Antonio Carlos Raposo

Levantamento de Requisitos e Modelagem do Ambiente Virtual de Aprendizagem COSMO

São Luís-MA, Brasil

2017

Antonio Carlos Raposo

Levantamento de Requisitos e Modelagem do Ambiente Virtual de Aprendizagem COSMO

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação

Universidade Federal do Maranhão – UFMA

Orientador: Carlos de Salles Soares Neto

São Luís-MA, Brasil

2017

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Ribeiro Raposo, Antonio Carlos.
Levantamento de requisitos e modelagem do ambiente virtual de Aprendizagem COSMO / Antonio Carlos Ribeiro Raposo. – 2018.
62 p.

Orientador(a): Carlos de Salles Soares Neto.
Monografia (Graduação) – Curso de Ciência da Computação, Universidade Federal do Maranhão, UFMA, 2018.

1. COSMO. 2. Levantamento de requisitos. 3. Modelagem. I. Soares Neto, Carlos de Salles. II. Título.

ANTONIO CARLOS RIBEIRO RAPOSO

LEVANTAMENTO DE REQUISITOS E MODELAGEM DO AMBIENTE VIRTUAL DE APRENDIZAGEM COSMO

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal Do Maranhão como parte dos requisitos necessários para obtenção do grau de bacharel em Ciência da computação

Trabalho Aprovado. São Luis-MA, Brasil, 15 de Janeiro de 2018:

Prof. Dr. Carlos De Salles Soares Neto
(Orientador)
Universidade Federal Do Maranhão

Profa. Dra. Simara V. Da Rocha
Universidade Federal Do Maranhão

Prof. Msc. Allan Kassio Beckman Soares Da Cruz
Unidade de Ensino Superior Dom Bosco

Agradecimentos

Os agradecimentos principais são destinados a minha mãe, Silvia Stela Durans Ribeiro, que sempre me apoiou e incentivou a seguir em frente em tudo na vida. Eu Não seria quem sou hoje sem você. Assim como meu irmão, Antonio Eliezer Raposo Jr, que sempre vi como um modelo a seguir e almejar.

Agradeço ao meu orientador, Carlos de Salles Soares Neto, por estar sempre disponível e por ser um ótimo amigo. Assim como Alexandre César, ex-tutor do PETCOMP ao qual me guiou no começo da minha graduação. Agradeço também a todos os membros do Telemídia, principalmente ao Dilson Rabêlo que me ajudou continuamente neste projeto, sem vocês nada disso seria possível.

Ao meu amigo Allan Kassio, que esteve sempre presente na minha vida e me inspirou a seguir a carreira na área de computação. Aos meus amigos Jordan Boaz, Jullyana Fialho e Danilo Nogueira que foram sempre leais. As minhas amigas Thais Lemos, Ana Isabel e Cintia Nogueira que sempre tiveram paciência para escutar o que eu tinha a dizer. Aos meus amigos dos grupos de *boardgame* e *TCG* que sempre estiveram lá por mim quando eu estava triste ou extressado.

Por fim, a todas as pessoas que já conheci até hoje, porque foram as experiências que eu tive com cada um que fez o que sou hoje.

"A felicidade pode ser encontrada inclusive nos momentos mais sombrios; só é preciso se lembrar de acender a luz.

(Alvo Dumbledore, Harry Potter e o prisioneiro de Azkaban)

Resumo

O curso de Ciência da Computação apresenta uma das maiores taxas de evasão entre os cursos superiores do Brasil. A disciplina de Algoritmos em geral apresenta o primeiro contato com programação de computadores e experiências negativas podem desaminar os estudantes afetando seriamente o aprendizado. Com isso em mente foi idealizado o COSMO, um ambiente virtual de aprendizagem focado no estudo de lógica de programação através de um conjunto de atividades, vídeos e jogos.Este trabalho tem por objetivo apresentar os requisitos funcionais e não funcionais do COSMO e sua modelagem. A coleta dos requisitos funcionais e não funcionais é feita utilizando técnicas de levantamento de requisitos e sua modelagem utiliza o padrão UML 2.0.Inicialmente foi feito entrevistas com os *stakeholders* do sistema separadamente a fim de se ter uma noção dos requisitos iniciais do COSMO, seguido por um *brainstorming* com a equipe de desenvolvimento a fim de adicionar possíveis funcionalidades que melhorem o sistema. foi então feita uma modelagem utilizando os requisitos coletados anteriormente. Esta modelagem produz um diagrama de caso de uso onde foi complementado os requisitos coletados anteriormente; diagramas de classe que apresenta um modelo conceitual e de domínio do sistema; diagramas de sequência que mostra a comunicação entre os objetos do sistema e diagramas de atividade que enfatiza a sequência de ações e condições a coordenar um comportamento do COSMO. Por fim foi comparado os requisitos obtidos do COSMO com diversas tecnologias relacionadas similares ao sistema apresentando o porque de cada plataforma não suprir as necessidades dos *stakeholders* e criado um protótipo do COSMO utilizando a modelagem coletada anteriormente.

Palavras-chave: Levantamento de Requisitos. Modelagem de Sistema. COSMO.

Abstract

The Computer Science course presents one of the highest dropout rates among college education in Brazil. The Algorithms discipline usually presents the first contact with computer programming and negative experiences can demean students seriously affecting learning. With this in mind it has been idealized COSMO, a virtual learning environment focused on the study of programming logic through a set of activities, videos and games.This paper aims to present the functional and non-functional requirements of COSMO and its modeling. The requirements gathering of functional and non-functional requirements is done using requirements surveying techniques and its modeling uses the UML 2.0 standard.Interviews were initially made with the stakeholders separately in order to get a sense of the initial requirements of COSMO followed by a brainstorming with the development team to add potential features that would improve the system. Modeling was then made using the requirements previously collected. This modeling produces a use case diagram where the requirements previously collected have been complemented; class diagrams that presents a conceptual and domain model of the system; sequence diagrams showing communication between system objects and activity diagrams that emphasizes the sequence of actions and conditions coordinating a COSMO behavior.Finally, the requirements obtained from COSMO were compared with several related technologies similar to the system, showing why each platform does not meet stakeholders' needs and it was created a COSMO prototype using the modeling previously collected.

Keywords: Requirements gathering. System Modeling. COSMO.

Lista de ilustrações

Figura 1 – Interface do UVa Online Judge para apresentar problemas 27
Figura 2 – Interface Uri Online Judge Academic para Professores 28
Figura 3 – Interface Uri Online Judge Academic para alunos 29
Figura 4 – Interface dos times no BOCA 30
Figura 5 – Interface dos Juízes no BOCA 30
Figura 6 – Interface dos Administradores no BOCA 31
Figura 7 – Interface dos *Staffs* no BOCA 31
Figura 8 – Interface dos Placar no BOCA 31
Figura 9 – Exemplo de Interface do Codecademy para exercícios 32
Figura 10 – Exemplo de um Ator chamado Professor 34
Figura 11 – Exemplo de um Caso de Uso chamado Criar Turma 34
Figura 12 – Diagrama de Caso de Uso Geral do COSMO 35
Figura 13 – Exemplo de classe 40
Figura 14 – Exemplo de associação 41
Figura 15 – Diagrama de Classe - Modelo Conceitual do COSMO 42
Figura 16 – Diagrama de Classe - Modelo de Domínio do COSMO 45
Figura 17 – Exemplo de Lifeline 48
Figura 18 – Exemplo de Mensagem 49
Figura 19 – Diagrama de Sequência - Realizar login 49
Figura 20 – Diagrama de Sequência - Registrar na Turma 50
Figura 21 – Diagrama de Sequência - Escolher Turma 50
Figura 22 – Diagrama de Sequência - Responder Tarefa 51
Figura 23 – Diagrama de Sequência - Visualizar Desempenho 51
Figura 24 – Diagrama de Sequência - Criar Turma 52
Figura 25 – Diagrama de Sequência - Manter Turma 52
Figura 26 – Diagrama de Sequência - Criar Tarefa 53
Figura 27 – Diagrama de Sequência - Manter Tarefa 53
Figura 28 – Exemplo de Nó de Ação 54
Figura 29 – Exemplo de Fluxo de Controle 55
Figura 30 – Diagrama de Atividade - Remover Aluno 55
Figura 31 – Diagrama de Atividade - Programming: Validar Resposta 56
Figura 32 – Diagrama de Atividade - Visualizar Desempenho 56
Figura 33 – Diagrama de Atividade - Responder Tarefa 57
Figura 34 – Diagrama de Atividade - Recomendar Tarefa 57
Figura 35 – Interface Inicial do COSMO 61
Figura 36 – Interface da recomendação de tarefas do cosmo 61

Figura 37 – Interface da tarefa 62

Lista de tabelas

Tabela 1 – Exemplo de Requisitos de Usuário e Sistema 18
Tabela 2 – Lista De Requisitos Funcionais 21
Tabela 3 – Requisito Funcional: Validar resposta da tarefa 22
Tabela 4 – Requisito Funcional: Recomendar Tarefa 22
Tabela 5 – Requisito Funcional: Buscar Turma 22
Tabela 6 – Requisito Funcional: Manter Turma 23
Tabela 7 – Requisito Funcional: Manter Tarefa 23
Tabela 8 – Requisito Funcional: Matricular na Turma 23
Tabela 9 – Requisito Funcional: Visualizar Desempenho 24
Tabela 10 – Requisito Funcional: Escolher Tarefa 24
Tabela 11 – Requisito Funcional: Tipos de Tarefa 24
Tabela 12 – Requisito Funcional: Responder Tarefa 25
Tabela 13 – Requisitos Não Funcionais 25
Tabela 14 – Caso de Uso : Realizar Login 34
Tabela 15 – Caso de Uso : Auto-registrar 35
Tabela 16 – Caso de Uso : Registrar na turma 36
Tabela 17 – Caso de Uso : Escolher Turma 36
Tabela 18 – Caso de Uso : Escolher Tarefa 36
Tabela 19 – Caso de Uso : Recomendar Tarefa 37
Tabela 20 – Caso de Uso : Responder Tarefa 37
Tabela 21 – Caso de Uso : Validar Resposta 37
Tabela 22 – Caso de Uso : Criar Turma 38
Tabela 23 – Caso de Uso : Manter Turma 38
Tabela 24 – Caso de Uso : Criar Tarefa 38
Tabela 25 – Caso de Uso : Manter Tarefa 39
Tabela 26 – Caso de Uso : Visualizar Desempenho 39
Tabela 27 – Descrição da classe Usuário 41
Tabela 28 – Descrição da classe Aluno 42
Tabela 29 – Descrição da classe Professor 43
Tabela 30 – Descrição da classe Turma 43
Tabela 31 – Descrição da classe Tarefa 43
Tabela 32 – Descrição da classe Programming 43
Tabela 33 – Descrição da classe Video 44
Tabela 34 – Descrição da classe Matricula 44
Tabela 35 – Descrição das classe Resposta 44
Tabela 36 – Métodos da classe Aluno 45

Tabela 37 – Métodos da classe Professor 46
Tabela 38 – Métodos da classe Programming 46
Tabela 39 – Métodos da classe Video 46
Tabela 40 – Métodos da classe Turma 47
Tabela 41 – Comparativo geral entre o COSMO e as tecnologias relacionadas . . . 58

Sumário

1 INTRODUÇÃO 14
1.1 Objetivos 15
1.1.1 Objetivo geral 15
1.1.2 Objetivos Específicos 15
1.2 Estrutura do trabalho 15

2 LEVANTAMENTO DE REQUISITOS 17
2.1 Técnicas Utilizadas 18
2.1.1 Entrevistas 19
2.1.2 Diagrama de Caso de Uso 19
2.1.3 Brainstorming 20
2.2 Requisitos Identificados 20

3 TECNOLOGIAS RELACIONADAS 26
3.1 UVa Online Judge 26
3.2 URI Online Judge Academic 27
3.3 BOCA 29
3.4 Codecademy 32

4 MODELAGEM UML 33
4.1 Diagrama de Caso de Uso 33
4.2 Diagrama de Classe 40
4.3 Diagrama de Sequência 48
4.4 Diagrama de Atividade 54

5 RESULTADOS 58
5.1 Comparativo de requisitos 58
5.1.1 Uva Online Judge 58
5.1.2 URI Online Judge 59
5.1.3 BOCA 59
5.1.4 Codecademy 59
5.1.5 COSMO 60
5.2 Prototipação do sistema 60

6 CONCLUSÃO 63

REFERÊNCIAS 64

1 Introdução

Um dos principais problemas enfrentados pelos cursos de graduação na área de computação é a alta taxa de desistência. Ciência da Computação possui uma das maiores taxas de evasão entre os cursos superiores no Brasil. Segundo (PALMEIRA; SANTOS, 2015), cursos de Ciência da Computacão possuem uma taxa de concluintes de 14,3%. (SANTOS; COSTA, 2006) afirmam que um dos motivos para desistência nos cursos é a dificuldade em aprender os conceitos de programação.

Então deve-se levar em consideração a importância da disciplina de algoritmos. Essa disciplina representa para a maioria dos estudantes um primeiro contato com programação de computadores. Experiências negativas podem desanimar os estudantes de um estudo mais profundo e afetar o aprendizado (ALLISON et al., 2002).

Uma maneira de reduzir as taxas de abandono é tornar o ensino de programacão mais dinâmico. Para isso, uma proposta baseada na solução de problemas é um modo de ensino em que o aluno aumenta sua capacidade em programação codificando uma solução computacional. O método atende as necessidades dos alunos e, assim, aumenta o êxito do aprendizado.

Com isso em mente foi criado o conceito do COSMO, um ambiente virtual de aprendizagem focado no estudo de lógica de programação através de um conjunto de atividades, vídeos e jogos. Com isso, almeja-se um aumento motivacional e melhoria de desempenho na disciplina de algoritmos.

Entretanto, a grande maioria dos softwares de média e alta complexidade, como o COSMO, falham durante o desenvolvimento. De acordo com diversos estudos, cerca de 50% a 80% dos projetos de software falham durante o seu desenvolvimento (RAJKUMAR; ALAGARSAMY, 2013).

Os projetos de software falham por diversos motivos que incluem falta de recursos, falta de estimação correta do custo de desenvolvimento e outros, mas principalmente pela falta de clareza dos objetivos e requisitos do sistema a ser desenvolvido (RAJKUMAR; ALAGARSAMY, 2013).

Falha do projeto devido a requisitos ruins acontecem quando a equipe do projeto entrega um produto sem ter uma clara compreensão do que o cliente quer e sem ter um conhecimento real dos requisitos. Quando o produto é finalizado, O cliente não fica satisfeito pois os requisitos não foram atendidos. Requisitos mal definidos e a falta de entendimento de requisitos em geral ocorre pela falta de envolvimento do cliente e pela falta de métodos de engenharia de software durante o processo de desenvolvimento

(RAJKUMAR; ALAGARSAMY, 2013).

Um método de engenharia de software é uma abordagem para desenvolvimento de software, cujo objetivo é facilitar o desenvolvimento de software de alta qualidade dentro de custos adequados (SOMMERVILLE et al., 2003). Não existe um método ideal e diferentes métodos possuem diferentes áreas onde são aplicáveis.

Sendo assim, o tema abordado trata-se do "Levantamento de requisitos e modelagem do ambiente virtual de aprendizagem COSMO", onde pretende-se, a partir de requisitos coletados utilizando técnicas de levantamento de requisitos, gerar uma modelagem de sistema no formato UML 2.0.

1.1 Objetivos

1.1.1 Objetivo geral

O principal objetivo deste trabalho é utilizar técnicas de elicitação de requisito a fim de apresentar os requisitos e a modelagem do ambiente virtual de aprendizagem COSMO. Os requisitos coletados e modelagem apontados devem ser apresentados com eficácia e clareza.

1.1.2 Objetivos Específicos

a) Utilizar técnicas de elicitação de requisitos para documentar os requisitos funcionais e não funcionais do COSMO.

b) Produzir uma modelagem utilizando diagrama de caso de uso, diagrama de classe , diagrama de sequência e de atividade presentes na UML 2.0 usando os requisitos coletados anteriormente.

c) Apresentar um comparativo entre os requisitos do ambiente virtual de aprendizagem COSMO e as tecnologias relacionadas apresentadas.

1.2 Estrutura do trabalho

Este trabalho está estruturado de forma a apresentar gradativamente todos os passos tomados para obter os requisitos e a modelagem do ambiente virtual de aprendizagem COSMO.

No Capítulo 2 é discorrido sobre a metodologia utilizada para fazer a elicitação de requisitos do COSMO. Além disso, também apresenta os requisitos funcionais e não funcionais do ambiente virtual de aprendizagem. Em seguida, o Capítulo 3 apresenta

um conjunto de tecnologias relacionadas ou similares ao COSMO, apresentando suas qualidades e problemas individualmente.

No Capítulo 4 é discorrido sobre a *unified modeling language* e seus diagramas. Além disso também será apresentada a modelagem do sistema utilizando os requisitos coletados no Capítulo 2, incluindo diagramas de caso e uso, diagrama de classe, diagrama de sequência e diagrama de atividade.

Em seguida, o Capítulo 5 apresenta um comparativo entre as tecnologias relacionadas apresentadas no Capítulo 3 e o ambiente virtual de aprendizagem COSMO, levando em conta os requisitos coletados durante o Capítulo 2. E por fim, uma discussão é feita dos resultados obtidos e o potencial do sistema em trabalhos futuros.

