UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO

BRUNO DE PINHO SCHETTINO

SAPOS 2: GESTÃO DE DISCIPLINAS, ETAPAS E CREDENCIAMENTOS NO SISTEMA DE APOIO À PÓS-GRADUAÇÃO

Niterói
2013

BRUNO DE PINHO SCHETTINO

SAPOS 2: GESTÃO DE DISCIPLINAS, ETAPAS E CREDENCIAMENTOS NO SISTEMA DE APOIO À PÓS-GRADUAÇÃO

Trabalho de conclusão de curso apresentado ao Departamento de Ciência da Computação da Universidade Federal Fluminense para obtenção do Grau de Bacharel em Ciência da Computação.

Orientadores: Prof. Dr. LEONARDO GRESTA PAULINO MURTA
Prof. Dra. VANESSA BRAGANHOLO MURTA

Niterói
2013

BRUNO DE PINHO SCHETTINO

SAPOS 2: GESTÃO DE DISCIPLINAS, ETAPAS E CREDENCIAMENTOS NO SISTEMA DE APOIO À PÓS-GRADUAÇÃO

Trabalho de conclusão de curso apresentado ao Departamento de Ciência da Computação da Universidade Federal Fluminense para obtenção do Grau de Bacharel em Ciência da Computação.

Aprovada em Agosto de 2013.

BANCA EXAMINADORA

_____

Prof. Dr. LEONARDO GRESTA PAULINO MURTA – Orientador
UFF

_____

Prof. Dra. VANESSA BRAGANHOLO MURTA – Orientador
UFF

_____

Prof. Dr. DANIEL CARDOSO MORAES DE OLIVEIRA
UFF

_____

Prof. Dr. CÉLIO VINICIUS NEVES DE ALBUQUERQUE
UFF

Niterói
2013

Dedico este trabalho a Coordenação da Pós-Graduação em Ciência da Computação da Universidade Federal Fluminense e aos meus orientadores Leonardo Murta e Vanessa Braganholo.

AGRADECIMENTOS

Ao meu pai Sergio e minha falecida mãe Beatriz.

À minha irmã Adrianne.

À minha avó Ilda e meu tio Roberto.

Aos meus falecidos avós Albertino e “Zé Pinheiro” e Sueli.

À minha amiga, companheira e noiva Giselle.

Aos “Manolos” de Nova Iguaçu: Álvaro, Bernardo, “Dudu”, Gabriel, João Filipe e Neves.

Aos velhos amigos: André, Filipe, Luiz Paulo, “Jerê”, José Renato, Rodrigo, Ygor e ao falecido amigo Lucas.

A todos os amigos da UFF, STI e Quantum. Um agradecimento especial a Abraão Caldas, Abraão Miranda, Jorge Thiago, Luan, “Max”, Thiago Nazareth, Raissa, Rodrigo Castro e Rennan.

A Tiago Amaro e Rodrigo Ferreira por terem ficado dispostos a tirar dúvidas sobre a primeira versão do SAPOS e ajudar com o desenvolvimento desse trabalho.

A Everton Moreth por ter participado do desenvolvimento da segunda versão do SAPOS.

A todos os professores que estiveram presentes durante a graduação na UFF, no colégio EME e no colégio Iguaçuano.

Aos orientadores Leonardo Murta e Vanessa Braganholo, agradeço pela dedicação, paciência e por todo o conhecimento passado durante o desenvolvimento desse trabalho.

RESUMO

As coordenações dos cursos de Pós-Graduação da Universidade Federal Fluminense (UFF) são responsáveis pelo controle de uma grande variedade de dados. Com o objetivo de diminuir o trabalho manual realizado pelas coordenações, a Superintendência de Tecnologia da Informação da UFF (STI) está desenvolvendo o Sistema de Gestão da Pós-Graduação (SisPós), que será responsável pela automatização dos processos executados. Porém, por ser um sistema que deverá ser adotado por todas as coordenações, o SisPós tem em seu planejamento desenvolver funcionalidades comuns a todas as coordenações, antes de atender as demandas mais específicas de cada coordenação.

