



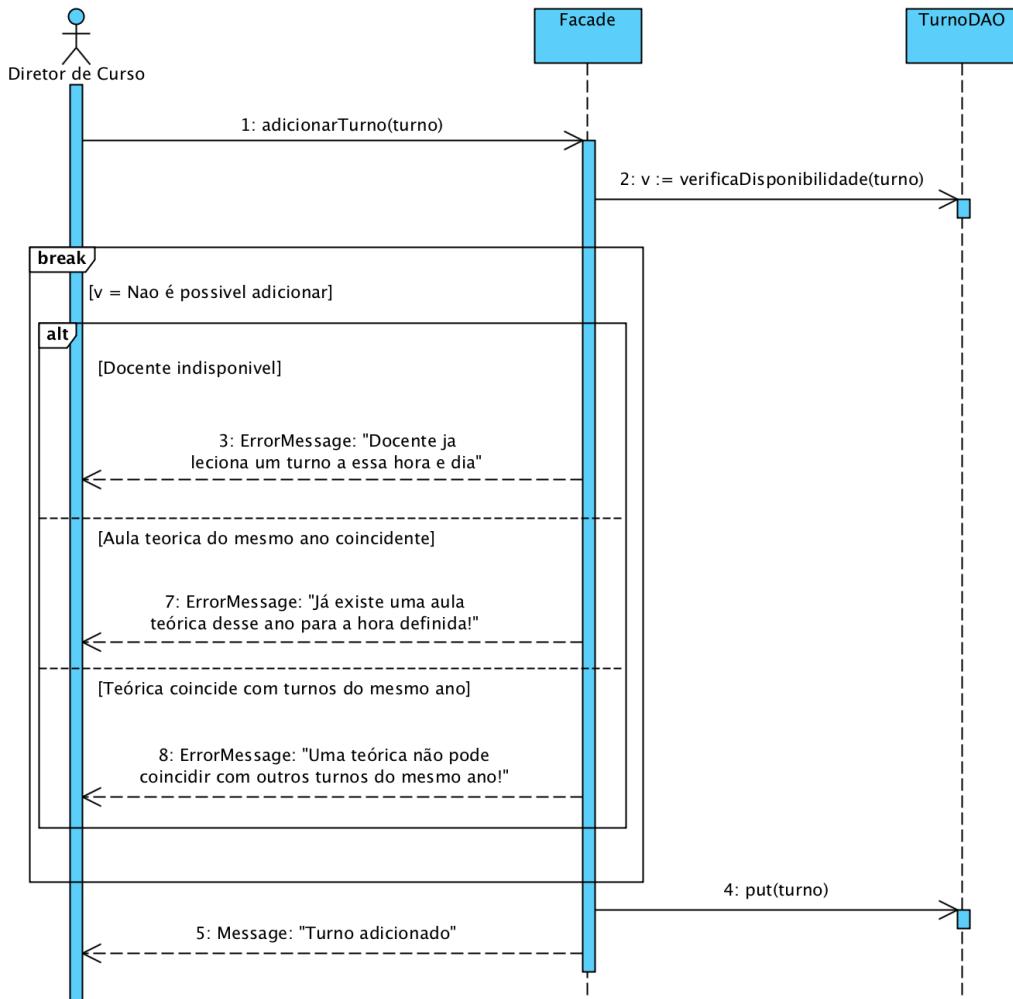
7. Alocar aluno



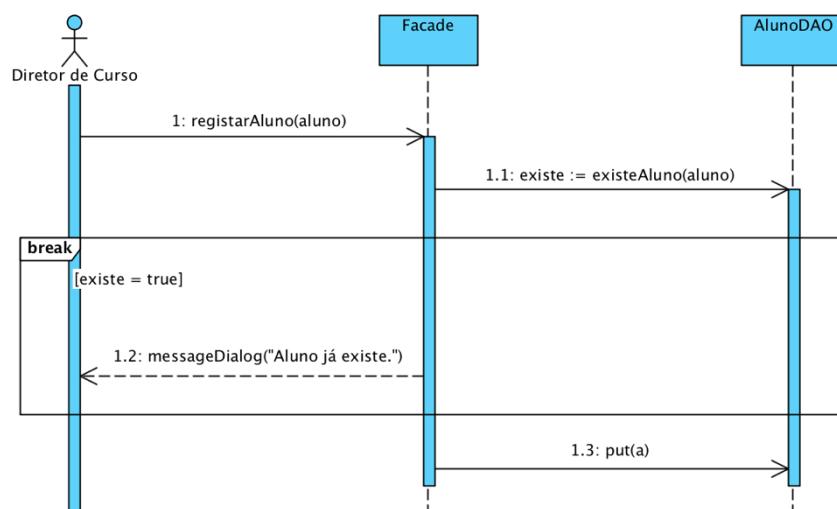
8. Definir capacidade



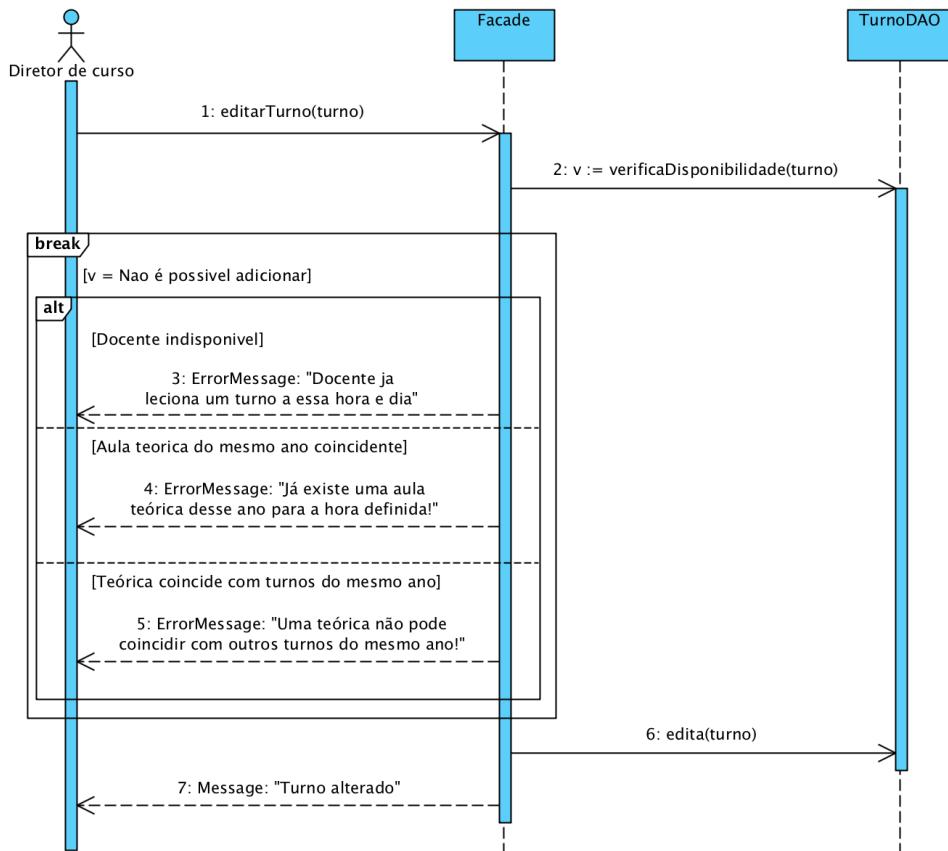