2 Levantamento de Requisitos

Os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento (SOMMERVILLE et al., 2003). Esses requisitos refletem as necessidades dos clientes para um sistema que serve a uma finalidade determinada, como manipular um dispositivo, fazer um pedido ou encontrar informações. O processo de descobrir, analisar, documentar e verificar esses serviços e restrições é chamado engenharia de software (SOMMERVILLE et al., 2003).

Alguns dos problemas que surgem durante o processo de engenharia de requisitos são as falhas em não fazer uma separação clara entre os diferentes níveis de descrição. (SOMMERVILLE et al., 2003) faz uma distinção entre eles usando os termos requisitos de usuário e de sistema. Os requisitos de usuário expressam as necessidades abstratas de alto nível. Por sua vez, requisitos de sistema expressam a descrição detalhada do que o sistema deve fazer. De acordo com (SOMMERVILLE et al., 2003), "requisitos de usuário e de sistema são definidos como segue: requisitos de usuário são declarações, em uma linguagem natural com diagramas, de quais serviços o sistema deverá fornecer ao seus usurários e as restrições com quais este deve operar, enquanto requisitos de sistema são descrições mais detalhadas das funções, serviços e restrições operacionais do sistema de software".

Diferentes níveis de requisitos são úteis, pois eles comunicam informações sobre o sistema para diferentes tipos de usuários. A Tabela 1 ilustra a distinção entre requisitos de usuário e de sistema. Tal exemplo mostra como um requisito de usuário pode ser expandido em diversos requisitos de sistema.

Além disso, Os requisitos de software são com frequência classificados como requisitos funcionais e não funcionais.

Os requisitos funcionais descrevem a funcionalidade ou os serviços que se espera que o sistema realize (FILHO, 2003). Quando expressos como requisitos de usuário, os requisitos funcionais são normalmente descritos de forma abstrata, para que sejam compreendidos pelos usuários do sistema, enquanto requisitos de sistema funcionais descrevem em detalhes a função do sistema (SOMMERVILLE et al., 2003).

Já os requisitos não funcionais são aqueles que não estão diretamente relacionados a funções especificas fornecidas pelo sistema (SOMMERVILLE et al., 2003). Eles podem estar relacionados a propriedades do sistema, como confiabilidade, tempo de resposta, espaço em disco, desempenho e outros atributos de qualidade do produto (FILHO, 2003).

Em geral, os requisitos não funcionais acabam sendo mais importantes que os funcionais, pois uma falha em um requisito não funcional pode comprometer o sistema

Tabela 1 – Exemplo de Requisitos de Usuário e Sistema

Requisito de Usuário	
Numero	Descrição
RF01	O sistema deve gerar relatórios gerenciais mensais que exibem o custo dos medicamentos por cada clinica durante o mês.
Requisito de Sistema	
RF01.1	No ultimo dia útil de cada mês deve ser gerado um resumo dos medicamentos prescritos, seus custos e as prescrições de cada clinica.
RF01.2	Após 17:30h do último dia útil do mês, o sistema deve gerar automaticamente o relatório para impressão..
RF01.3	Um relatório será criado para cada clínica, listando os nomes dos medicamentos,o número total de prescrições, o número de doses prescritas e o custo total dos medicamentos prescritos.
RF01.4	Se os medicamentos estão disponíveis em diferentes unidades de dosagem (por exemplo, 10 mg, 20 mg), devem ser criados relatórios separados para cada unidade..
RF01.5	O acesso aos relatórios de custos deve ser restrito a usuários autorizados por uma lista de controle de gerenciamento de acesso.

Fonte: (SOMMERVILLE et al., 2003)

como um todo (SOMMERVILLE et al., 2003).

Os requisitos não funcionais não estão relacionados somente com o sistema de software a ser desenvolvido. Algum deles podem restringir o processo que deve ser usado para desenvolver o sistema. (SOMMERVILLE et al., 2003) classifica os requisitos não funcionais em três tipos: requisito de produto; requisitos organizacionais; e requisitos externos.

Requisitos de produto especificam o comportamento do produto. Entre exemplos estão requisitos de desempenho quanto à rapidez do sistema, requisitos de confiabilidade que define as taxas aceitáveis de falhas e requisitos de usabilidade.

Requisitos organizacionais são políticas e procedimentos da organização do cliente e do desenvolvedor, alguns exemplos incluem padrões de processo que devem ser usados.

Requisitos externos abrangem todos os requisitos derivados de fatores externos ao sistema e seu processo de desenvolvimento, como requisitos éticos e legais que garantem que o sistema funcione dentro da lei.

2.1 Técnicas Utilizadas

Para fazer o levantamento de requisitos, os engenheiros de software trabalham com os clientes e usuários finais do sistema para aprender sobre o domínio da aplicação, quais

serviços o sistema deve fornecer, entre outras informações.

O levantamento de requisitos pode envolver várias pessoas. O termo *stakeholder* é usado para referir a qualquer pessoa ou grupo afetado pelo sistema, direta ou indiretamente (SOMMERVILLE et al., 2003).

O levantamento de requisitos e compreensão dos requisitos dos *stakeholders* são difíceis devido a diversas razões, que incluem a falta de conhecimento do que querem e a dificuldade de se expressar, além de termos gerais por parte dos *stackholders*. Além disso, os mesmos expressam requisitos de forma natural em seus próprios termos, o que faz com que os engenheiros sem conhecimento do domínio tenham dificuldades em entender os requisitos. Devido a isso foram desenvolvidos diversos métodos para realizar o levantamento de requisitos.

2.1.1 Entrevistas

Entrevistas formais e informais com os *stackholders* do sistema são parte da maioria dos processos de levantamento de requisitos (SOMMERVILLE et al., 2003). Nessa etapa, a equipe de engenharia de requisitos entrevista os *stackholders* sobre os sistemas usados no momento e sobre o sistema que será desenvolvido. Os requisitos surgem a partir das respostas a essas perguntas.

As entrevistas podem ser de dois tipos: entrevistas fechadas, em que o *stakeholder* responde a um conjunto predefinido de perguntas, e entrevistas abertas, em que não existe uma agenda predefinida (SOMMERVILLE et al., 2003).

Entrevistas são boas para obter uma compreensão do que os *stackholder* fazem, como eles devem interagir com o novo sistema e as dificuldades que eles enfrentam com os sistemas atuais.

Informações recolhidas em entrevistas suplementam outras informações sobre o sistema, advindas de documentos que descrevem processos de negócios ou sistemas existentes etc (SOMMERVILLE et al., 2003). Em alguns casos, além da informação contida nos documentos do sistema, as entrevistas podem ser a única fonte de informação sobre os requisitos do sistema. No entanto, a entrevista por si só provavelmente deixará escapar informações essenciais; por isso, deve ser usada em conjunto com outras técnicas de levantamento de requisitos.

2.1.2 Diagrama de Caso de Uso

Os casos de uso são uma técnica de descoberta de requisitos introduzida no método Objectory (JACOBSON et al., 1997). Eles já se tornaram uma característica fundamental da linguagem de modelagem unificada. Um caso de uso identifica os atores envolvidos em uma interação e dá nome ao tipo de interação, que é então suplementada por informações

adicionais que descrevem a interação com o sistema (SOMMERVILLE et al., 2003). A informação adicional pode ser uma descrição textual ou um ou mais modelos gráficos, como o diagrama de sequência da UML.

Os casos de uso são documentados por um diagrama de casos de uso de alto nível. O conjunto de casos de uso representa todas as possíveis interações que serão descritas nos requisitos de sistema. Atores, que podem ser pessoas ou outros sistemas, são representados como figuras 'palito'. Cada classe de interação é representada por uma elipse. Uma definição mais ampla dos diagramas de casos de uso é feita no Capitulo 4.

2.1.3 Brainstorming

A técnica de tempestade de ideias, mais conhecida como *brainstorming* visa a geração de ideias em grupo. A equipe se reune e se concentra em analisar um problema e apresentar a solução para o mesmo (CYBIS et al., 2015). A vantagem do *brainstorming* está nas discussões que se estabelecem, melhorando a compreensão tanto do problema quanto da solução.

A técnica prevê duas etapas básicas: a geração de ideias, seguida de sua crítica(CYBIS et al., 2015).

Na etapa de geração de ideias, deve-se organizar uma mesa redonda e convidar cada um dos membros para expor suas ideias. Nesse momento é necessário ter cuidado com três aspectos importantes: garantir que todos os grupos tenham iguais oportunidade de expressar suas ideias; evitar que outros integrantes façam críticas ou tentativas de avaliação das ideias; e certificar-se de que todas as ideias geradas sejam registradas em uma lista (CYBIS et al., 2015). O encerramento dessa etapa se faz quando o fluxo de geração de ideias acaba.

A etapa de crítica se caracteriza pelo descarte de ideias consideradas impraticáveis ou equivocadas.

2.2 Requisitos Identificados

Para este trabalho, inicialmente foi feita uma entrevista com os *stakeholders* do ambiente virtual de aprendizagem COSMO, inicialmente com o professor que idealizou o conceito e depois com o aluno de graduação desenvolvedor do sistema. Cada entrevista teve uma duração de aproximadamente três horas e foi seguido o padrão de entrevista livre.

Durante a entrevista foi identificado a necessidade de uma validação automática das respostas apresentadas pelos alunos para os problemas apresentados pelo professor, além disso ficou claro que o professor não quer ter que apontar quais tarefas serão feitas

pelos alunos, surgindo então a necessidade de um recomendador de tarefas. O *stakeholder* principal também deixou claro que o sistema precisar ser web e online sempre, pois um dos problemas é manter a atenção dos alunos dentro e fora de aula.

Em seguida, foi feita uma sessão de *brainstorming* com os *stakeholders* e a equipe de desenvolvimento do COSMO, cada membro teve oportunidade de expor suas ideias e então foram descartadas as inviáveis e equivocadas.

O *brainstorming* serviu principalmente para aprimorar os requisitos já coletados anteriormente na etapa da entrevista e para adicionar novas possíveis funcionalidades ao COSMO. Durante o *brainstorming* ficou claro que a recomendação de tarefas tinha que ser feita baseado no histórico do aluno e que o professor precisaria de um conceito de turma dentro do sistema para ficar mais fácil manusear o sistema.

Por fim, foi utilizado Diagrama de Caso de Uso para refinar os requisitos coletados nas etapas anteriores. O diagrama de caso de uso e sua documentação pode ser visto no Capítulo 4.

Os requisitos coletados durante o processo são apresentados na Tabela 2.

Tabela 2 – Lista De Requisitos Funcionais

Numero	Descrição
RF01	Validar resposta da Tarefa.
RF02	Recomendar Tarefa.
RF03	Buscar Turma
RF04	Manter Turma.
RF05	Manter Tarefa.
RF06	Matricular na Turma.
RF07	Visualizar Desempenho.
RF08	Escolher Tarefa.
RF09	Tipos de Tarefa.
RF10	Responder Tarefa.

A Tabela 3 representa o detalhamento do requisito funcional Validar Resposta da Tarefa. Este requisito funcional foi identificado durante a a etapa da entrevista, o *stakeholder* principal do sistema deixou claro que um dos objetivos do sistema é de automatizar a validação das tarefas apresentadas durante o curso. O mesmo indicou que seria inviável fazer o acompanhamento e correção manual em uma turma com 50 alunos, como as que trabalha comumente.

A Tabela 4 representa o detalhamento do requisito funcional Recomendar Tarefa. Este requisito funcional foi identificado inicialmente durante a entrevista com o *stakeholder* principal, e depois expandido durante o *brainstorming*. De acordo com o observado em tais etapas, a recomendação de tarefas por parte do sistema para o aluno é fundamental para o usuário final do sistema.

Tabela 3 – Requisito Funcional: Validar resposta da tarefa

Numero	Descrição
RF01	O professor precisa que o sistema valide a resposta apresentada pelo aluno para cada tarefa.
RF01.1	O tempo da validação da resposta pelo sistema deve ser instantâneo.
RF01.2	Após o aluno enviar sua resposta, o sistema deve apresentar uma mensagem de erro ou de acerto, e indicar uma outra tarefa para ser respondida.
RF01.3	Para respostas do tipo programáveis, o sistema devera comparar a entrada/saída da resposta do aluno com uma base de entrada/saída já validadas.
RF01.4	Para respostas do tipo video, o sistema devera reconhecer que o aluno assistiu o video até o final e considerar a tarefa como correta.

Tabela 4 – Requisito Funcional: Recomendar Tarefa

Numero	Descrição
RF02	O Aluno precisa que o sistema recomende tarefas para ele.
RF02.1	O sistema deve recomendar tarefas para o aluno de acordo com as tarefas já respondidas na turma pelo mesmo.
RF02.2	O Aluno deve ser capaz de pular uma tarefa recomendada a ele que ele não quer responder.
RF02.3	Caso não exista mais tarefas para serem respondidas o sistema deve mostrar uma mensagem parabenizando o aluno.

A Tabela 5 representa o detalhamento do requisito funcional Buscar Turma. Este requisito funcional foi identificado inicialmente durante o *brainstorming* e expandido durante a etapa de criação dos casos de uso. durante o brainstorming ficou claro que as turmas indisponíveis não devem aparecer na busca de turmas.

Tabela 5 – Requisito Funcional: Buscar Turma

Numero	Descrição
RF03	O Aluno precisa ser capaz de buscar e filtrar as turmas disponíveis.
RF03.1	O aluno deve ser capaz de filtrar as turmas por matéria, nome, professor, etc.
RF03.2	Ao filtrar o sistema deve sempre mostrar as turmas em que o aluno já está registrado primeiro.
RF03.3	Turmas indisponíveis não devem aparecer na busca.

A Tabela 6 representa o detalhamento do requisito funcional Manter Turma. Este requisito funcional foi identificado inicialmente durante a criação dos casos de uso. É necessário que o professor seja capaz de manipular os dados da turma.

A Tabela 7 representa o detalhamento do requisito funcional Manter Tarefa. Este

Tabela 6 – Requisito Funcional: Manter Turma

Numero	Descrição
RF04	O professor precisa poder criar turmas , editar e manipular elementos dentro das turmas criadas por ele.
RF04.1	O professor deve ser capaz de alterar o nome da turma.
RF04.2	O professor deve ser capaz de alterar a descrição da turma.
RF04.3	O professor deve ser capaz de remover ou banir um aluno da turma.
RF04.4	O professor deve ser capaz de fazer com que a turma seja ou se torne publica ou privada.
RF04.5	O professor deve ser capaz de adicionar ou remover tarefas a turma.
RF04.6	O professor deve ser capaz de deletar uma turma somente se esta não possuir nenhum aluno, caso tenha alunos, ele deve ser capaz de torná-la indisponível.

requisito funcional foi identificado inicialmente durante a criação dos casos de uso. É necessário que o professor seja capaz de manipular os dados de uma tarefa.

Tabela 7 – Requisito Funcional: Manter Tarefa

Numero	Descrição
RF05	O professor precisa poder criar tarefas e editar, manipular ou deletar tarefas criadas por ele.
RF05.1	O professor deve ser capaz de escolher o tipo de tarefa criada por ele
RF05.2	O professor deve apresentar entrada/saída para a validação das respostas dos tipos programáveis durante a criação.
RF05.3	o professor deve ser capaz de adicionar ou remover mais entradas/saídas a uma tarefa do tipo programável.
RF05.3	Ao criar tarefas do tipo video, o professor deve apresentar uma URL do site YouTube para devido video.

A Tabela 8 representa o detalhamento do requisito funcional Matricular na Turma. Este requisito funcional foi identificado durante a fase de *brainstorming* e expandido durante a criação dos casos de uso.

Tabela 8 – Requisito Funcional: Matricular na Turma

Numero	Descrição
RF06	O professor precisa que o aluno se matricule na turma para que o mesmo tenha acesso ao conteúdo da turma.
RF06.1	Caso a turma seja privada, o professor que criou a turma deve aceitar solicitação do aluno para que o mesmo tenha acesso ao conteúdo da turma.
RF06.2	O sistema não deve permitir a matricula de alunos a turmas indisponíveis.

A Tabela 9 representa o detalhamento do requisito funcional Visualizar Desempenho. Este requisito funcional foi identificado durante a entrevista, o *stakeholder* do sistema tem a necessidade de visualizar como a turma está de desenvolvendo para que ele possa adaptar suas aulas de acordo.

Tabela 9 – Requisito Funcional: Visualizar Desempenho

Numero	Descrição
RF07	O professor precisa ser capaz de visualizar o desempenho individual do aluno e coletivo da turma.

A Tabela 10 representa o detalhamento do requisito funcional Escolher Tarefa. Este requisito funcional foi identificado durante a entrevista e aprimorado durante o *brainstorming* com a equipe de desenvolvimento do sistema.

Tabela 10 – Requisito Funcional: Escolher Tarefa

Numero	Descrição
RF08	O aluno precisa ser capaz de escolher a tarefa que ele deseja responder dentro de uma turma.
RF08.1	O Aluno somente será capaz de escolher tarefas novas sugeridas pelo sistema.
RF08.2	O Aluno deve ser capaz de responder novamente tarefas respondidas anteriormente a qualquer momento.

A Tabela 11 representa o detalhamento do requisito funcional Tipos de Tarefa. Este requisito funcional foi identificado durante a entrevista e aprimorado durante o *brainstorming* com a equipe de desenvolvimento do sistema. a necessidade de que exista vários tipos de tarefas é de extrema importância para o funcionamento do sistema. Um aluno na sala de aula deve resolver problemas não só computacionais, mas de qualquer tipo.

Tabela 11 – Requisito Funcional: Tipos de Tarefa

Numero	Descrição
RF09	O Professor precisa que existam diversos tipos de tarefas, e que futuramente possa ser adicionado mais tipos.
RF09.1	Deve existir um tipo de validação único para cada tipo de tarefa.
RF09.2	O professor precisa que um dos tipos de tarefa seja problemas computacionais programáveis.
RF09.2	O professor precisa que um dos tipos de tarefa seja assistir vídeos-aulas.