Para suprir as necessidades do Instituto de Computação da Universidade Federal Fluminense (IC-UFF), foi desenvolvido o Sistema de Apoio à Pós-Graduação (SAPOS). A primeira versão do sistema contempla o cadastro de informações de alunos, matrículas, professores, orientações, bolsas e um suporte inicial ao controle de etapas. Para o desenvolvimento do SAPOS, foi adotada uma metodologia de desenvolvimento ágil, que tem como característica a constante comunicação com o usuário, visando a sua maior satisfação. Em alinhamento com essa metodologia, foi adotado o *framework Ruby on Rails* para o desenvolvimento do sistema, com o objetivo de acolher mudanças com facilidade durante a implementação do mesmo.

Este trabalho trata-se da segunda versão do SAPOS, que consiste na implementação de novas funcionalidades como gestão de disciplinas e credenciamentos, aprimoramentos no controle de etapas, buscas avançadas, criação de testes automatizados, entre outras. Essa versão do sistema está em uso no momento, contando com 385 matrículas de alunos atuais e passados na sua base de dados, 75 credenciamentos de professores nos níveis de mestrado ou doutorado, 248 etapas registradas, 164 prorrogações concedidas e 92 disciplinas cadastradas, com 651 inscrições feitas sobre 121 turmas.

Palavras-chaves: Gestão Acadêmica de Pós-Graduação, Engenharia de Software, Aplicação *Web*, Ruby on Rails.

ABSTRACT

The Graduate Programs at Fluminense Federal University are responsible for managing a variety of data. Aiming at reducing the manual work done by the Graduate Programs, the Information Technology division at UFF (STI) is developing a system called SisPós, which will be responsible for the automation of such processes. However, SisPós is now focusing on features that are common to all Graduate Programs before approaching specific demands of each Graduate Program.

SAPOS was developed to supply the specific needs of the Graduate Program on Computing at Fluminense Federal University. The first version of the system manages students, enrollments, professors, advisements, scholarships, and provides an initial support for phases. SAPOS was developed following an agile methodology, which is characterized by constant communication with the user, aiming at his greatest satisfaction. The Ruby on Rails framework was chosen, in alignment with an agile methodology, in order to easily embrace changes during development.

This work it is the second version of SAPOS, which implements new features such as managing disciplines and advisement authorizations, improvements on phases, advanced searches, the creation of automated tests, among others. This version of SAPOS, which is currently in production, stores 385 student enrollments, 75 advisement authorizations, 248 accomplished phases, 164 phase deferrals, and 92 courses with 651 class enrollments on 121 classes.

Keywords: Post-graduation Academic Management, Software Engineering, Web Applications, Ruby on Rails.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 9
LISTA DE ILUSTRAÇÕES 10
Capítulo 1 – INTRODUÇÃO 11
Capítulo 2 – SISTEMAS DE APOIO À PÓS-GRADUAÇÃO 13
2.1 SISPÓS 13
2.2 PRIMEIRA VERSÃO DO SAPOS 15
2.3 DISCUSSÃO 18
Capítulo 3 – SAPOS 2 19
3.1 NOVO CONTROLE DE ETAPAS E PRORROGAÇÕES 19
3.2 NOVA BUSCA AVANÇADA 21
3.3 CONTROLE DE CREDENCIAMENTOS 23
3.4 CONTROLE DE DISCIPLINAS 25
3.5 NOVOS RELATÓRIOS 27
3.6 DISCUSSÃO 29
Capítulo 4 – DESENVOLVIMENTO 30
4.1 METODOLOGIA 30
4.2 PADRÃO MVC 32
4.3 IMPLEMENTAÇÃO 34
4.3.1 RUBY 34
4.3.2 RUBY ON RAILS 34
4.3.3 BIBLIOTECAS AUXILIARES 38
4.4 FERRAMENTAS 40
4.4.1 MODELAGEM 40
4.4.2 VERSIONAMENTO 41
4.4.3 CONTROLE DE MODIFICAÇÕES 43
4.5 TESTES AUTOMATIZADOS 44
4.6 DISCUSSÃO 46
Capítulo 5 – CONCLUSÃO 47
REFERÊNCIAS BIBLIOGRÁFICAS 49
APÊNDICE A – MAPEAMENTO DAS CLASSES DO SISTEMA GERADO PELO RAILS ERD 51

LISTA DE ABREVIATURAS E SIGLAS

CAPES: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

CoC: *Convention Over Configuration*

CRUD: *Create, Read, Update and Destroy*

DRY: *Don't Repeat Yourself*

HTTP: *Hypertext Transfer Protocol*

IC-UFF: Instituto de Computação da Universidade Federal Fluminense

MVC: Modelo - Visão – Controle

ORM: *Object-Relational Mapping*

PROPPI: Pró-Reitoria de Pesquisa e Pós-Graduação.