9. Adicionar turno



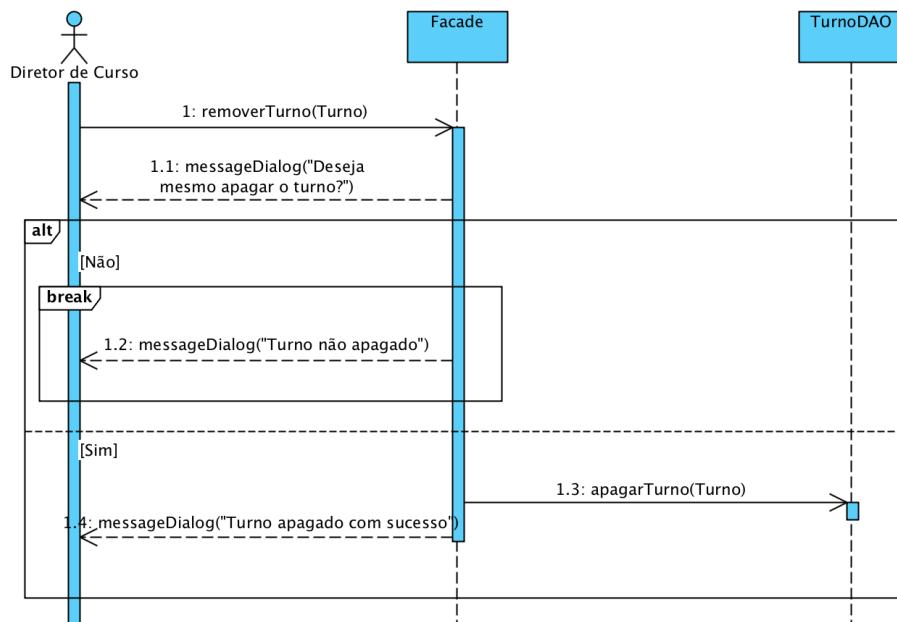
10. Registrar aluno



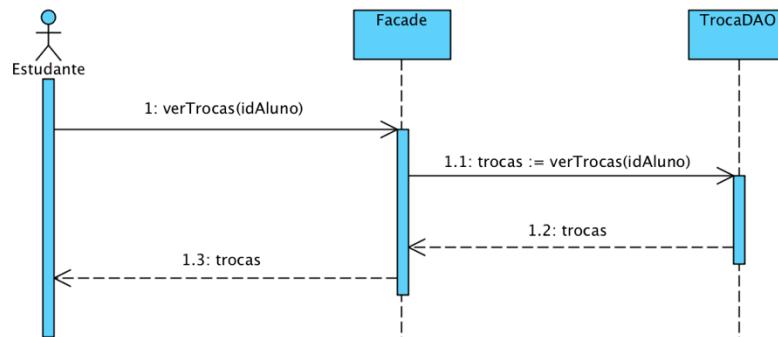
11. Editar turno