A Tabela 12 representa o detalhamento do requisito funcional Responder Tarefa. Este requisito funcional foi identificado durante o *brainstorming* com a equipe de desenvolvimento do sistema.

Tabela 12 – Requisito Funcional: Responder Tarefa

Numero	Descrição
RF10	O aluno precisa ser capaz de apresentar uma resposta para uma tarefa.
RF10.1	O aluno deve ser capaz de escolher a linguagem em que ele responderá a tarefa caso ela seja programável.
RF10.2	O aluno precisa ser capaz de programar as tarefas do tipo programáveis dentro do próprio sistema.

A Tabela 13 representa os requisitos não funcionais do sistema. Estes requisitos não funcionais foram identificados durante a fase de entrevista e *brainstorming*. Eles representam algumas das limitações que o sistema deve ter como ser um sistema web e fazer uso do servidor Apache.

Tabela 13 – Requisitos Não Funcionais

Numero	Descrição
RNF01	Facilidade de uso : O usuário do sistema deve ter facilidade de uso do sistema, ou seja, realizar tarefas com menos de 1 hora de treinamento.
RNF02	Interface web: o usuário utilizará o sistema através de um browser.
RNF03	O sistema deve ter um tempo de resposta médio inferior a 1s.
RNF04	O sistema deve prover os mecanismos de segurança necessários para a autenticação de usuários e especificação de diferentes níveis de acesso para dados e funcionalidades do mesmo.
RNF05	Usabilidade : o sistema deve ter uma interface gráfica amigável, de fácil navegação para facilitar a usabilidade por parte dos usuários.
RNF06	Servidor de Aplicação:Para os administradores do sistema, será utilizado o servidor de aplicação (Apache 2.0 ou superior) com qualquer Sistema Operacional (Windows XP ou superior ou Linux em qualquer distribuição). O backend da aplicação será desenvolvido em PHP.
RNF07	O sistema deverá estar disponível 24 horas por dia e sete dias por semana, executando-se em tempos pré-agendados pelos administradores para manutenção do mesmo.

3 Tecnologias Relacionadas

Este capítulo apresenta as tecnologias relacionadas ao ambiente virtual de aprendizagem COSMO. Desta forma, realiza-se um levantamento de softwares que possuem semelhança com o sistema proposto neste trabalho.

3.1 UVa Online Judge

Um *Online Judge* é em geral um servidor que contém descrições de problemas de diversos concursos, bem como conjuntos de dados para avaliar se uma determinada solução resolve qualquer destes problemas (REVILLA et al., 2008).

O *University of Valladolid* (UVa) Online Judge foi criado por Ciríaco Garcia de Celis (REVILLA et al., 2008). Ele era um estudante de informática quando desenvolveu a primeira versão do online judge em 1995.

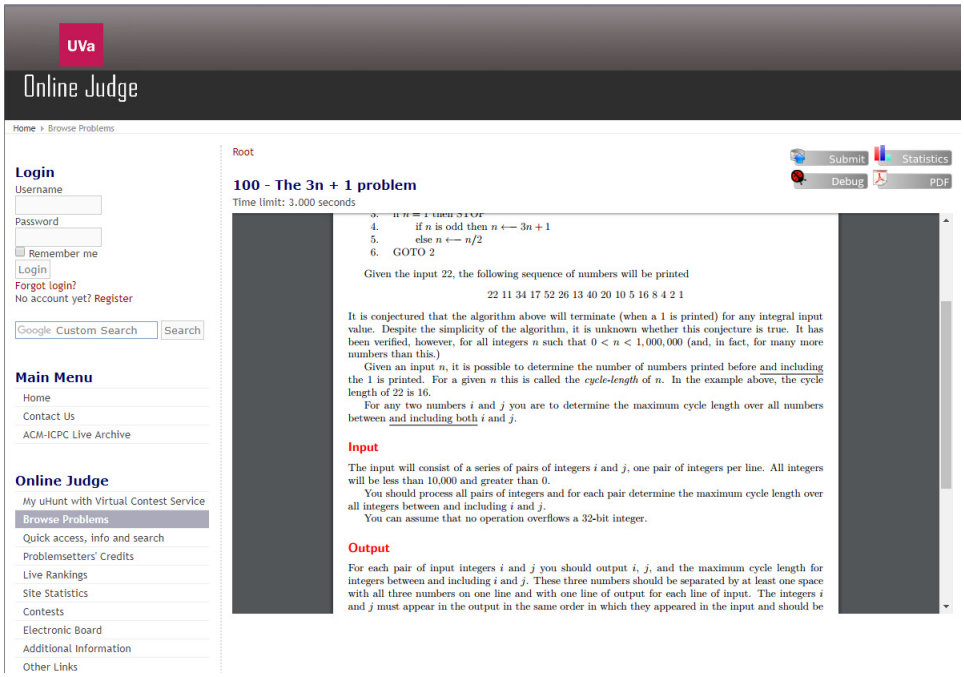
Dar as notas manualmente tem os seus problemas como, por exemplo: indentação de codigo; problema em calcular o tempo de execução; ênfase em uma solução comum; etc (KURNIA et al., 2001). Naturalmente outra maneira de se resolver esses problemas seria dar as notas automaticamente.

O *UVa Online judge* foi criado com a intenção de automatizar a avaliação de uma determinada solução para um problema (KURNIA et al., 2001). Normalmente seriam contratados vários profissionais durante uma competição para dar notas às respostas submetidas pelos alunos. Isto fica cada vez mais difícil à medida que mais pessoas se inscrevem para tais competições.

A automatização da correção fornecida pelo Uva Online Judge possui várias vantagens pois resolve grande parte dos problemas da correção manual (KURNIA et al., 2001). A indentação do código se torna irrelevante pois a correção é feita por um computador. Um computador consegue calcular o tempo de execução de uma resposta com precisão e consegue dizer se a resposta apresentada está correta com clareza mesmo se a resposta não for ortodoxa. A Figura 1 mostra como um problema é apresentado pelo *UVa Online Judge*.

Entretanto, já que o sistema foi feito para competições, se espera que o usuário final seja capaz de programar em diversas linguagens e os problemas apresentados são longe de triviais. Nesse aspecto, o sistema não apoia tão bem alunos que estão aprendendo a programar. Ele é mais indicado para melhorar as habiidades de quem já é programador em tópicos avançados.

Figura 1 – Interface do UVa Online Judge para apresentar problemas



Fonte: https://uva.onlinejudge.org/index.php

3.2 URI Online Judge Academic

O URI Online Judge é um projeto que vem sendo desenvolvido na URI - Universidade Regional Integrada. Trata-se de um portal que contém problemas no estilo do ICPC (*International Collegiate Programming Contest*) da ACM e que oferece ao usuário um juiz online para testar suas soluções a estes problemas (SELIVON et al., 2015).

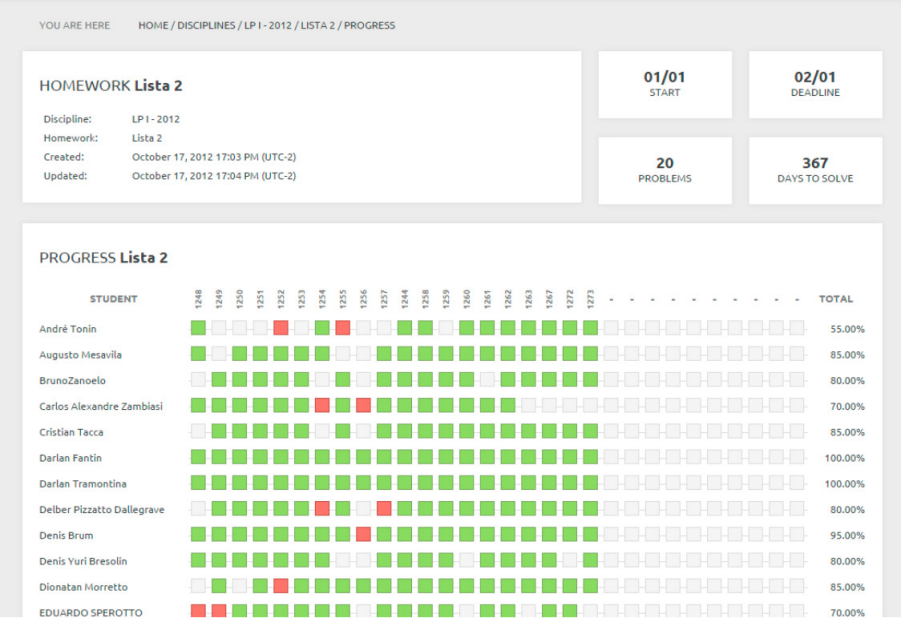
O objetivo principal da criação do portal foi desenvolver funcionalidades que oferecessem uma alternativa ao método tradicional de ensino de algoritmos (SELIVON et al., 2015).

A ferramenta possui todas as características básicas de um portal de programação como UVa, tal como: correção em tempo real, alta disponibilidade e aceita diferentes tipos de linguagens de programação.

Para que os professores e coaches de equipes de programação pudessem acompanhar o rendimento de seus estudantes foi criado em 2013 um novo módulo denominado Academic (SELIVON et al., 2015). O novo ambiente é integrado ao portal e permite o controle de listas de exercícios e de estudantes com o acompanhamento em tempo real pelo professor.

No módulo *academic*, o professor registra suas disciplinas e cadastra em cada uma delas os exercícios que comporão listas de exercícios a serem liberadas para resolução pelos alunos. As disciplinas criadas podem ser facilmente visualizadas, editadas ou excluídas.

Figura 2 – Interface Uri Online Judge Academic para Professores



Fonte: (SELIVON et al., 2015)

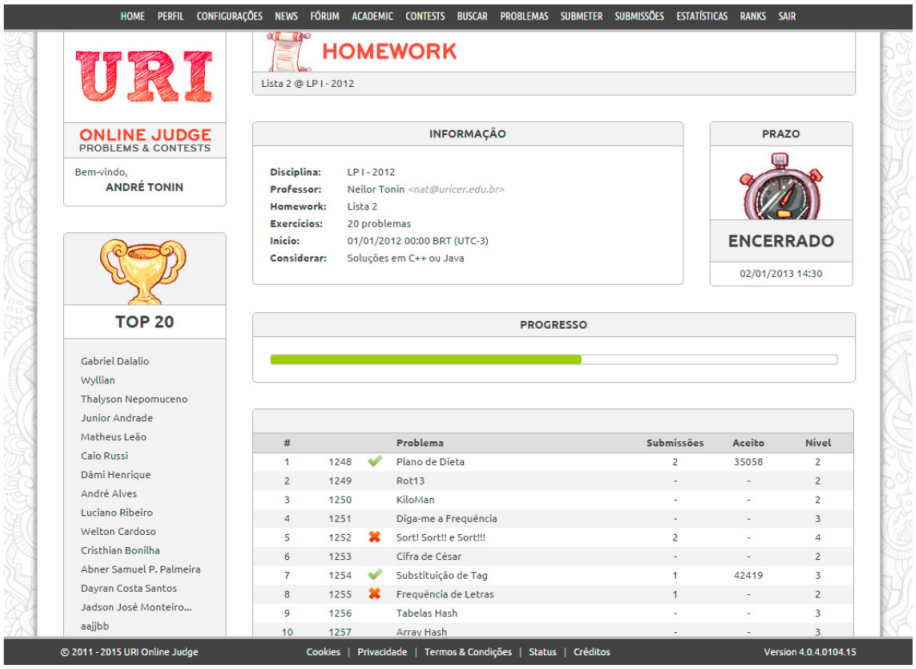
Durante a criação destas listas é possível para o professor estabelecer a linguagem que deve ser utilizada para resolução do exercício, além de escrever instruções para os alunos e delimitar um prazo para a resolução dos exercícios. Os exercícios das listas podem ser selecionados a partir da lista oferecida pelo *URI Online Judge*.

Após criar a lista, o professor deve enviar um convite para os alunos comporem a turma. Assim que o convite for aceito ele poderá observar o avanço individual da turma a partir de uma apresentação percentual e da marcação dos exercícios já resolvidos como apresentado na Figura 2.

Para ter acesso aos exercícios, o aluno precisa aceitar o convite do professor através do sistema. Ao aceitar, ele terá acesso às disciplinas onde estarão as listas e exercícios que deverá fazer.

O andamento da lista pode ser facilmente visualizado pelo aluno por uma barra de progresso. A lista, além de outras informações, apresenta a identificação do problema, o nível de dificuldade e o prazo para conclusão com contagem regressiva de tempo como mostrado na figura 3.

Figura 3 – Interface Uri Online Judge Academic para alunos



Fonte: (SELIVON et al., 2015)

3.3 BOCA

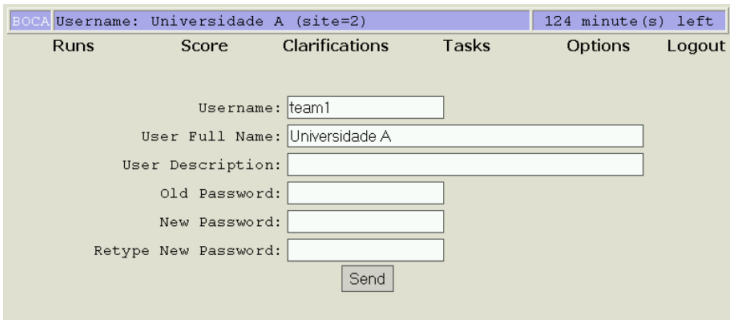
O BOCA é um sistema de apoio a competições de programação desenvolvido para uso em maratonas promovidas pela Sociedade Brasileira de Computação (FRANÇA et al., 2011). Ele oferece suporte online para a competição, gerenciando times de alunos e juízes, permitindo a proposição de problemas de programação bem como a submissão e avaliação automática de soluções dos problemas.

O BOCA pode ser dividido em cinco partes de acordo com a especificidade de cada usuário: time, juiz, administrador, *staff* e placar (CAMPOS; FERREIRA, 2004). Cada um tem interface própria de acordo com as necessidades e responsabilidades de cada perfil.

Os times possuem uma interface simplificada, onde é possível enviar os programas para correção, alterar informações do time, submeter dúvidas aos juízes e verificar o placar online da competição. A Figura 4 mostra a interface do BOCA para os times.

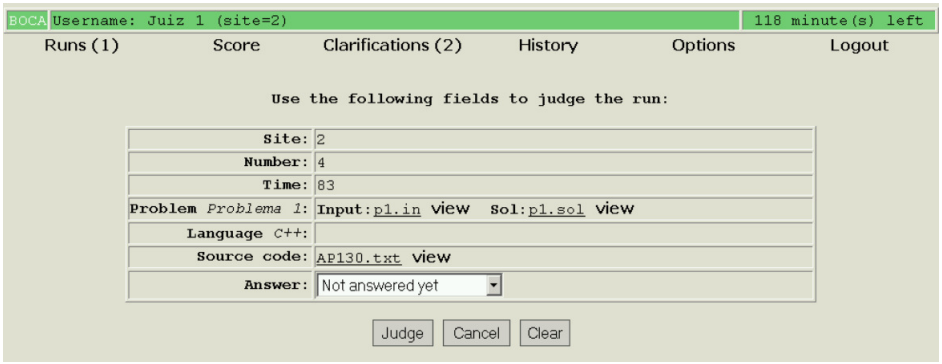
A interface do juiz permite responder perguntas e dúvidas de forma individual ou coletiva, além de permitir ao juiz criar perguntas globais para serem respondidas por todos. O juiz também pode analisar e corrigir os programas enviados pelos times, definindo qual a resposta o time responderá por sua tentativa. O juiz também pode acompanhar o placar online da competição no seu próprio computador ou alterar os dados, como nome, senha, etc. A Figura 5 mostra a interface do BOCA para os times.

Figura 4 – Interface dos times no BOCA



Fonte: (CAMPOS; FERREIRA, 2004)

Figura 5 – Interface dos Juízes no BOCA



Fonte: (CAMPOS; FERREIRA, 2004)

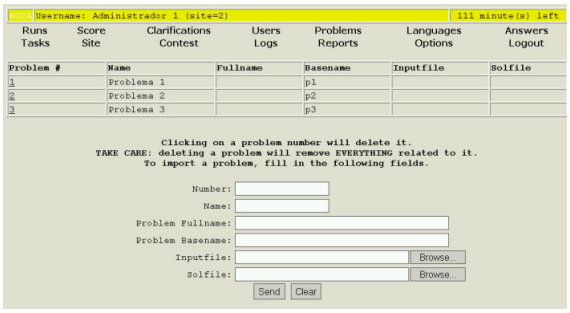
A interface do administrador é mais complicada, ela permite a configuração das informações da competição, sobre as linguagens de programação permitidas e as questões da prova. É atraves dessa interface que é feita a inclusão dos times, juízes, placares, etc. São permitidas diversos dados das competições, incluindo submissões, programas, perguntas, controle de acesso, etc. A Figura 6 mostra a interface do BOCA para os administradores.

As principais funções dos *staff* durante a prova são: controlar o acesso e movimentação dos times no local da prova; imprimir arquivos solicitados pelos times; entregar balões para cada time que acerta um problema; dar assistência a times com problemas de hardware ou software (CAMPOS; FERREIRA, 2004).

Apesar de todas essas funções precisarem de uma ação física, o controle sobre os pedidos de impressões, entrega de balões e pedidos de assistências podem ser feitas pelo BOCA, assim é disponibilizado para os *staff* uma interface capaz de alterar seus dados, verificar o placar durante a prova, etc. A Figura 7 mostra a interface do boca para os *staffs*.

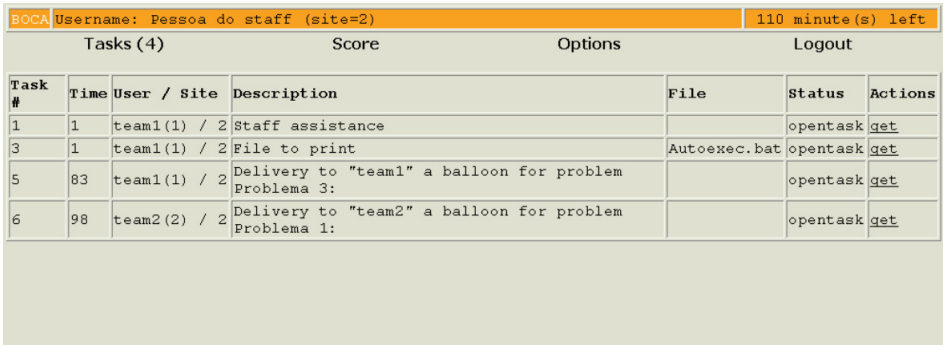
Por fim o objetivo do placar é poder disponibilizar pela rede o resultado em tempo

Figura 6 – Interface dos Administradores no BOCA



Fonte: (CAMPOS; FERREIRA, 2004)

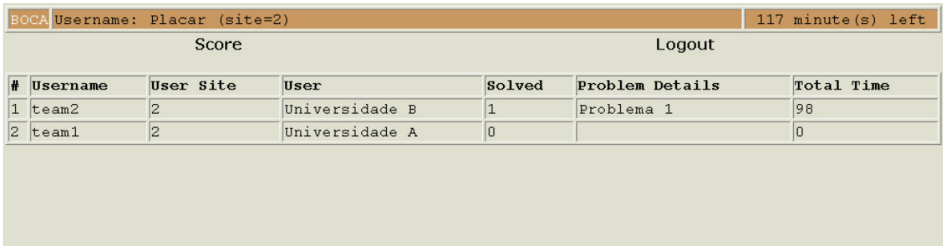
Figura 7 – Interface dos *Staffs* no BOCA



Fonte: (CAMPOS; FERREIRA, 2004)

real da competição para as pessoas que não estão diretamente envolvidas com ela. a Figura 8 mostra a interface do boca para o placar.

Figura 8 – Interface dos Placar no BOCA

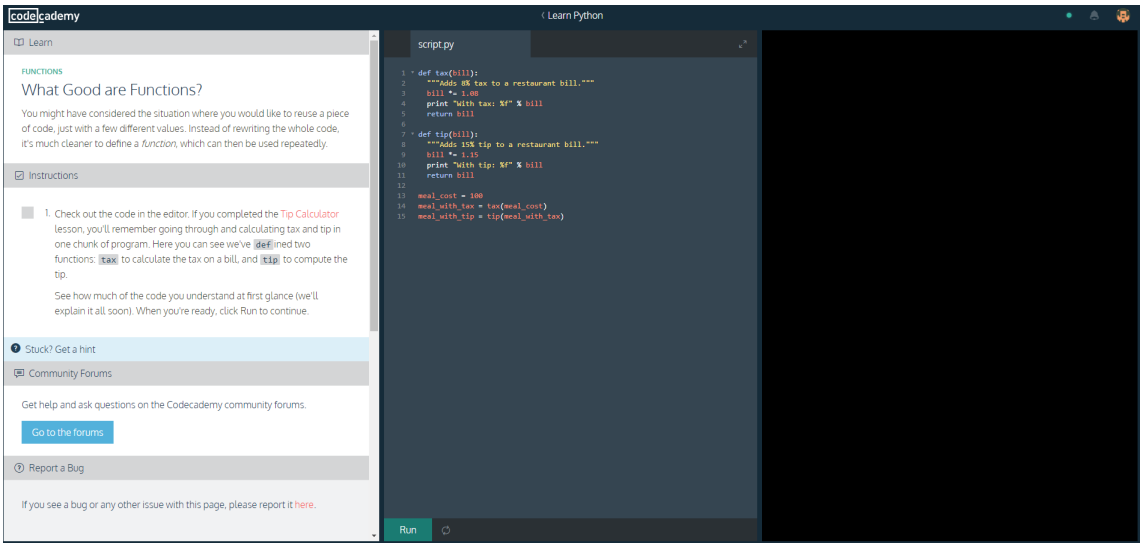


Fonte: (CAMPOS; FERREIRA, 2004)

3.4 Codecademy

Codecademy é uma plataforma e-learning para o aprendizado de programação projetada utilizando gamificação. Nela, os cursos de programação são organizados em seções que possuem sequências de exercícios, Cada exercício possui um texto para introduzir o tema e instruções que mostram ao estudante o que fazer, alem de possíveis dicas para ajudar o estudante caso ele não consiga achar uma solução. A Figura 9 apresenta um exemplo de interface para os exercícios.

Figura 9 – Exemplo de Interface do Codecademy para exercícios



Fonte: https://www.codecademy.com/

Os alunos são recompensados com *badges* quando atingem determinado número de pontos, completarem certas aulas ou cursos , ou algum pré-requisito estabelecido pelo Codecademy como uma sequência de dias usando o sistema.

Entretanto o professor não tem a possibilidade de acompanhar o progresso do estudante pela plataforma. O sistema não é sensível ao contexto e não se adapta ao usuário para auxiliar no aprendizado do estudante.

4 Modelagem UML

A UML (*Unified Modeling Language* - Linguagem de modelagem unificada) é um padrão para descrever projetos de software (PRESSMAN, 1995). A UML pode ser usada para visualizar, especificar e construir os artefatos de um sistema.

A UML fornece 13 diferentes diagramas para o uso da modelagem de software(PRESSMAN, 1995). Entretanto pode ser usado apenas alguns dos diagramas a fim de evitar o congestionamento dos diagramas com detalhes irrelevantes. Este trabalho faz uso de quatro diagramas: diagrama de caso de uso; diagrama de classe; diagrama de sequência; e diagrama de atividade.

4.1 Diagrama de Caso de Uso

O diagrama de casos de uso procura, por meio de uma linguagem simples, possibilitar a compreensão do comportamento externo do sistema por qualquer pessoa, tentando apresentar o sistema através da perspectiva do usuário (GUEDES, 2009).

Ele é entreos diagramas da UML o mais abstrato, sendo assim o mais maleável e informal. O diagrama de uso é amplamente usado para apoiar o levantamento de requisitos(SOMMERVILLE et al., 2003), embora venha ser consultado durante todo o processo de engenharia e sirva como a base para a modelagem de outros diagramas.

Esse diagrama é de grande auxílio para a compreensão dos requisitos do sistema, ajudando a visualizar, especificar e documentar as características e serviços do sistema que o usuário deseja, além de identificar os tipos de usuário que irão interagir com o sistema, quais papeis esses usuários irão assumir e quais funções cada usuário poderá requisitar.

O diagrama de caso de uso concentra-se em dois itens principais: atores e casos de uso. Os atores representam os papéis desempenhados pelos diversos usuários que poderão utilizar os serviços e funções do sistema (SOMMERVILLE et al., 2003). Eventualmente, um ator pode representar algum hardware especial ou mesmo outro software que interaja com o sistema, como no caso de um agente de software ou um sistema integrado (GUEDES, 2009). A Figura 10 apresenta um exemplo de ator.

Os casos de uso referem-se aos serviços, tarefas ou funcionalidades que podem ser utilizados de alguma maneira pelos atores que interagem com o sistema, sendo utilizados para expressar e documentar os comportamentos pretendidos para as funções deste (GUEDES, 2009). Os casos de uso são representados por elipses. A Figura 11 apresenta um exemplo de caso de uso representando a ação criar turma.

Figura 10 – Exemplo de um Ator chamado Professor

Figura 11 – Exemplo de um Caso de Uso chamado Criar Turma

Diagramas de casos de uso dão uma visão simples de uma interação. Logo, é fundamental fornecer mais detalhes para entender o que está envolvido (SOMMERVILLE et al., 2003).Os casos de uso costumam ser documentados, fornecendo instruções de como será seu funcionamento, o que deverá ser executado e qual evento forçará sua execução, quais atores poderão utilizá-los e quais suas possíveis restrições.

O diagrama apresentado na Figura 12 e documentação apresentados da Tabela 14 a Tabela 26 tem por objetivo apresentar uma visão externa geral das funcionalidades da plataforma de ensino COSMO, sem se preocupar com a questão de como tais funcionalidades serão implementadas.

Tabela 14 – Caso de Uso : Realizar Login

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas por um aluno ou professor para realizar o login no sistema.
Atores	Aluno e Professor
Pré-condições	
Pós-condições	
Ações do Ator	O ator deve informar o seu usuário e senha para o sistema.
Ações do Sistema	Verificar se a senha e usuário apresentados estão corretos.
Restrições	Caso o usuário não tenha login e senha no sistema ele deverá se auto-registrar.

Figura 12 – Diagrama de Caso de Uso Geral do COSMO

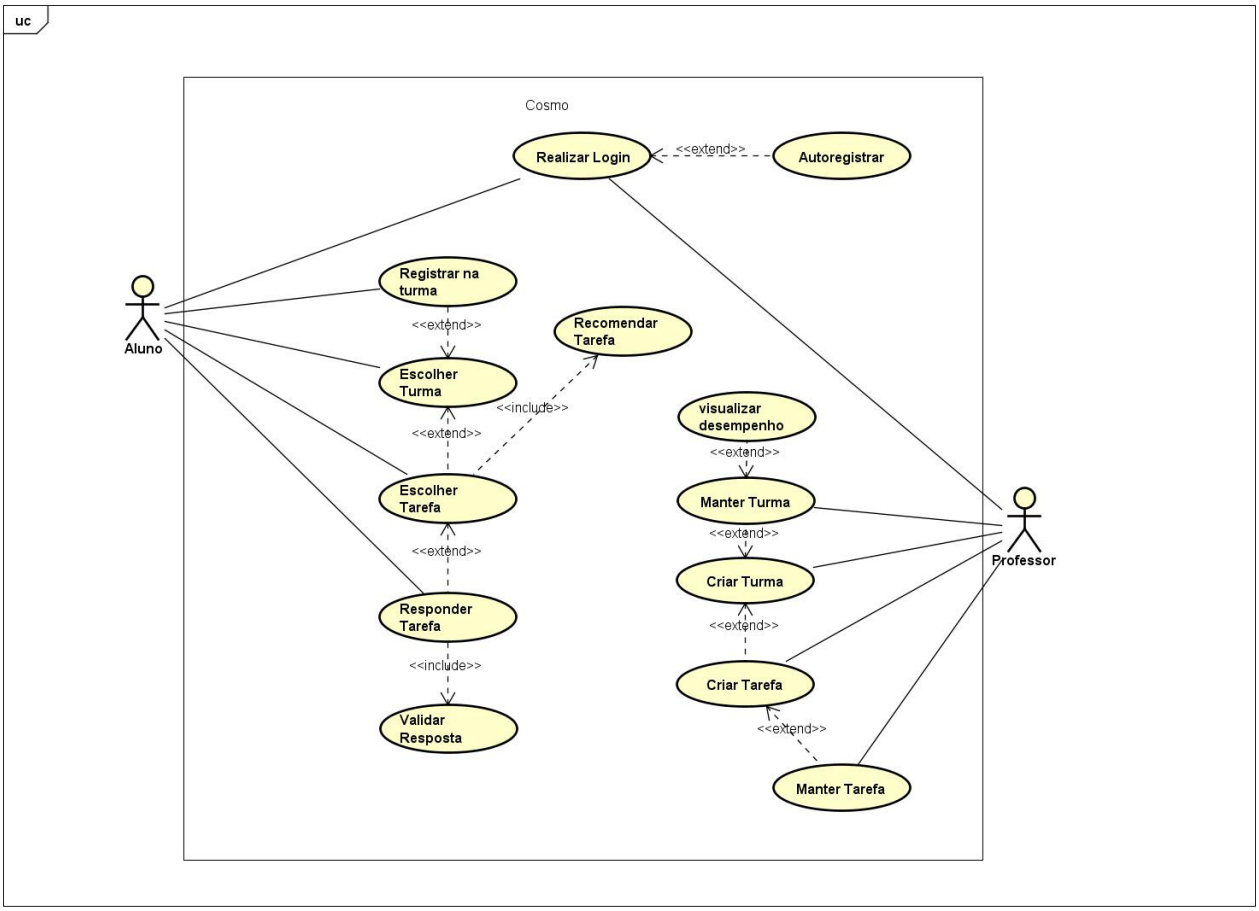


Tabela 15 – Caso de Uso : Auto-registrar

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas por um aluno para se registrar no sistema.
Atores	Aluno
Pré-condições	
Pós-condições	
Ações do Ator	O ator deve informar o seu usuário e senha para o sistema e e-mail para cadastro.
Ações do Sistema	Verificar se já existem usuários ou emails com mesmo nome cadastrados,caso não, criar cadastro.
Restrições	Um Professor não pode se cadastrar no sistema.

Tabela 16 – Caso de Uso : Registrar na turma

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas por um aluno para se registrar em uma turma.
Atores	Aluno
Pré-condições	Ter selecionado uma turma.
Pós-condições	
Ações do Ator	O ator clica no botão "registrar".
Ações do Sistema	Adiciona o aluno na lista de alunos registrados.
Restrições	Caso a turma seja privada, uma mensagem sera enviada para o professor da turma e o aluno somente terá acesso a turma depois que o professor confirmar o registro.

Tabela 17 – Caso de Uso : Escolher Turma

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas por um aluno para escolher uma turma.
Atores	Aluno
Pré-condições	Estar logado no sistema.
Pós-condições	
Ações do Ator	O aluno seleciona uma turma em uma lista de turmas disponíveis.
Ações do Sistema	mostrar para o aluno o conteúdo da turma.
Restrições	O sistema não deve mostrar o conteúdo para o aluno caso ele não esteja registrado na turma e deve apresentar a opção de se registrar.

Tabela 18 – Caso de Uso : Escolher Tarefa

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas por um aluno para escolher uma tarefa.
Atores	Aluno
Pré-condições	Estar registrado na turma que em que está a tarefa.
Pós-condições	Visualiza as informações da tarefa .
Ações do Ator	O ator seleciona uma tarefa em uma lista de tarefa recomendadas.
Ações do Sistema	Mostrar para o ator o conteúdo da tarefa.
Restrições	

Tabela 19 – Caso de Uso : Recomendar Tarefa

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas pelo aluno para que o sistema recomende tarefas .
Atores	Aluno
Pré-condições	Estar em uma turma.
Pós-condições	
Ações do Ator	Acessar uma turma.
Ações do Sistema	Apresentar 4 opções de tarefas para serem respondidas baseado no que o aluno respondeu na turma.
Restrições	Caso o aluno não queira responder uma tarefa, ele pode pular ela e o sistema deve sugerir outra tarefa no lugar.

Tabela 20 – Caso de Uso : Responder Tarefa

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas por um aluno para responder uma tarefa.
Atores	Aluno
Pré-condições	Estar matriculado na turma
Pós-condições	Receber resultado da sua resposta.
Ações do Ator	O aluno apresenta uma resposta para a tarefa apresentada.
Ações do Sistema	Validar a resposta do aluno e mostrar resultado
Restrições	Caso a resposta do aluno esteja incorreta , mostrar opção para tentar novamente.

Tabela 21 – Caso de Uso : Validar Resposta

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas pelo aluno para que o sistema valide uma resposta apresentada a uma tarefa.
Atores	Aluno
Pré-condições	Ter respondido uma tarefa
Pós-condições	Confirmar a resposta como certo ou errado.
Ações do Ator	O Aluno responde uma tarefa.
Ações do Sistema	Verificar se a resposta apresentada pelo aluno para a tarefa é correta ou não.
Restrições	

Tabela 22 – Caso de Uso : Criar Turma

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas por um professor para criar uma turma.
Atores	Professor
Pré-condições	Estar logado no sistema.
Pós-condições	Turma será criada.
Ações do Ator	Informar os atributos básicos de uma turma, como nome, matéria, etc.
Ações do Sistema	Verificar se não já existe uma turma com o mesmo nome e criar a turma.
Restrições	Caso já exista uma turma com o mesmo nome, comunicar ao professor e impedir a criação da turma.