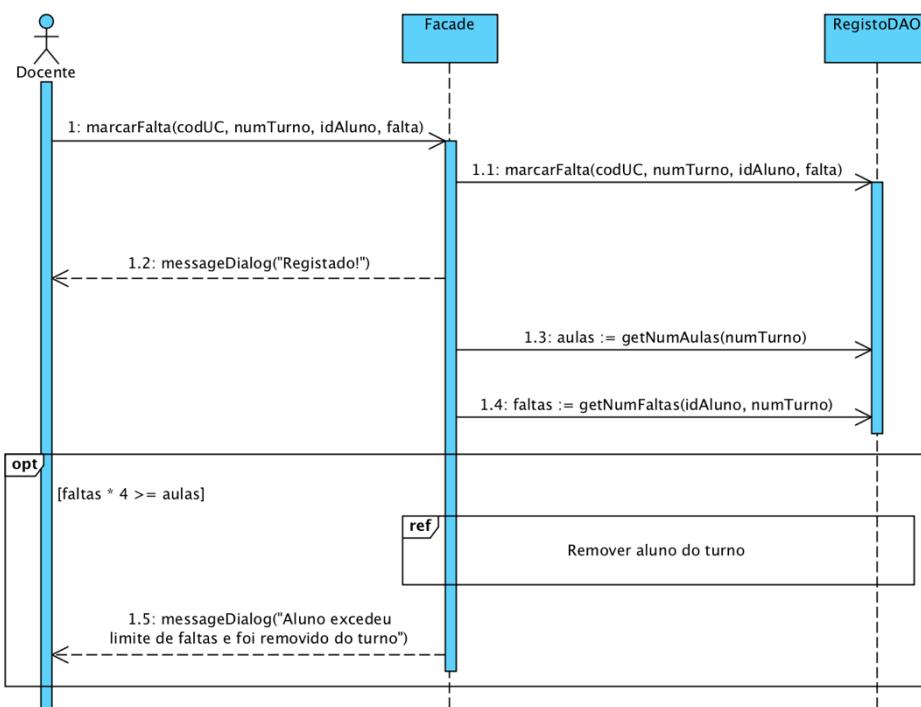
12. Remover turno



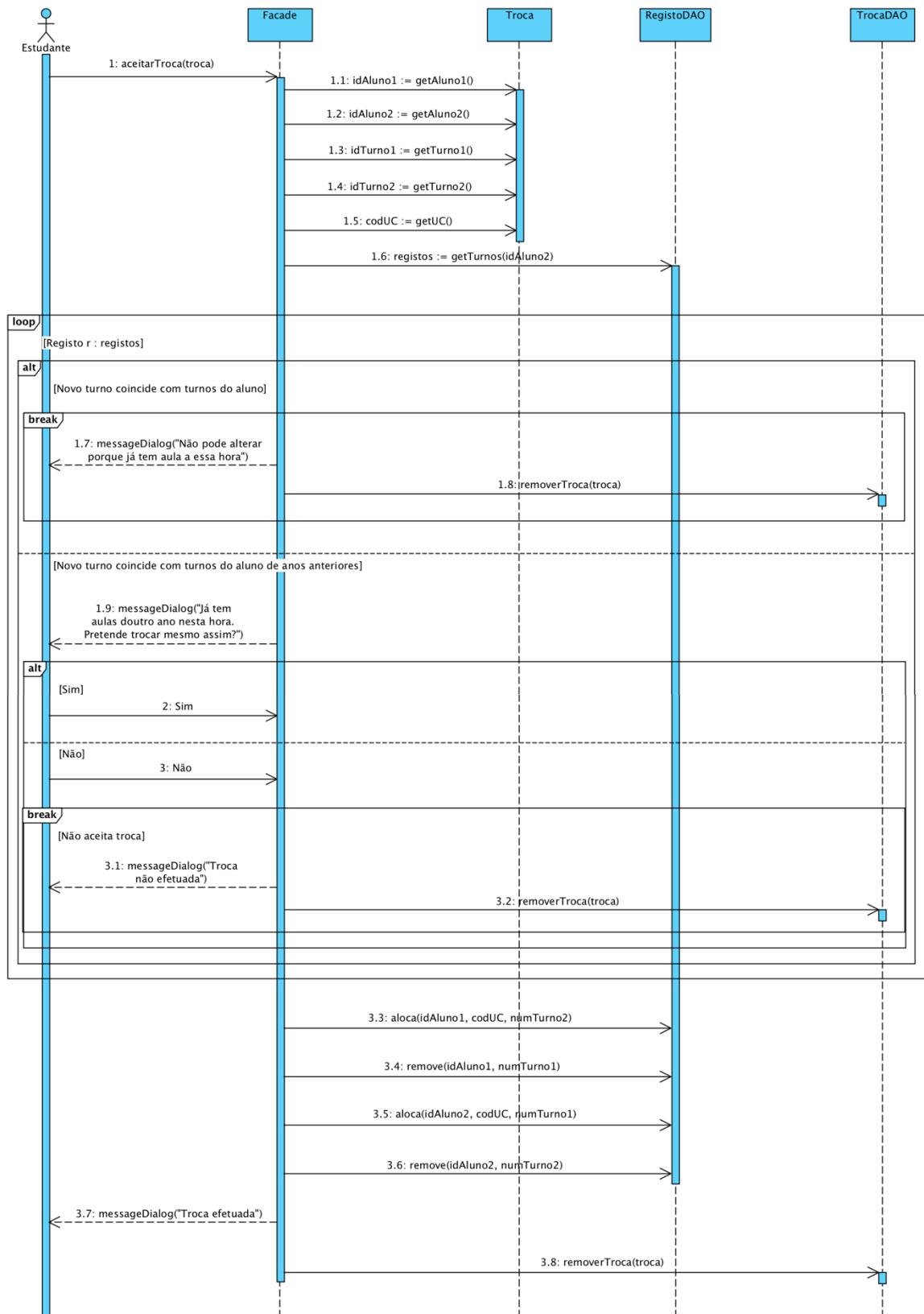
13. Ver trocas



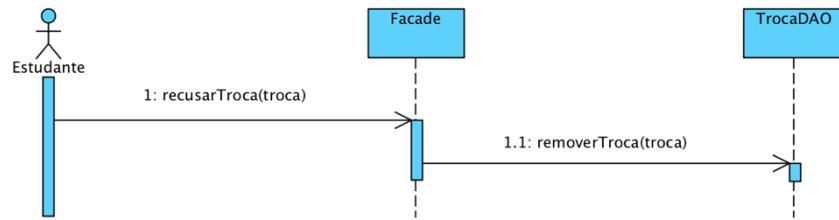