Tabela 23 – Caso de Uso : Manter Turma

Realizar Login	
Objetivo	Este caso de uso descreve as possíveis atividades de manutenção do cadastro de uma turma, ou seja, permite incluir, alterar ou consultar turmas.
Atores	Professor
Pré-condições	Ter criado uma turma..
Pós-condições	Turma será criada.
Ações do Ator	Escolher uma turma, e se necessário,alterar ou excluir os dados da turma.
Ações do Sistema	Apresentar os dados da turma.
Restrições	

Tabela 24 – Caso de Uso : Criar Tarefa

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas por um professor para criar uma tarefa .
Atores	Professor
Pré-condições	Ter criado uma turma.
Pós-condições	Turma será criada.
Ações do Ator	Informar os atributos básicos de uma tarefa, como nome, tipo, entrada e saida, etc.
Ações do Sistema	Registrar tarefa.
Restrições	

Tabela 25 – Caso de Uso : Manter Tarefa

Realizar Login	
Objetivo	Este caso de uso descreve as possíveis atividades de manutenção do cadastro de uma Tarefa, ou seja, permite incluir, alterar ou consultar Tarefa.
Atores	Professor
Pré-condições	Ter criado uma tarefa.
Pós-condições	.
Ações do Ator	Escolher uma tarefa, se necessário,alterar ou excluir os dados da turma.
Ações do Sistema	Apresentar dados da Tarefa.
Restrições	

Tabela 26 – Caso de Uso : Visualizar Desempenho

Realizar Login	
Objetivo	Este caso de uso descreve as etapas percorridas para um professor visualizar o desempenho da turma.
Atores	Professor
Pré-condições	Ter criado uma turma.
Pós-condições	
Ações do Ator	Clicar na aba de desempenho da turma.
Ações do Sistema	Mostrar dados pertinentes ao desempenho da turma.
Restrições	

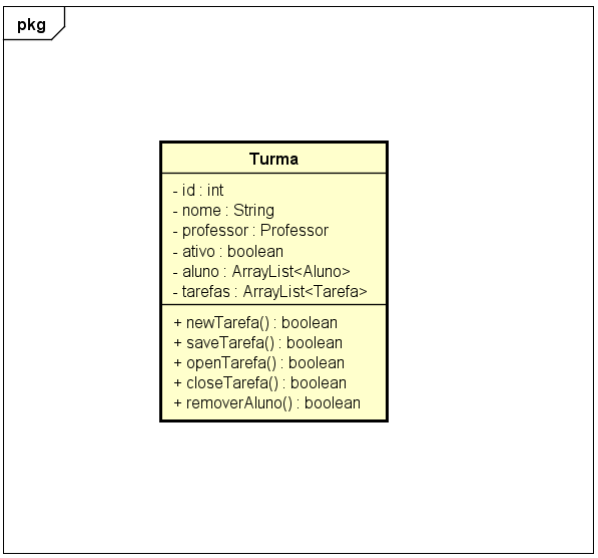
4.2 Diagrama de Classe

O diagrama de classe é um dos mais importantes e mais utilizados da UML. Seu principal enfoque está em permitir a visualização das classes que irão compor o sistema com seus respectivos atributos e métodos, bem como em demonstrar como as classes do diagrama se relacionam, complementam e transmitem informações entre si (GUEDES, 2009). Este diagrama apresenta uma visão estática das classes, preocupando-se em definir a estrutura lógica das mesmas (GUEDES, 2009). O diagrama de classe serve ainda como base para a construção da maioria dos outros diagramas da linguagem UML.

O diagrama de classe é composto por suas classes e pelas associações existentes entre elas, ou seja, o relacionamento entre as classes (GUEDES, 2009). Classes possuem atributos que armazenam os dados dos objetos das classes, e métodos que são funções que uma instancia da classe pode executar (PRESSMAN, 1995). Os valores dos atributos são variáveis, possibilitando assim identificar cada objeto individualmente, diferentemente dos métodos que são idênticos para todas as instâncias de uma classe especifica.

Embora os métodos sejam declarados no diagrama de classe, identificando os possíveis parâmetros e retornos, o diagrama de classe não se preocupa em definir as etapas que tais métodos deverão percorrer quando forem chamados (GUEDES, 2009). A figura 13 apresenta um exemplo de classe contendo atributos e métodos.

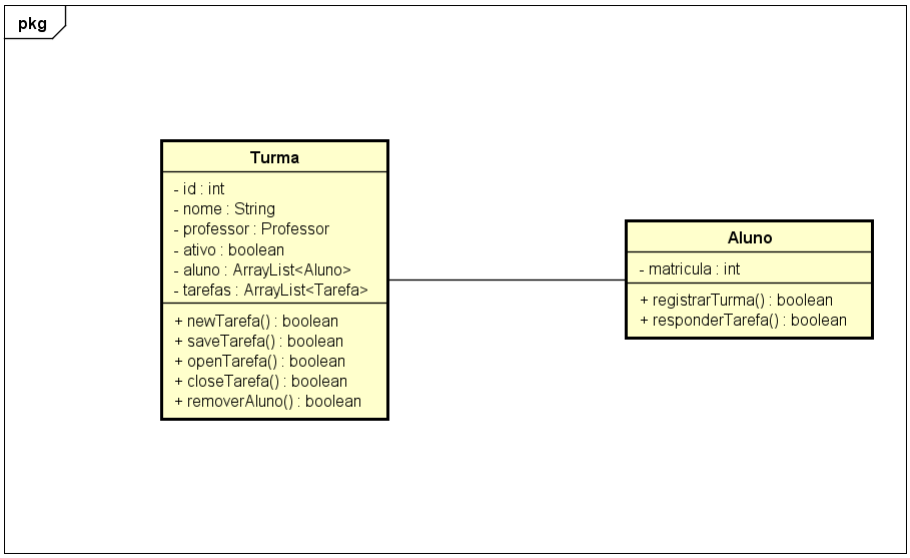
Figura 13 – Exemplo de classe



As classes costumam ter relacionamentos entre si, chamados associações, que permitem que elas compartilhem informações entre si e colaborem para a execução dos processos executados pelo sistema(GUEDES, 2009). uma associação entre duas classes

indica que há relação estrutural entre elas (PRESSMAN, 1995). A Figura 14 apresenta um exemplo de associação entre duas classes.

Figura 14 – Exemplo de associação



Os diagrama apresentados na Figura 15 tem por objetivo apresentar uma visão das classes que irão compor o COSMO com seus atributos, além de demostrar como as classes se relacionam e transmitem informações entre si, preocupando-se em definir a estrutura lógica das mesmas.

Tabela 27 – Descrição da classe Usuário

Classe	Usuário
Descrição	A classe Usuário armazena as informações gerais dos usuários, tanto aluno quanto professor. Ela é uma superclasse abstrata(por isso seu nome está em itálico) e portanto não interage diretamente com a classe Turma. As classes que interagem com a classe Turma são suas subclasses Professor e Aluno que herdam todos os atributos da classe pessoa e ainda tem seus próprios atributos particulares. Os atributos da classe Usuário são nome, sobrenome, e-mail,login e senha tipo String e id do tipo Int.

O diagrama apresentado na Figuras 16 e as Tabelas 36 à Tabela 40 referem-se ao domínio da solução, incrementando o modelo conceitual e detalhando métodos, navegabilidade, etc.

Figura 15 – Diagrama de Classe - Modelo Conceitual do COSMO

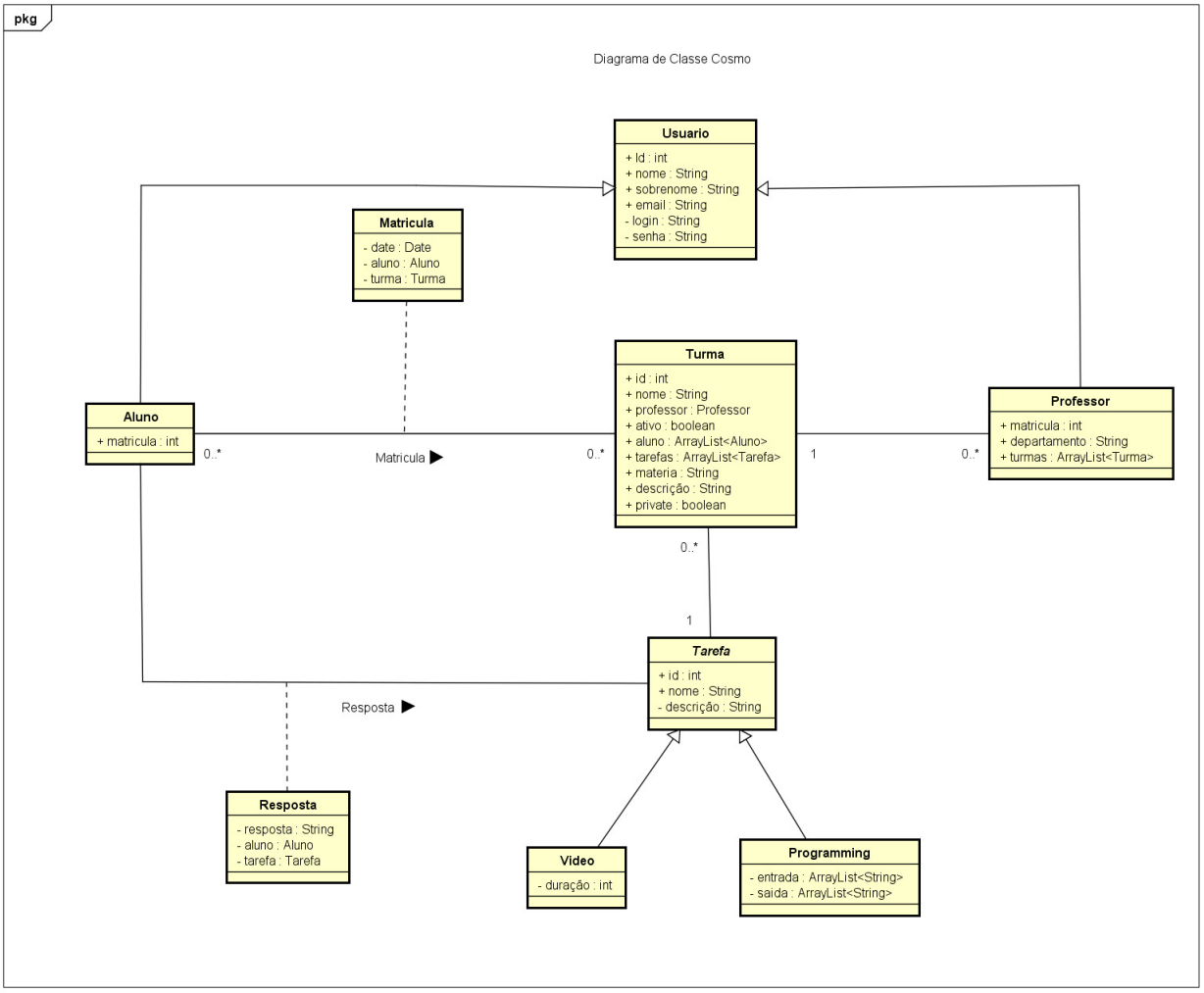


Tabela 28 – Descrição da classe Aluno

Classe	Aluno
Descrição	A classe Aluno é uma subclasse derivada da classe Usuário, representado os alunos que estão registrados no sistema. Esta classe herda todos os atributos da sua superclasse, tendo o atributos extra matricula do tipo int.

Tabela 29 – Descrição da classe Professor

Classe	Professor
Descrição	A classe Professor também é uma subclasse derivada da classe Usuário, representando os professores no sistema. Da mesma maneira, a classe professor herda todos os atributos da classe Usuário, tendo como atributos extras matricula do tipo int, departamento do tipo string, e turmas do tipo ArrayList<Turma>.

Tabela 30 – Descrição da classe Turma

Classe	Turma
Descrição	A classe Turma representa, como o nome diz, uma turma criada por um professor, seu objetivo é armazenar as informações gerais de uma turma, como nome, professor, lista de alunos e lista de tarefas. A classe Turma possui os atributos id do tipo Int, nome do tipo String, professor do tipo Professor, ativo do tipo boolean, aluno do tipo ArrayList<Aluno> e tarefas do tipo ArrayList<Tarefa> , alem de matéria do tipo string,descrição e private do tipo string.

Tabela 31 – Descrição da classe Tarefa

Classe	Tarefa
Descrição	A classe Tarefa armazena as informações gerais das tarefas. Ela é uma superclasse abstrata e portanto não interage diretamente com a classe Turma. As classes que interagem com a classe Turma são suas subclasses Video e Programming que herdam todos os atributos da classe tarefa e ainda tem seus próprios atributos particulares. A classe Tarefa possui os atributos id do tipo Int, descrição e nome do tipo string.

Tabela 32 – Descrição da classe Programming

Classe	Programming
Descrição	A classe Programming é uma subclasse derivada da classe Tarefa, representado um tipo de tarefa que está implementados no sistema. Esta classe herda todos os atributos da sua superclasse, tendo o atributos extra entrada do tipo ArrayList<String> e saida do tipo ArrayList<String>.

Tabela 33 – Descrição da classe Video

Classe	Video
Descrição	Esta classe também é uma subclasse derivada da classe Tarefa, representando outro tipo de tarefa que está implementado no sistema. Da mesma maneira, a classe professor herda todos os atributos da classe Tarefa, tendo como atributos extras a duração do tipo int.

Tabela 34 – Descrição da classe Matricula

Classe	Matricula
Descrição	Matricula representa uma classe associativa entre Alunos e Turmas, estas classes são necessárias nos casos em que existem atributos relacionados a associação que nãoo podem ser armazenados por nenhuma das classes envolvidas.ela tem como objetivo armazenar a data em que o aluno se matriculou na turma e serve para representar a matricula do mesmo. Ela possui os atributos resposta do tipo Date, aluno do tipo Aluno e turma do tipo Turma.

Tabela 35 – Descrição das classe Resposta

Classe	Resposta
Descrição	Resposta representa outra classe associativa , desta vez entre Alunos e Tarefa, estas classes são necessárias nos casos em que existem atributos relacionados a associação que não podem ser armazenados por nenhuma das classes envolvidas. Ela tem como objetivo armazenar a resposta oferecida pelo aluno como solução para a tarefa em questão e só é instanciada caso a reposta esteja correta. Ela possui os atributos resposta do tipo String, aluno do tipo Aluno e tarefa do tipo Tarefa.

Figura 16 – Diagrama de Classe - Modelo de Domínio do COSMO

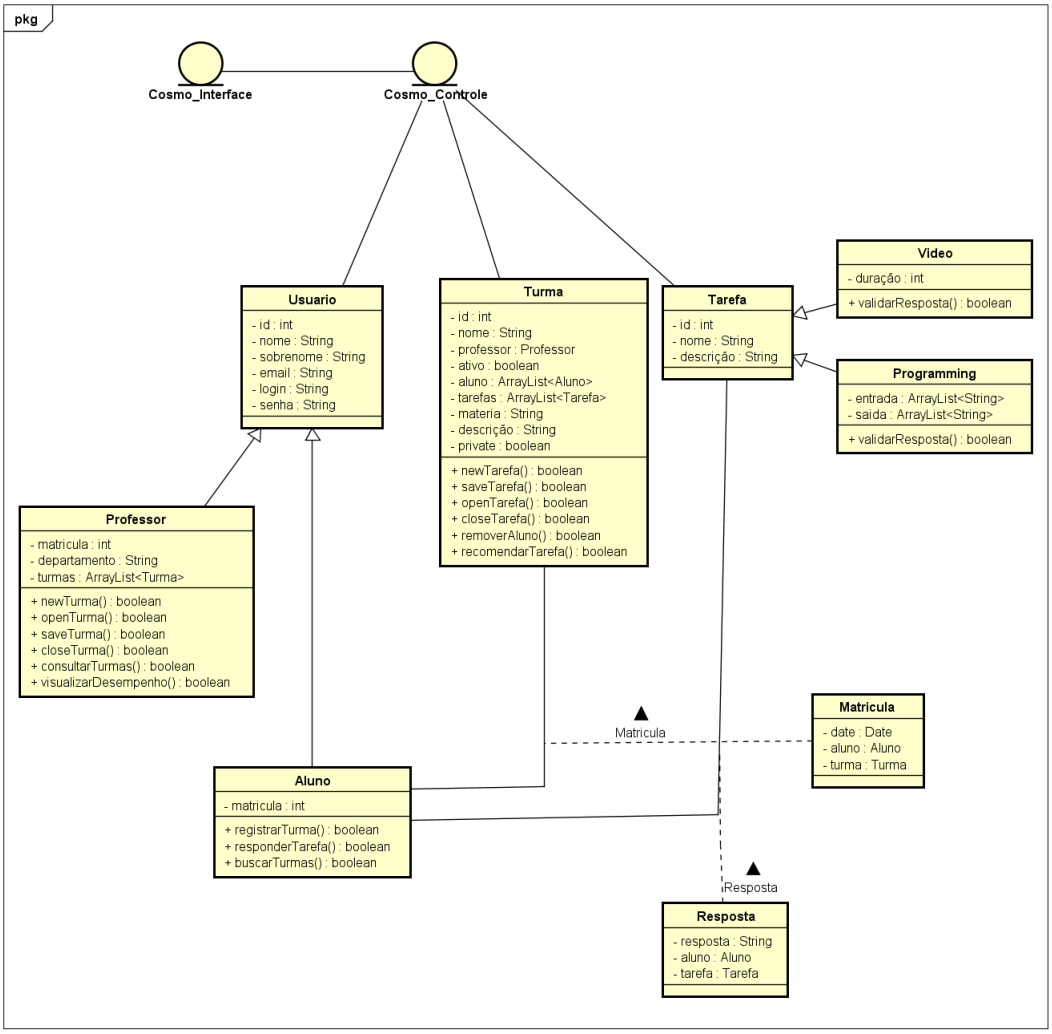


Tabela 36 – Métodos da classe Aluno

Método	registrarTurma
Descrição	A função deste método é registrar o aluno em uma turma. Ele recebe um inteiro sendo o id da turma e cria uma instancia da classe associativa matricula, tendo a data que foi registrada, o aluno e a turma, alem de adicionar o aluno a lista de alunos da turma,retornando true se tiver sucesso e false caso contrario.
Método	responderTarefa
Descrição	A função deste método é iniciar o processo para o aluno responder uma tarefa. Ele recebe uma string e a tarefa , sendo essa string a resposta apresentado pelo aluno, logo em seguida é chamado o método validarResposta que a tarefa em questão possui, se retornar 1 é criado uma instancia da classe resposta, e retorna true, caso contrario o método retorna false.

Tabela 37 – Métodos da classe Professor

Método	newTurma
Descrição	Tem por objetivo criar uma nova turma, ela recebe todos os atributos nécessarios para instanciar uma Turma e retorna true caso tenha sucesso e false caso contrário.
Método	openTurma
Descrição	Tem por objetivo abrir o conteúdo de uma turma, ela recebe o id de uma turma e recebe de volta o objeto Turma.
Método	saveTurma
Descrição	Tem por objetivo atualizar dados de uma turma, ela recebe todos os atributos necessários para instanciar uma Turma e atualiza seus valores, retorna true caso tenha sucesso e false caso contrário.
Método	closeTurma
Descrição	Tem por objetivo tornar uma turma inativa, uma turma não pode ser deletada para manter o histórico do sistema, o método recebe o id da turma e altera o valor de ativo para false, retorna true caso tenha sucesso e false caso falhe.
Método	visualizarDesempenho
Descrição	Tem por objetivo visualizar o desempenho geral da turma, o método recebe o id da turma e retorna os dados de maneira organizados mostrando o desempenho da turma.

Tabela 38 – Métodos da classe Programming

Método	validarResposta
Descrição	Este método tem por resposta validar a resposta apresentada pelo aluno, cada subclasse de Tarefa possui seu próprio método para validar a resposta apresentada, facilitando assim que outras subclasses sejam criadas futuramente.

Tabela 39 – Métodos da classe Video

Método	validarResposta
Descrição	Este método tem por resposta validar a resposta apresentada pelo aluno, cada subclasse de Tarefa possui seu próprio método para validar a resposta apresentada.

Tabela 40 – Métodos da classe Turma

Método	newTarefa
Descrição	Tem por objetivo criar uma nova tarefa, o método recebe todos os atributos necessários para instanciar uma tarefa e retorna true caso tenha sucesso e false caso contrário.
Método	openTarefa
Descrição	Tem por objetivo abrir o conteúdo de uma tarefa, ela recebe o id de uma tarefa e recebe de volta o objeto Tarefa.
Método	saveTarefa
Descrição	Tem por objetivo Atualizar dados de uma Tarefa. Ela recebe todos os atributos nécessarios para instanciar uma Tarefa e atualiza seus valores, retorna true caso tenha sucesso e false caso contrário.
Método	closeTarefa
Descrição	Tem por objetivo deletar uma tarefa. O método recebe o id da tarefa e a deleta da lista de tarefas, retorna true caso tenha sucesso e false caso contrário.
Método	removerAluno
Descrição	Tem por objetivo remover um aluno de uma turma. O método recebe o id do aluno e o deleta da lista de alunos, retorna true caso tenha sucesso e false caso contrário.
Método	recomendarTarefa
Descrição	Tem por objetivo recomendar tarefas para um aluno. O método recebe o id do aluno e retorna uma lista de tarefas com 4 tarefas de acordo com as questões respondidas pelo aluno anteriormente na turma.

4.3 Diagrama de Sequência

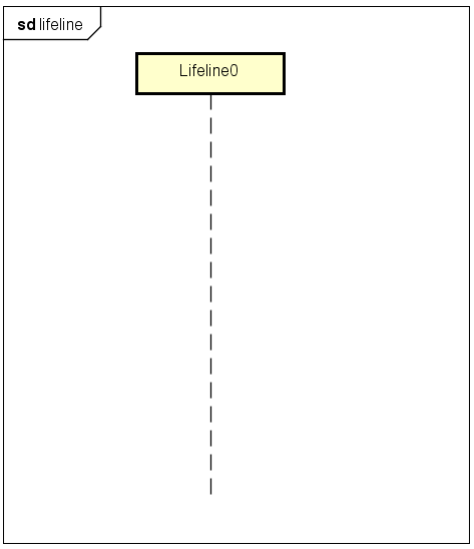
O diagrama de sequência é utilizado para identificar as comunicações dinâmicas entre objetos durante a execução de uma tarefa (PRESSMAN, 1995). Assim, determinar a ordem em que os eventos ocorrem, quais mensagens que são enviadas, os métodos que são chamados e como os objetos interagem dentro de um determinado processo (GUEDES, 2009).

O diagrama de sequência baseia-se no diagrama de caso de uso, havendo normalmente um diagrama de sequência para cada caso de uso declarado, uma vez que um caso de uso, em geral, refere-se a um processo disparado por um ator (GUEDES, 2009). Ele depende também do diagrama de classes, já que as classes dos objetos utilizados no diagrama de sequência estão descritas nele.

O diagrama é composto principalmente por atores e *lifelines*. Os atores são exatamente os mesmos descritos no diagrama de caso de uso, ou seja, entidades externas que interagem com o sistema e que solicitam serviços gerando, assim, eventos que iniciam processos.

Um *lifeline* é um participante individual em uma interação e representa a linha do tempo da instância ou objeto (PRESSMAN, 1995). Na maioria das vezes, um *lifeline* irá se referir a uma instância de uma classe. Um *lifeline* pode existir desde o início do processo ou ser criado durante o decorrer da execução do mesmo (GUEDES, 2009). A Figura 17 apresenta um exemplo de *lifeline*.

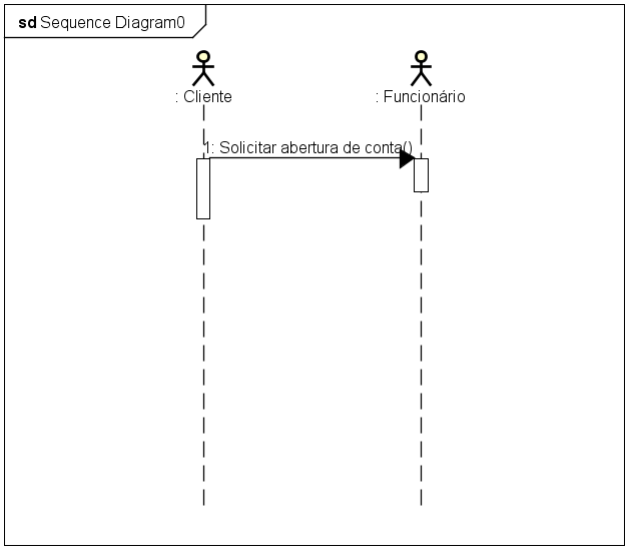
Figura 17 – Exemplo de Lifeline



Além de atores e *lifelines*, existem as mensagens. Elas são utilizadas para demonstrar a ocorrência de eventos que normalmente forçam a chamada de um método em algum

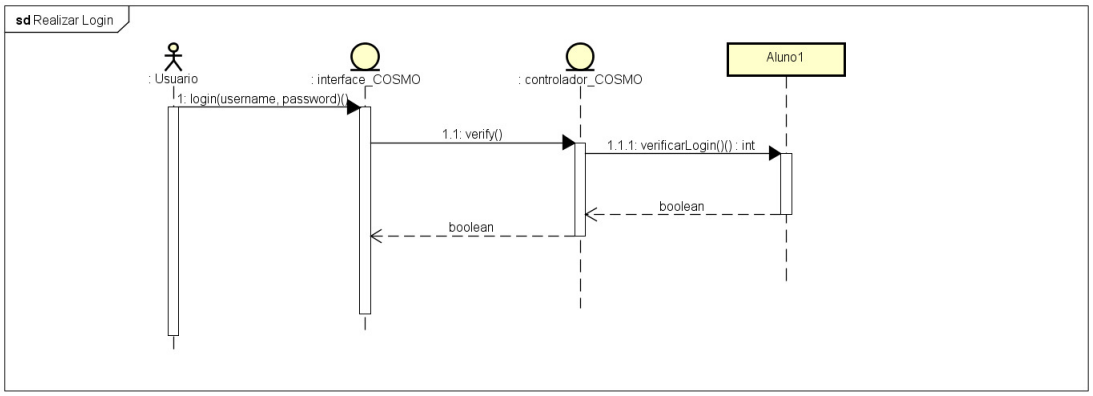
dos objetos envolvidos (GUEDES, 2009). A Figura 18 apresenta um exemplo simples de mensagem.

Figura 18 – Exemplo de Mensagem



No total foram produzidos um diagrama de sequência para cada caso de uso. Os diagramas a seguir têm como objetivo apresentar as ideias mais importantes da modelagem de sequência do COSMO.

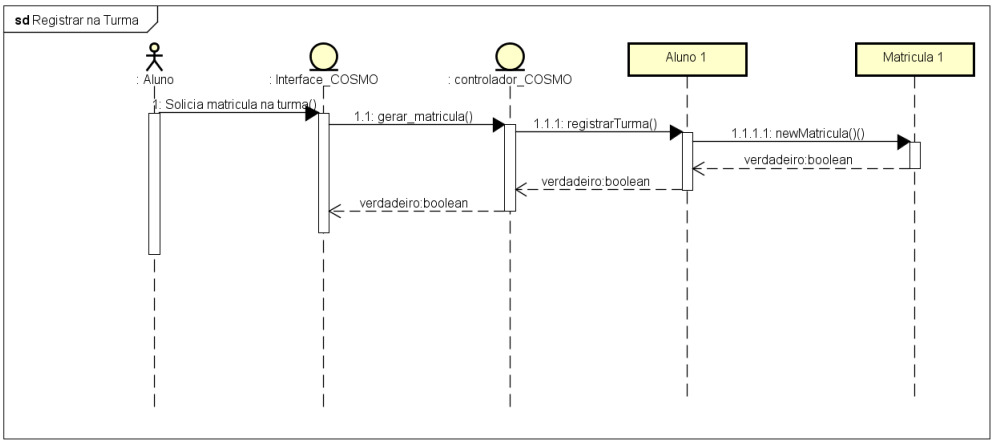
Figura 19 – Diagrama de Sequência - Realizar login



A Figura 19 apresenta o diagrama de sequencia para a ação Realizar Login. Inicialmente, o usuário do sistema realiza a ação de fazer login, fornecendo um usuário e senha. Isso então acarreta o disparo do método verificarLogin() que retornará verdadeiro ou falso caso os dados estejam de acordo.

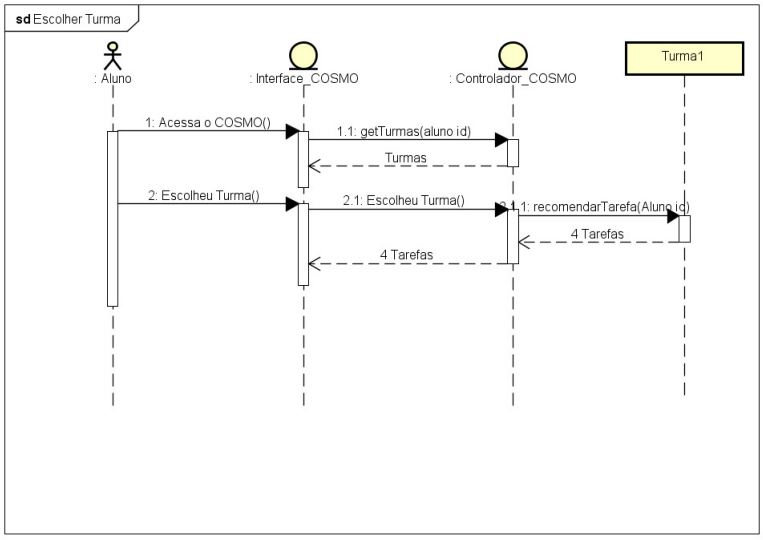
A Figura 20 apresenta o diagrama de sequência para a ação Registrar na Turma. Inicialmente, o aluno solicita a matrícula em uma turma, o que dispara o método regis-

Figura 20 – Diagrama de Sequência - Registrar na Turma



trarTurma(). Este método, por sua vez, passa uma mensagem para um objeto do tipo Turma, disparando o método newMatricula(). Isso então irá instanciar um objeto do tipo Matricula e retornará uma mensagem de verdadeiro caso tenha êxito.

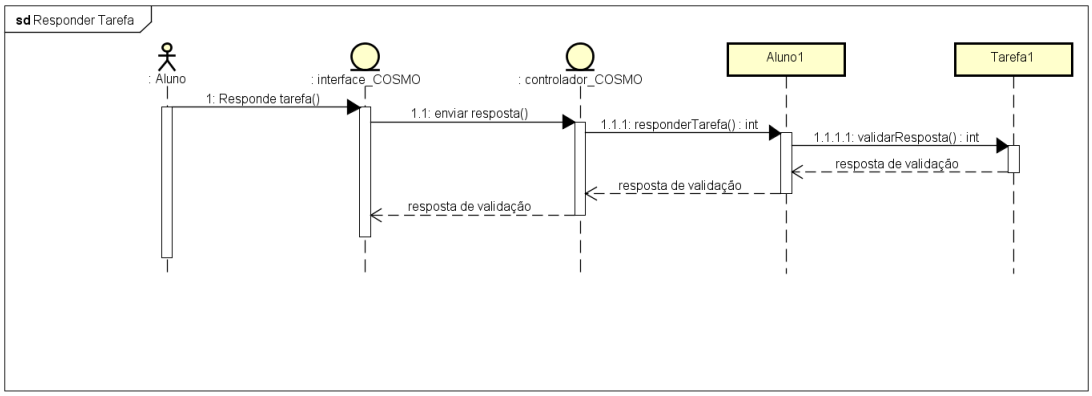
Figura 21 – Diagrama de Sequência - Escolher Turma



A Figura 21 apresenta o diagrama de sequência para a ação Escolher Turma, Inicialmente o aluno acessará o COSMO, o que acarreta o disparo do método getTurmas(), recebendo de volta uma lista de turmas disponíveis. Logo em seguida o aluno escolhe uma turma, o que acarreta o disparo do método recomendarTarefa(), que retorna uma lista com quatro tarefas para o aluno.

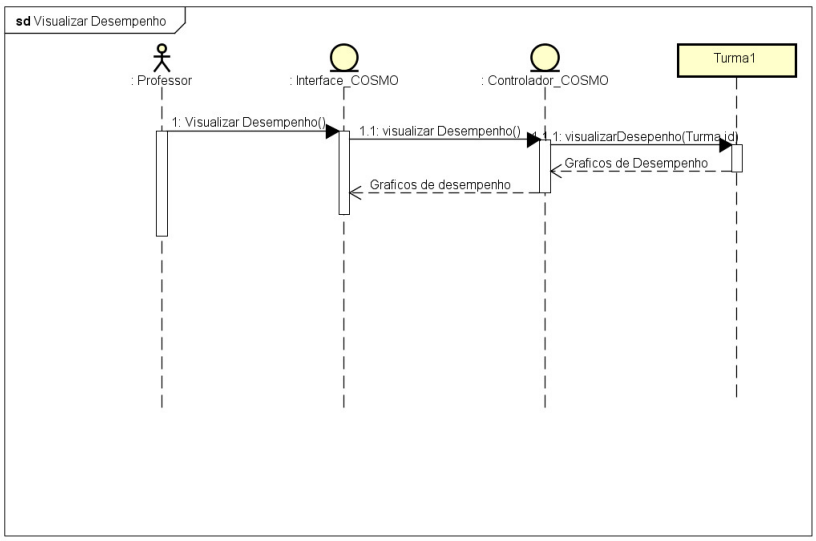
A Figura 22 apresenta o diagrama de sequência para a ação Escolher Turma. Inicialmente o aluno responde uma tarefa, o que inicia o disparo do método responder-

Figura 22 – Diagrama de Sequência - Responder Tarefa



Tarefa(). Este método por sua vez passa uma mensagem para um objeto do tipo Tarefa para validar a resposta apresentada pelo aluno, chamando o método validarResposta(). O que então retornará verdadeiro caso a resposta esteja correta, fazendo com que o método responderTarefa() também retorne verdadeiro e instancie um objeto da classe Resposta.

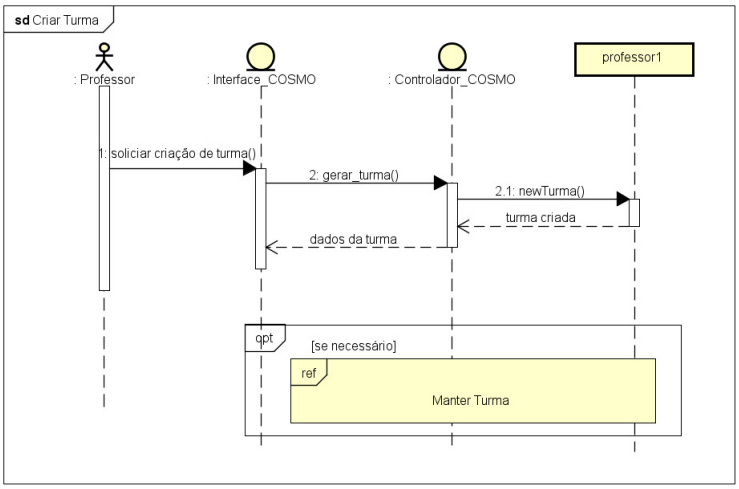
Figura 23 – Diagrama de Sequência - Visualizar Desempenho



A Figura 23 apresenta o diagrama de sequência para a ação Visualizar Desempenho. Inicialmente, o professor clica na opção visualizar desempenho da turma, o que dispara o método visualizarDesempenho(). Este método, por sua vez, retorna os dados da turma relacionados ao desempenho da turma.