14. Marcar faltas



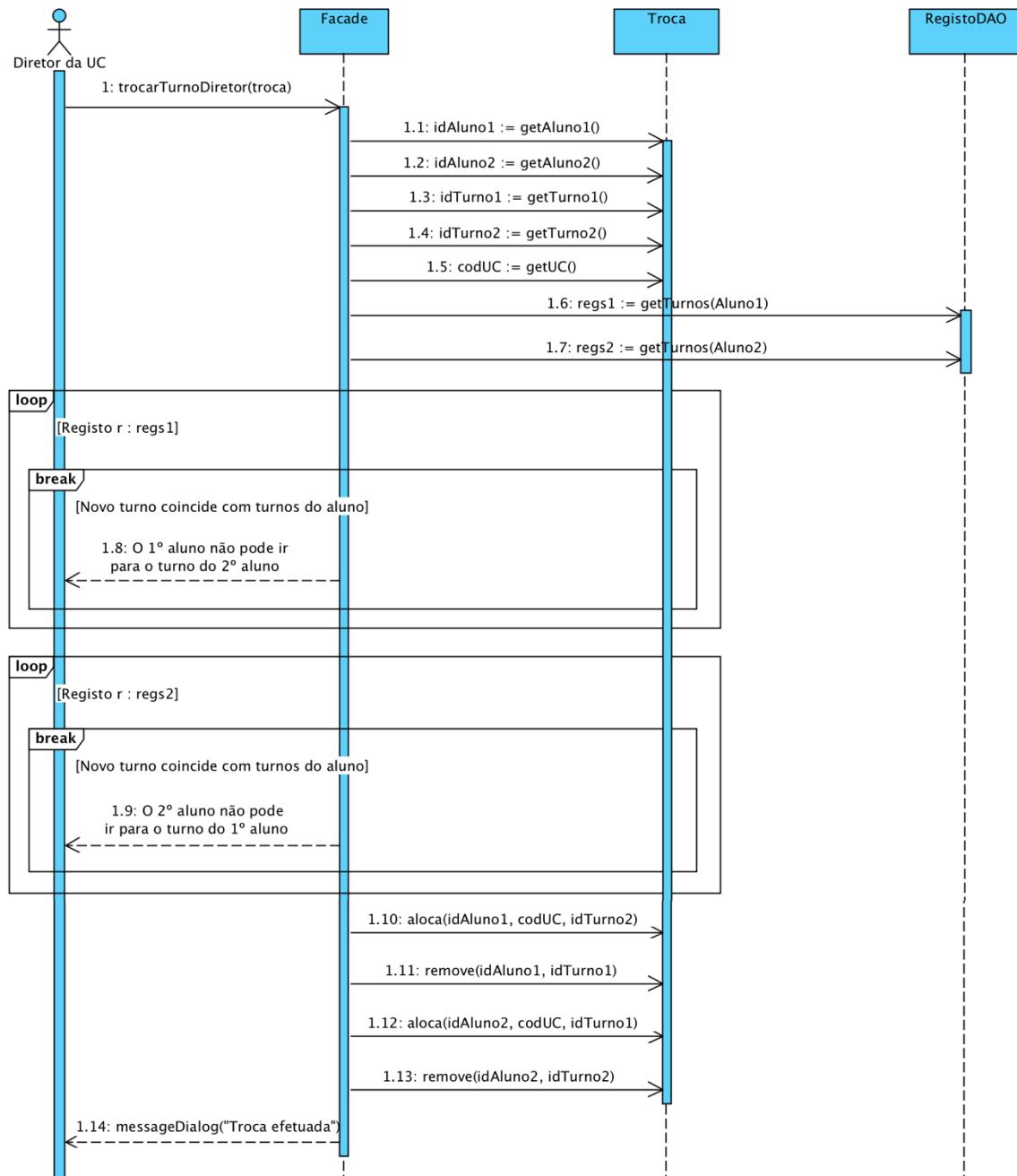
15. Aceitar troca



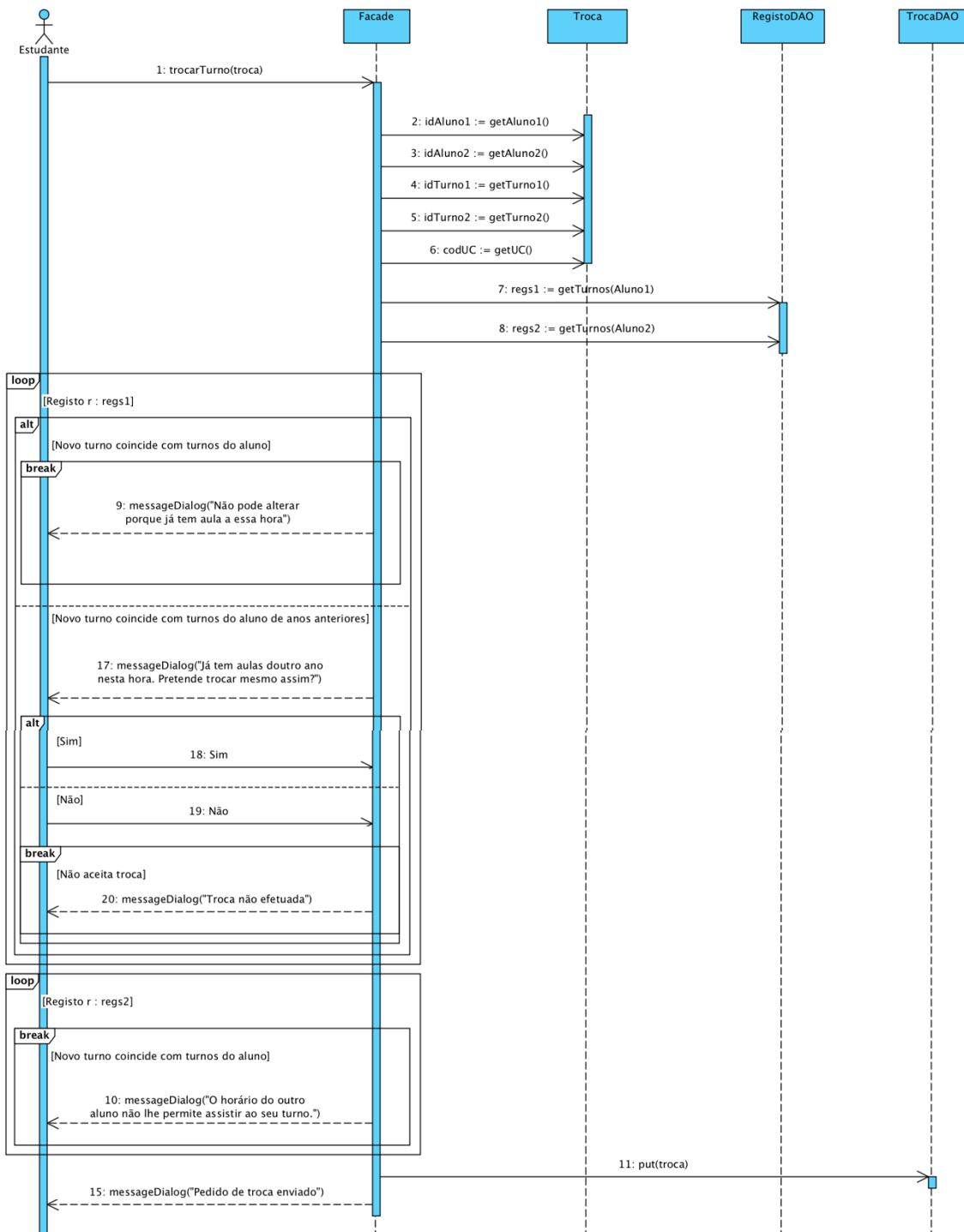
16. Recusar troca