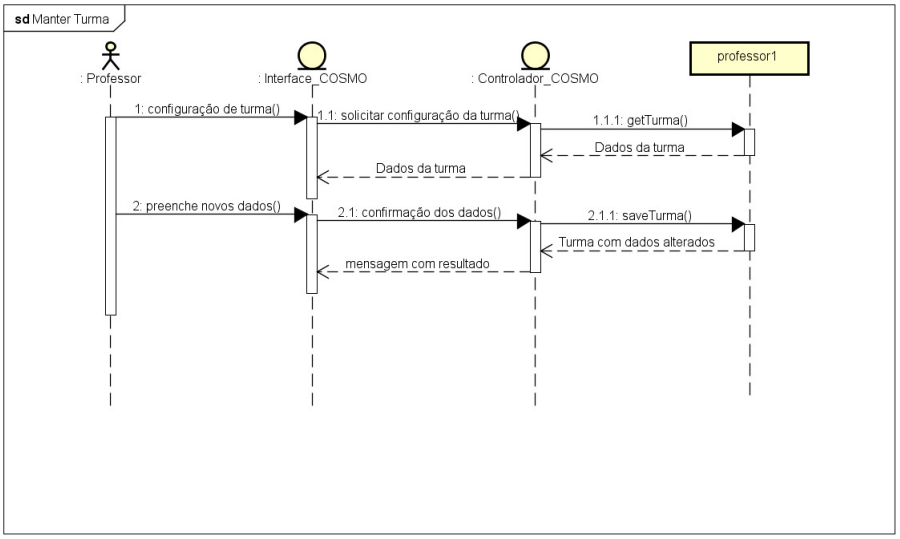
A Figura 24 apresenta o diagrama de sequencia para a ação Criar Turma, primeiramente o professor deseja criar uma turma nova, o que acarretará o disparo do método newTurma() em um objeto da classe Professor, passando como parâmetro os dados neces-

Figura 24 – Diagrama de Sequência - Criar Turma



sários para a criação de uma turma (nome, etc), o que retornará uma mensagem caso a turma seja criada e os dados da turma criada caso o método tenha êxito, e uma mensagem de erro caso contrário. Em seguida, conforme demonstra o fragmento combinado do tipo opt, a turma poderá ser atualizada, caso necessário.

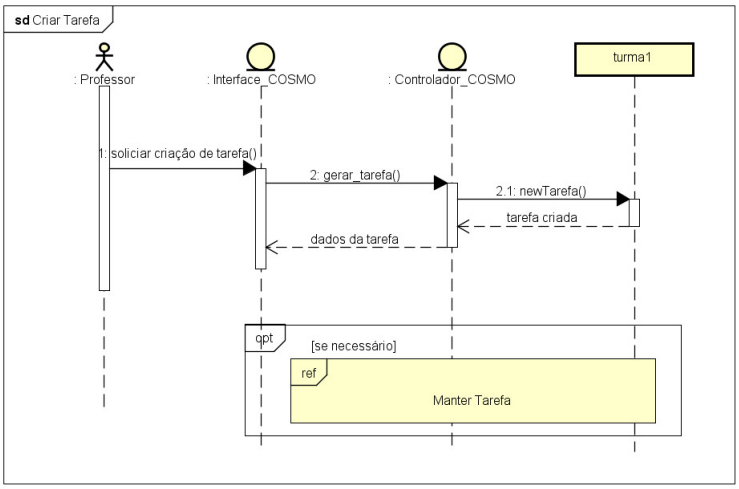
Figura 25 – Diagrama de Sequência - Manter Turma



A Figura 25 apresenta o diagrama de sequência para a ação Manter Turma. O professor clica no botão de configurações de turma, o que dispara o método getTurmas(), retornando assim as informações da turma. Logo em seguida, ele preenche os novos dados, o que dispara o método saveTurma(), retornando uma mensagem com o resultado da ação.

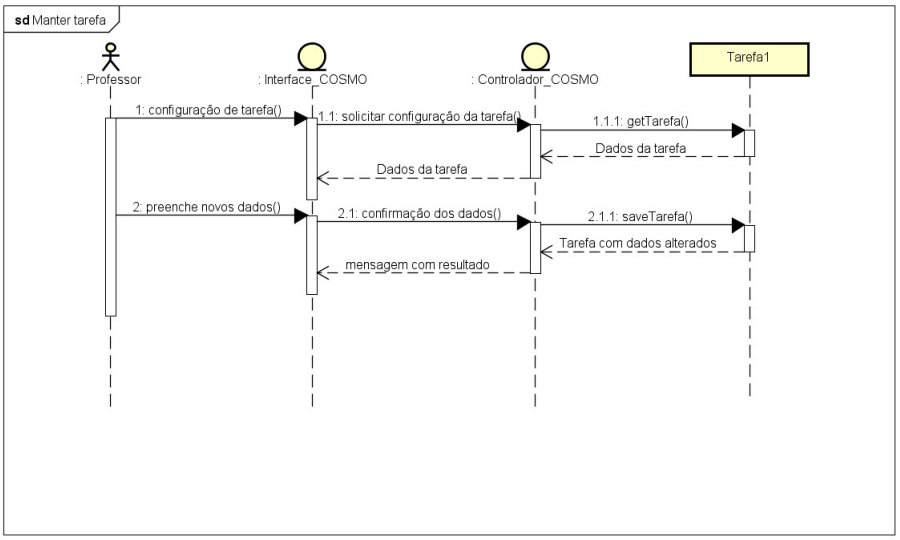
A Figura 26 apresenta o diagrama de sequência para a ação Criar Tarefa. Primei-

Figura 26 – Diagrama de Sequência - Criar Tarefa



ramente o professor deseja criar uma tarefa nova, o que acarretará o disparo do método newTarefa() em um objeto da classe Turma, passando como parâmetro os dados necessários para a criação de uma Tarefa (nome, etc). Isso retornará uma mensagem caso a Tarefa seja criada e os dados da turma criada caso o método tenha êxito, e uma mensagem de erro, em caso contrário. Em seguida, conforme demonstra o fragmento combinado do tipo opt, a Tarefa poderá ser atualizada, caso necessário.

Figura 27 – Diagrama de Sequência - Manter Tarefa



A Figura 27 apresenta o diagrama de sequência para a ação Manter Tarefa. O professor clica no botão de configurações de tarefa, o que dispara o método getTarefa(), retornando assim as informações da tarefa. Logo em seguida, ele preenche os novos dados,

o que dispara o método saveTarefa(), retornando uma mensagem com o resultado da ação.

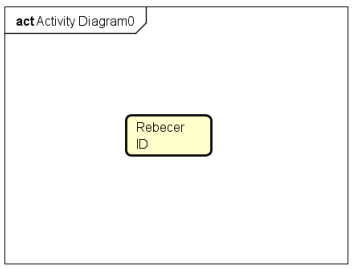
4.4 Diagrama de Atividade

A modelagem de atividade enfatiza a sequência de ações e condições para coordenar comportamento de baixo nível (GUEDES, 2009). Dessa forma, o diagrama de atividade é o diagrama com maior ênfase ao nível de algoritmo e provável um dos mais detalhistas.

O diagrama de atividade tem como objetivo mostrar as atividades que compõem um processo de sistema e o fluxo de controle de uma atividade para a outra (SOMMERVILLE et al., 2003). Atividades podem descrever computação procedural. Neste contexto, elas são métodos correspondentes às operações sobre classes. Uma atividade é composta por um conjunto de ações, ou seja, os passos necessários para que a atividade seja concluída (GUEDES, 2009).

O diagrama é composto principalmente por nós de ação e fluxos de controle. Um nó de ação é o elemento mais básico de uma atividade. Ele representa uma tarefa executada por um sistema de software (PRESSMAN, 1995). Um nó de ação é atômico, não podendo ser decomposto. A Figura 28 apresenta um exemplo de nó de ação.

Figura 28 – Exemplo de Nó de Ação



Já o fluxo de controle é um conector que liga dois nós, enviando sinais de controle a eles. Ele é representado por uma linha contendo uma seta apontando para o novo nó e partindo do antigo.

Na Figura 29 há dois nós de ação. A primeira ação representa o recebimento de um ID enquanto a segunda representa a consulta do ID. A passagem de uma ação para outra é representada pela linha de fluxo de controle que une as duas ações. A seta de fluxo de controle demonstra a ordem em que as ações serão executadas.

Os diagramas apresentados a seguir têm por objetivo apresentar uma visão das sequências de ações e condições necessários para executar alguns dos métodos mais importante do ambiente virtual de aprendizagem COSMO.

Figura 29 – Exemplo de Fluxo de Controle

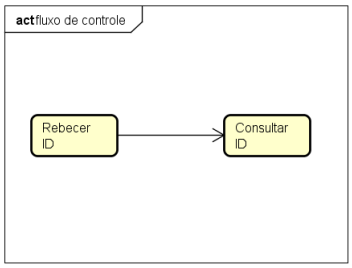
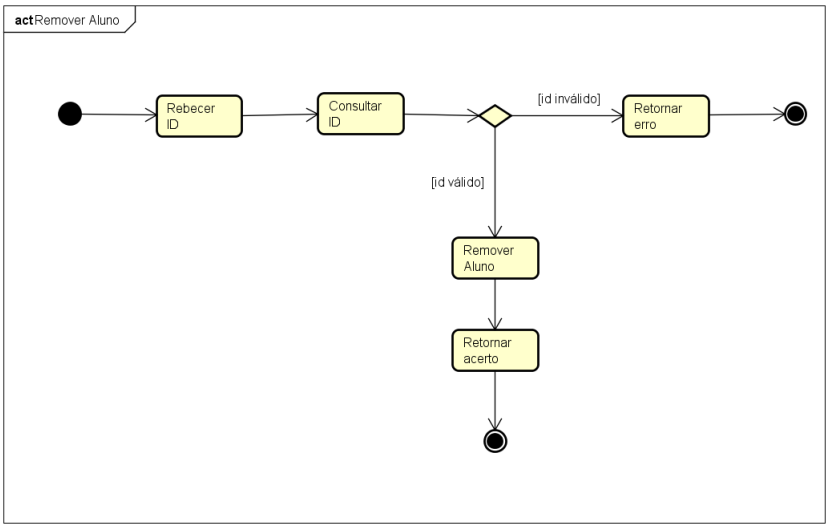


Figura 30 – Diagrama de Atividade - Remover Aluno



Na Figura 30, a primeira ação desse método é receber o ID do aluno que deve ser removido da turma. Depois de receber o id do aluno, passa-se a ação de consultá-lo na lista de alunos registrados na turma. Em seguida, há um nó de decisão no qual é verificado se este aluno está registrado na turma. Caso ele não esteja registrado é retornado uma mensagem de erro e o método é encerrado. Caso ele esteja registrado, o aluno é então removido e é retornada uma mensagem de acerto.

A Figura 31 apresenta o diagrama de atividades para o método visualizar desempenho. A primeira ação deste método é receber a resposta apresentada pelo aluno para a tarefa. Depois de receber a resposta, é então montado um conjunto de entradas e saídas com as respostas apresentadas. Em seguida, são comparados os resultados gerados pela resposta apresentada com os resultados esperados pela tarefa. Caso tais resultados sejam idênticos é retornada uma mensagem de acerto, caso contrário é retornada uma mensagem de erro.

A Figura 32 apresenta o diagrama de atividades para o método visualizar desempe-

Figura 31 – Diagrama de Atividade - Programming: Validar Resposta

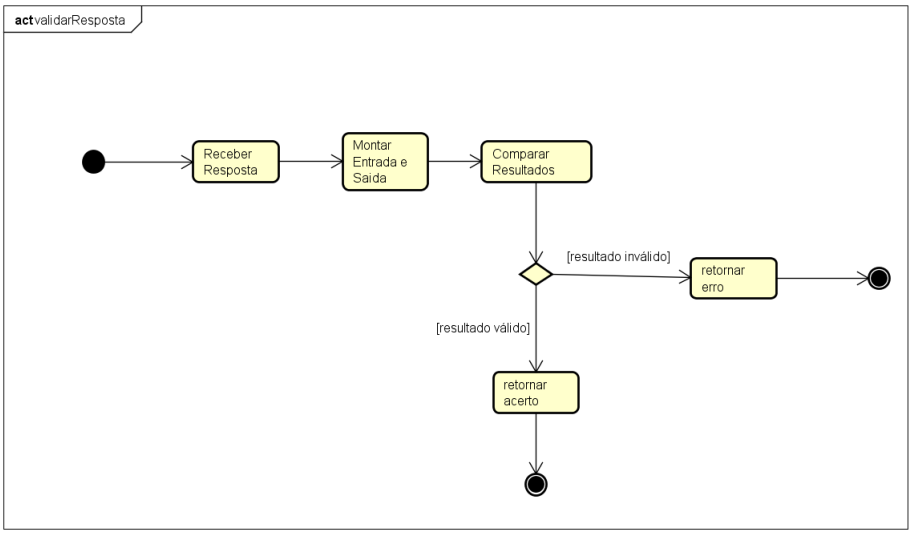
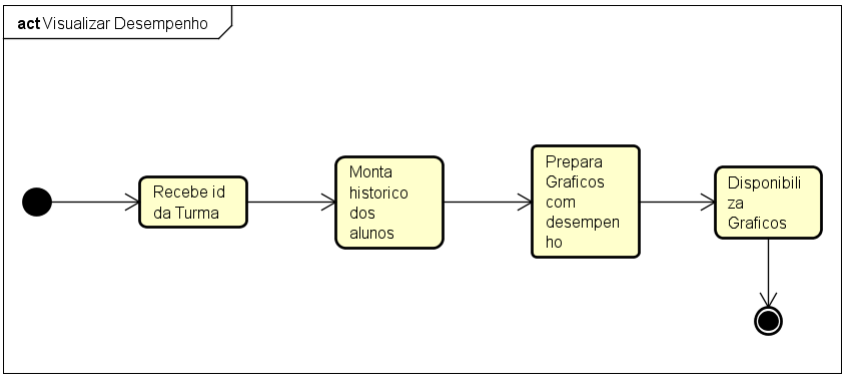


Figura 32 – Diagrama de Atividade - Visualizar Desempenho



nho. A primeira ação é receber o ID da turma, logo em seguida é montado um histórico dos alunos da turma. Este histórico é então usado para preparar gráficos com o desempenho da turma.

A Figura 33 apresenta o diagrama de atividades para o método responder tarefa. A primeira ação deste método é receber a resposta apresentada pelo aluno para a solução da tarefa. Em seguida é recebido a tarefa em questão para ser respondida,e então é executado o método de validação da tarefa, cada tipo de tarefa possui um método próprio de validação, a Figura 31 apresenta o método de validação da subclasse Programming. Caso a resposta de retorno tenha sido de erro, o método retorna uma mensagem de erro e encerra, caso a mensagem de retorno do método de validação tenha sido aceito, a resposta apresentada pelo aluno é então armazenada e o método retorna aceito.

Figura 33 – Diagrama de Atividade - Responder Tarefa

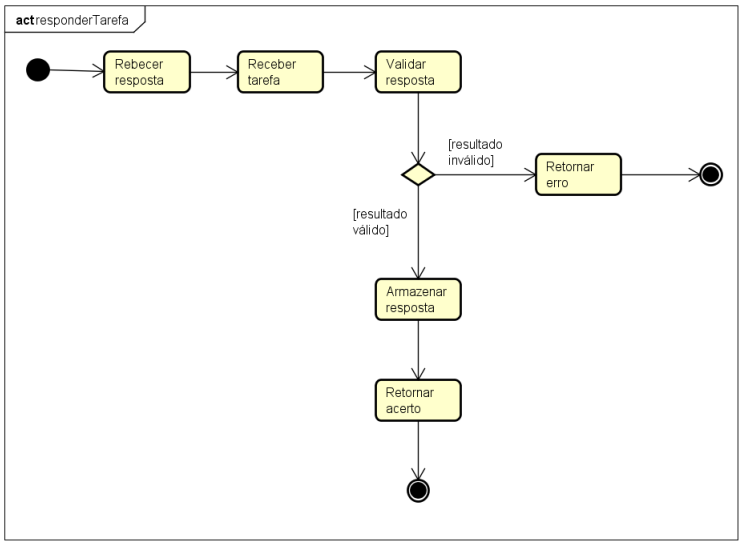
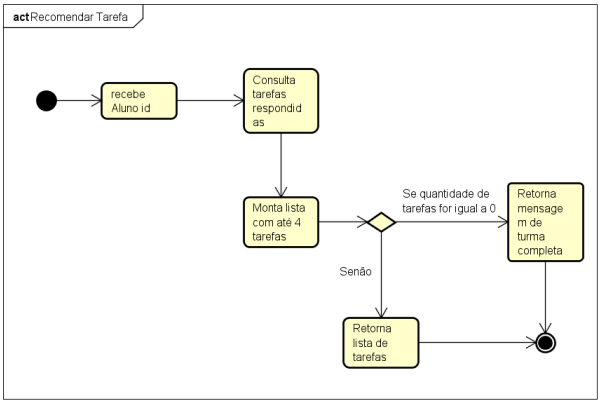


Figura 34 – Diagrama de Atividade - Recomendar Tarefa



A Figura 34 apresenta o diagrama de atividade para o método recomendar tarefa, a primeira ação deste diagrama é receber o id do aluno, em seguida este id é utilizado para consultar as tarefas respondidas pelo aluno anteriormente. Com esse histórico é montada uma lista de tarefas com até 4 tarefas que serão recomendadas para este aluno, se a quantidade de tarefas nesta lista for 0, é retornada uma mensagem indicando que a turma foi completa, caso contrário é retornado a lista com as tarefas.

5 Resultados

5.1 Comparativo de requisitos

Esta seção apresenta um comparativo dos requisitos coletados no Capítulo 2 entre as tecnologias relacionadas apresentadas no Capítulo 3 e o ambiente virtual de aprendizagem COSMO.