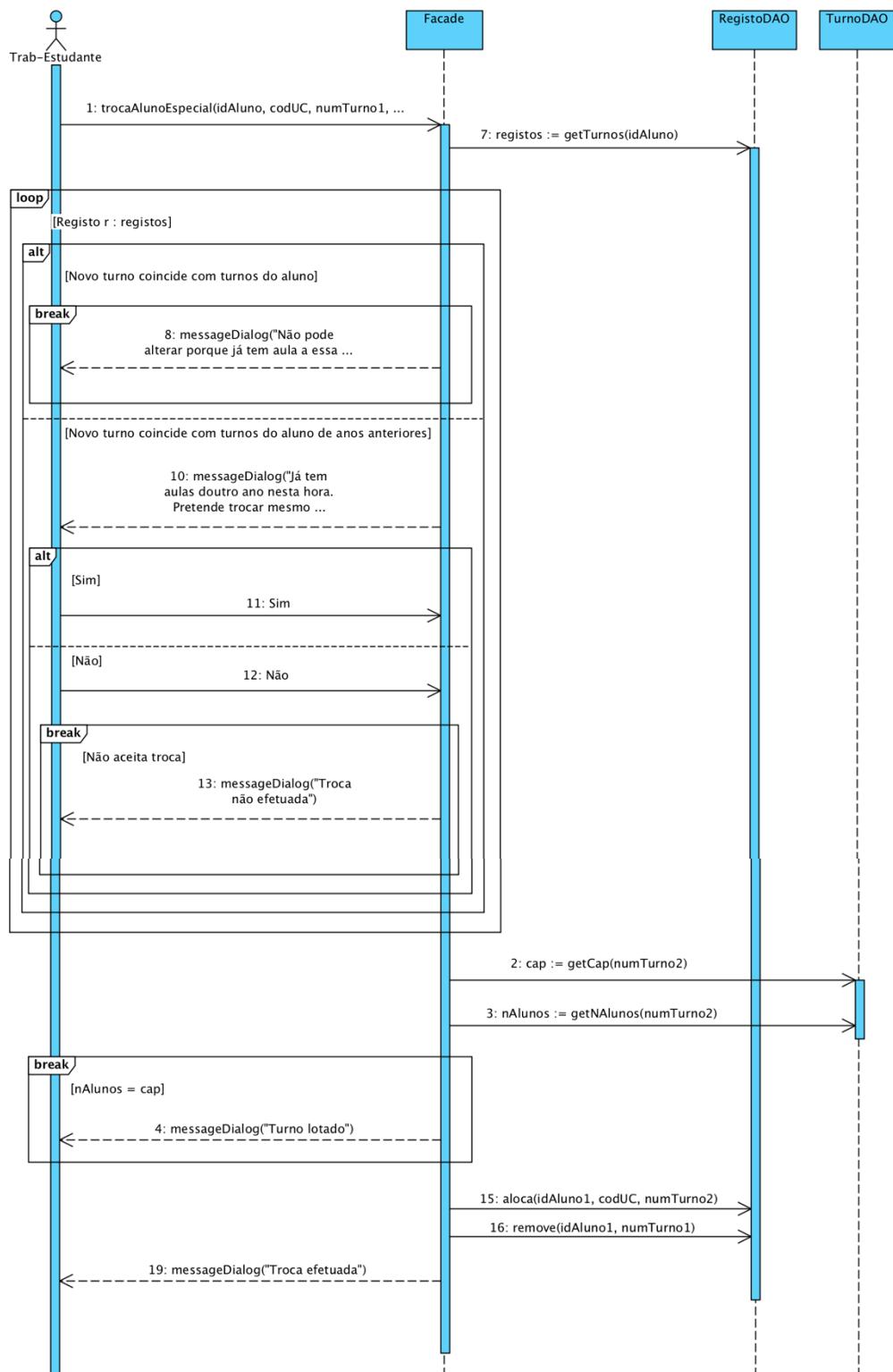
17. Trocar alunos



18. Trocar de turno (efetuar pedido)

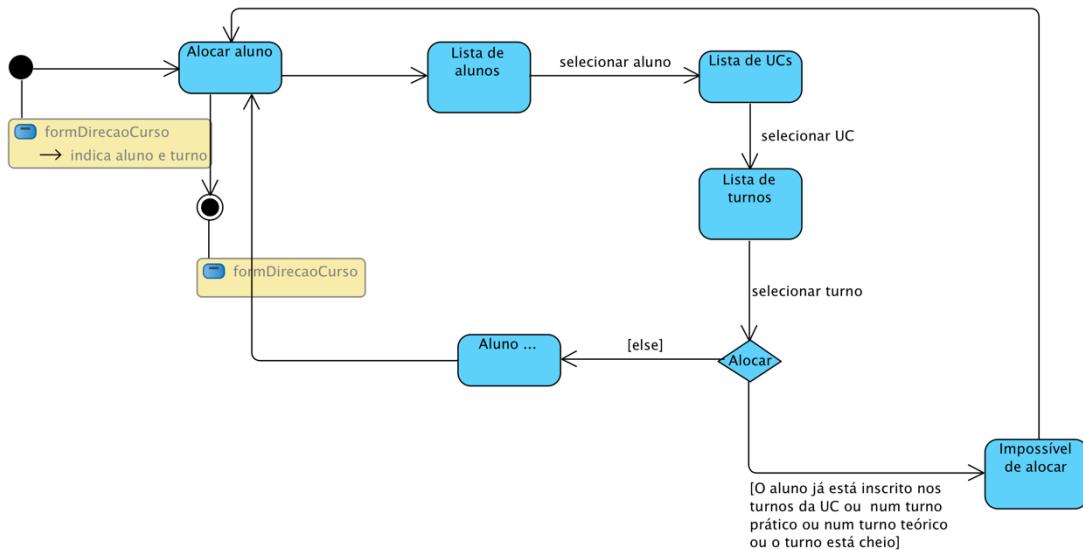


19. Trocar turno (trabalhador-estudante)

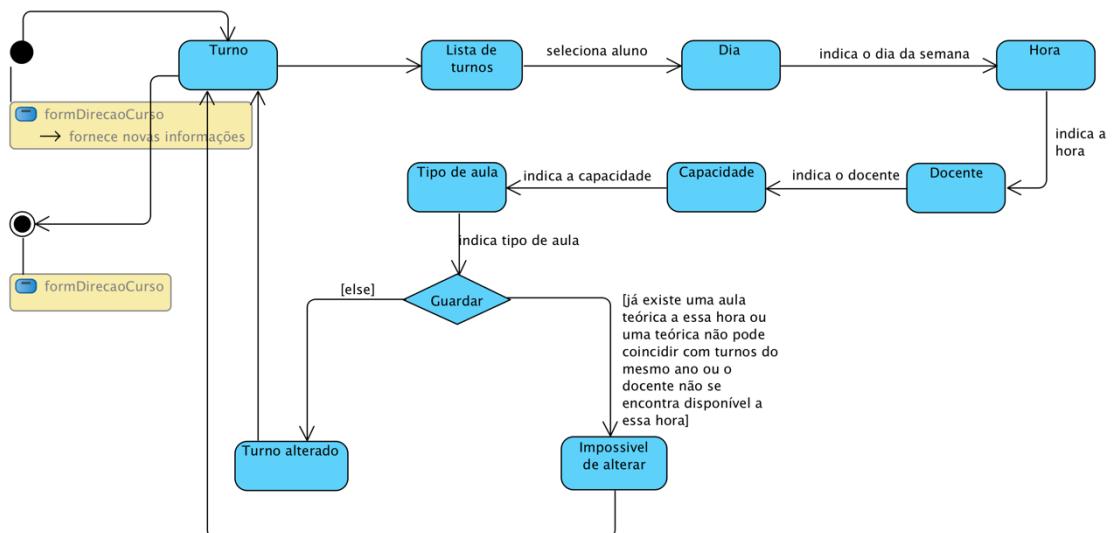


Subdiagramas de máquina de estado

1. Alocar aluno



2. Alterar turno



Bibliografia

- [1] P. Kroll e P. Kruchten, *Rational Unified Process Made Easy*, Addison Wesley, 2003, p. 464.
- [2] J. C. Campos e A. N. Ribeiro, *Desenvolvimento de Sistemas Software*, Escola de Engenharia - Universidade do Minho, 2017.
- [3] J. Hunt, *Guide to the Unified Process featuring UML, Java and Design Patterns*, Springer, 2003, p. 424.