Tabela 41 – Comparativo geral entre o COSMO e as tecnologias relacionadas

	COSMO	**UVa**	**URI**	**BOCA**	**Codecademy**
RF01	x	x	x		x
RF02	x				
RF03	x				
RF04	x		x	x	
RF05	x	x	x	x	x
RF06	x		x		
RF07	x	x	x		x
RF08	x	x		x	x
RF09	x				
RF10	x	x	x		x

Como visto na Tabela 41, apesar de cumprir grande parte dos requisitos, as tecnologias relacionadas não atendem às necessidades dos *stakeholders*.

5.1.1 Uva Online Judge

O *UVa Online Judge* atende o requisito funcional de validar respostas da tarefa. Nele é possível responder um exercício dentro do próprio sistema, escolher a linguagem para responder tal problema e ter o resultado da sua resposta imediatamente após enviar o exercício.

Alem disso é possível também visualizar o desempenho do aluno ao responder a tarefa através do tempo de execução da solução apresentada pelo aluno, mesmo que seja bem simplificado.

O Aluno também é capaz de escolher os exercícios que quer responder dentre um robusto acervo de problemas.

Entretanto, dentro do *UVa Online Judge* não existe uma interface voltada para o professor, logo o conceito de turma não existe. Isso impossibilita a capacidade do professor de acompanhar o desempenho da turma e criar novos exercícios de acordo com sua necessidade.

O *UVa Online Judge* não possui nenhum sistema de recomendação de exercícios, o que faz com que o aluno se sinta intimidado e sem saber por onde começar. Alem disso, não existe um senso de progressão por parte do sistema que indique que o aluno esteja melhorando.

5.1.2 URI Online Judge

O *URI Online Judge* é a plataforma que atende mais requisitos dentre as apresentadas. Nele é possível validar a resposta da tarefa dentro do próprio sistema de forma autônoma, possibilitando também escolher a linguagem escolhida pelo aluno.

Além disso o *URI Online Judge* possui um conceito de turma implementado nele. É possível criar uma turma, adicionar membros e execícios.

O sistema também possui uma interface para o professor visualizar o desempenho dos alunos.

Entretanto o sistema não é capaz de recomendar tarefas para o aluno. As tarefas apresentadas pelo sistema para os alunos são tarefas escolhidas pelo professor e adicionadas a lista de tarefas da turma.

Além disso, o sistema não possibilita ao professor criar novas tarefas, apenas adicionar tarefas já existentes à lista de tarefas disponíveis e possui apenas um tipo de tarefa, a do tipo programável.

5.1.3 BOCA

Apesar do boca apresentar um conceito parecido com o conceito de turma na forma de grupos e ser possível criar tarefas dentro do sistema, ele não cumpre grande parte dos requisitos necessários.

O BOCA não valida a resposta apresentada pelo aluno automaticamente, é necessário que uma pessoa corrija manualmente.

Além disso, não há uma forma de recomendação de quais tarefas devem ser feitas, também não é possível buscar uma turma, todo o processo de matricula é feito por terceiros e só existe um tipo de tarefa, a programável.

5.1.4 Codecademy

O *Codecademy* cumpre grande parte dos requisitos apresentados, ele pode validar a resposta apresentada imediatamente, é possível visualizar o desempenho em forma de *badges* e é possível escolher tarefas e o tópico a ser resolvido.

Entretanto o *Codecademy* não possui conceito de turma e nem possui uma interface para o professor, o que invalida grande parte dos requisitos necessários, como matricula de turma, manter turma e buscar turma.

Além disso, o *Codecademy* é linear e não recomenda tarefas para o usuário e possui apenas um tipo de tarefa, a programável.

5.1.5 COSMO

O COSMO cumpre todos os requisitos descobertos durante a fase de levantamento de requisitos. Nele a resposta apresentada pelo aluno é validada instantaneamente pelo sistema, retornando o resultado da tarefa (RF01).

Além disso, o sistema recomenda para os alunos 4 tarefas a serem feitas, levando em consideração seu histórico(RF02). Cada tarefa pode ser pulada e caso o aluno não a queira resolver, o sistema imediatamente recomenda outra tarefa para o aluno.

O aluno pode buscar a turma que ele quiser das turmas disponíveis, contando que ela não seja uma turma já indisponível(RF03).

Para o professor é possível criar turmas(RF04) e tarefas(RF05) à vontade e manipulá-las, adicionando ou removendo conteúdo.

Para acessar o o conteúdo da turma, o aluno precisa se matricular na turma, e caso a turma seja privada o professor precisa aceitar tal matricula(RF06).

O sistema possibilita ao professor visualizar o desempenho geral da turma, baseando-se nos alunos matriculados(RF07).

O aluno também é capaz de escolher qual tarefa responder entre as quatro sugeridas pelo sistema, além disso ele é capaz de responder qualquer tarefa já respondida anteriormente a qualquer momento(RF08).

O COSMO também possibilita que o professor escolha o tipo de tarefa a ser criada, sendo feito de uma maneira extensiva para que mais tipos de tarefas sejam adicionadas no futuro(RF09).

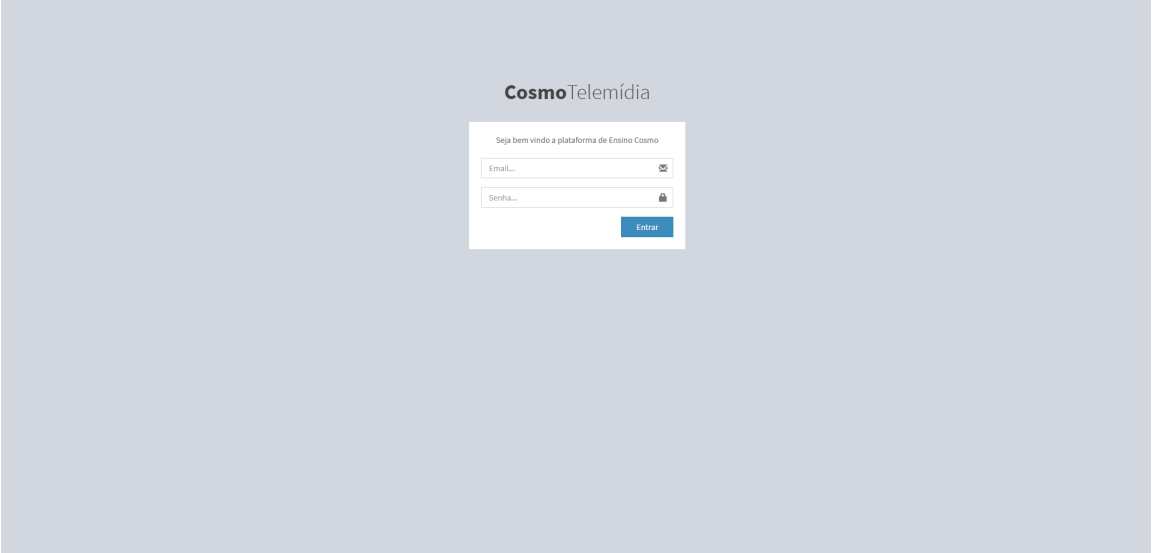
Além disso, o aluno também é capaz de apresentar uma resposta para uma tarefa no sistema, escolhendo o tipo de linguagem a ser utilizada na resposta(RF10).

5.2 Prototipação do sistema

Esta seção apresenta um protótipo do COSMO criado pela equipe de desenvolvimento utilizando a modelagem apresentada no Capítulo 4. Foi feita uma sessão de teste do protótipo na disciplina de Algoritmos 1 do departamento de informática no curso de ciência da computação na UFMA.

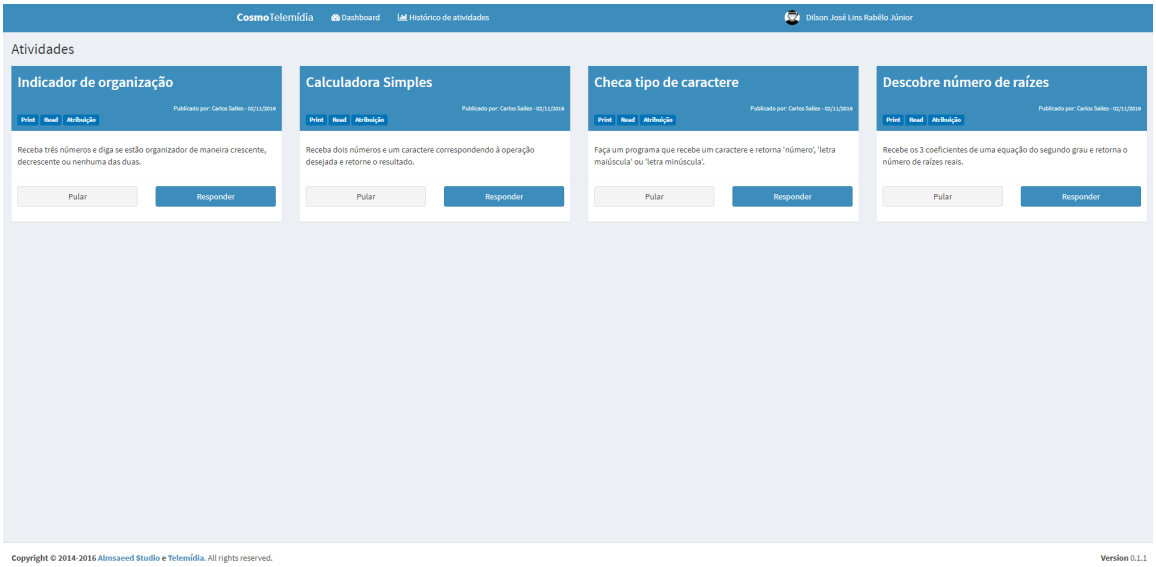
A figura 35 apresenta a tela inicial do sistema, onde o aluno deve fazer o acesso, apresentando o login e a senha. Caso o mesmo não possua cadastro, ele deve se cadastrar, apesar da funcionalidade de criar login e senha já esteja feita, ela não está disponível ainda para controlar o acesso ao protótipo.

Figura 35 – Interface Inicial do COSMO



A tela após o login do sistema é a tela de recomendação de tarefa apresentada na Figura 36. Nota-se que o motivo da criação de tal protótipo foi o teste com os alunos, que serão os usuários finais do sistema. Por esse motivo não foi implementado o conceito de turma e nenhuma das interfaces do professor, apenas as funcionalidades do aluno.

Figura 36 – Interface da recomendação de tarefas do cosmo

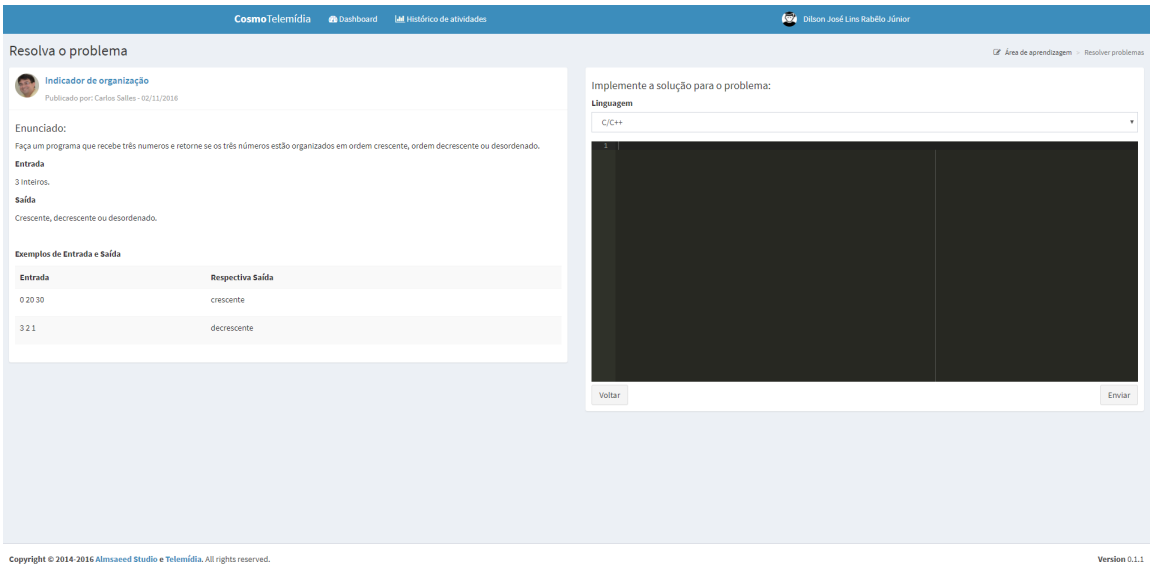


Na tela de recomendação de tarefas, o aluno consegue ver um resumo da tarefa, seguido do conteúdo que a tarefa cobre e a opção de pular caso o mesmo não queira responder tal tarefa.

A Figura 37 apresenta a tela de resolução da tarefa do tipo programável. Nela o aluno deve implementar a solução para o problema ao lado direito da tela. O Aluno também pode selecionar o tipo de linguagem que ele quer utilizar para implementar a solução.

Cada tarefa possui um enunciado, onde apresenta o problema, além de uma possível entrada e saída. Além disso também é apresentado possíveis exemplos de entrada e saídas para ficar mais claro para o aluno.

Figura 37 – Interface da tarefa



6 Conclusão

O objetivo principal deste trabalho foi alcançado quanto ao levantamento de requisitos e modelagem do ambiente virtual de aprendizagem COSMO. Os requisitos funcionais e não funcionais coletados durante a etapa de entrevista e *brainstorming* serviram como base para toda a modelagem com eficacia. Os diagramas apresentados no formato UML ficaram claros e simples, com um foco em apresentar as funcionalidades necessárias para cumprir os requisitos coletados anteriormente. Alem disso foi mostrado que apesar de existirem diversas tecnologias similares ao COSMO, elas não atendem os requisitos necessários pelos *stakeholders* do sistema.

Apesar dos resultados promissores, muito ainda tem que ser feito. Inicialmente deve-se dar continuidade ao protótipo apresentado do COSMO. Com isso, como trabalhos a serem executados futuramente, pretende-se implementar as funcionalidades apresentadas na modelagem do sistema. Alem disso, é possível que novas funcionalidades venham a ser pensadas e adicionadas aos requisitos e modelagem , como possíveis tipos de tarefas, novas maneiras de recomendações de tarefas e uma possível expansão para atividades de áreas alem da computação.

O levantamento de requisitos e a modelagem são etapas de extrema importância para o desenvolvimento de softwares. Assim, é sempre necessário que toda e qualquer mudança e incremento seja adicionada aos requisitos e modelagem apresentadas.

Referências

ALLISON, I. K.; ORTON, P.; POWELL, H. A virtual learning environment for introductory programming. In: HIGHER EDUCATION ACADEMY SUBJECT CENTRE FOR INFORMATION AND COMPUTER SCIENCES (HEA-ICS). *Proceedings of the 3rd Conference of the LTSN-ICS*. [S.l.], 2002. Citado na página 14.

CAMPOS, C. P. D.; FERREIRA, C. E. Boca: um sistema de apoio a competições de programação. In: *Workshop de Educação em Computação*. [S.l.: s.n.], 2004. p. 885–895. Citado 3 vezes nas páginas 29, 30 e 31.

CYBIS, W. de A.; BETIOL, A. H.; FAUST, R. *Ergonomia e Usabilidade 3ª edição: Conhecimentos, Métodos e Aplicações*. [S.l.]: Novatec Editora, 2015. Citado na página 20.

FILHO, W. de P. P. *Engenharia de software*. [S.l.]: LTC, 2003. v. 2. Citado na página 17.

FRANÇA, A. B. et al. Um sistema orientado a serviços para suporte a atividades de laboratório em disciplinas de técnicas de programação com integração ao ambiente moodle. *RENOTE*, v. 9, n. 1, 2011. Citado na página 29.

GUEDES, G. T. *UML 2: uma abordagem prática*. [S.l.]: Novatec Editora, 2009. Citado 5 vezes nas páginas 33, 40, 48, 49 e 54.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. The objectory software development process. *ISBN: 0-201-57169-2, Addison Wesley*, 1997. Citado na página 19.

KURNIA, A.; LIM, A.; CHEANG, B. Online judge. *Computers & Education*, Elsevier, v. 36, n. 4, p. 299–315, 2001. Citado na página 26.

PALMEIRA, L. B.; SANTOS, M. P. Evasão no bacharelado em ciência da computação da universidade de brasília: análise e mineração de dados. 2015. Citado na página 14.

PRESSMAN, R. S. *Engenharia de software*. [S.l.]: Makron books Sao Paulo, 1995. v. 6. Citado 5 vezes nas páginas 33, 40, 41, 48 e 54.

RAJKUMAR, G.; ALAGARSAMY, D. K. the most common factors for the failure of software development project. 2013. Citado 2 vezes nas páginas 14 e 15.

REVILLA, M. A.; MANZOOR, S.; LIU, R. Competitive learning in informatics: The uva online judge experience. *Olympiads in Informatics*, v. 2, p. 131–148, 2008. Citado na página 26.

SANTOS, R. P. dos; COSTA, H. A. X. Análise de metodologias e ambientes de ensino para algoritmos, estruturas de dados e programação aos iniciantes em computação e informática. *INFOCOMP*, v. 5, n. 1, p. 41–50, 2006. Citado na página 14.

SELIVON, M. et al. Uri online judge academic: Integraç ao e consolidaç ao da ferramenta no processo de ensino/aprendizagem. 2015. Citado 3 vezes nas páginas 27, 28 e 29.

SOMMERVILLE, I. et al. *Engenharia de software*. [S.l.]: Addison Wesley São Paulo, 2003. v. 6. Citado 8 vezes nas páginas 15, 17, 18, 19, 20, 33, 34 e 54.