SAPOS: Sistema de Apoio à Pós-Graduação.

SisPós: Sistema de Gestão da Pós-Graduação.

SQL: *Structured Query Language*

STI: Superintendência de Tecnologia da Informação.

UFF: Universidade Federal Fluminense.

UML: *Unified Modeling Language*

LISTA DE ILUSTRAÇÕES

Figura 2.1: SisPós 14
Figura 2.2: Busca Simples 16
Figura 2.3: Busca Avançada 17
Figura 2.4: Relatório de orientações 17
Figura 3.1: Algoritmo para adição de semestres letivos 20
Figura 3.2: Nova criação de etapas 20
Figura 3.3: Nova Busca Avançada 22
Figura 3.4: Modelo simplificado de Matrícula 22
Figura 3.5: Modelo de credenciamentos 23
Figura 3.6: Controle de Credenciamentos 24
Figura 3.7: Lista de orientandos na primeira versão do SAPOS 24
Figura 3.8: Tabela de Orientações 25
Figura 3.9: Modelo do controle de disciplinas 25
Figura 3.10: Seção de Disciplinas 27
Figura 3.11: Pauta extraída do SAPOS 2 28
Figura 3.12: Histórico extraído do SAPOS 2 28
Figura 4.1: Processo na metodologia Cascata 31
Figura 4.2: Processo típico de metodologia ágil 32
Figura 4.3: Interação entre as camadas no padrão MVC 33
Figura 4.4: Modelo de Inscrições 35
Figura 4.5: Migração utilizada para a tabela de Inscrições. 36
Figura 4.6: SQL gerado pela migração da entidade Inscrição 37
Figura 4.7: Exemplo de consulta 37
Figura 4.8: Exemplo de SQL gerado em consulta pelo ORM 37
Figura 4.9: Exemplo de consulta ORM com SQL 38
Figura 4.10: Comando de geração de entidade pelo Active Scaffold 39
Figura 4.11 Diagrama de classes exportado pelo Astah 41
Figura 4.12: Tela principal do Github 42
Figura 4.13: Exemplo de ramificações do sistema 43
Figura 4.14: Interface do Sistema Redmine 44
Figura 4.15: Testes automatizados 44
Figura 4.16: Relatório de cobertura de código 45
Figura 4.17: Detalhamento de cobertura de código do modelo de Orientação 46

CAPÍTULO 1 – INTRODUÇÃO

O uso de tecnologias para armazenamento e organização de dados em forma digital tem se tornado muito comum no cotidiano de pessoas em todo o mundo. Muitos dos dados que antes eram preenchidos manualmente e armazenados em papel hoje são cadastrados por via de formulários eletrônicos e armazenados em bancos de dados digitais. Essa substituição vem acontecendo porque o ganho de organização e desempenho oferecidos pela adoção de um serviço digital compensa o custo para implantá-lo e mantê-lo.

Também existem casos nos quais dados já são armazenados em forma digital em bases de dados dispersas, ou seja, dados sobre o mesmo domínio guardados em dois ou mais lugares diferentes. A vantagem de não ter dados centralizados em uma única base de dados é que se uma base ficar indisponível, outra base pode ser acessada e assim resgatar os dados necessários. Porém, quando cada uma dessas bases possui dados redundantes e é controlada por sistemas diferentes, o risco de haver inconsistências é muito alto. Nesse cenário, é interessante centralizar todos os dados do domínio em uma única base de dados para garantir que estejam consistentes e evitar que fiquem dispersos, diminuindo a dificuldade na sua manutenção.

No caso da coordenação da Pós-Graduação em Computação da UFF, existiam informações do mesmo domínio armazenadas no *Microsoft Excel* e *Microsoft Access*, o que fazia com que as consultas e a manutenção se tornassem mais difíceis (AMARO; FERREIRA, 2013). Outro problema gerado por essas bases dispersas era a replicação dos dados. Era preciso se preocupar em atualizar todas as bases com a mesma informação para garantir a consistência tanto nos dados guardados *Microsoft Excel* quanto nos armazenados no *Microsoft Access*. Conforme o número de informações aumentava, a situação era agravada.

Dado este problema, a coordenação requisitou o desenvolvimento de um sistema *Web*, com o objetivo de centralizar os dados antes dispersos em planilhas e bancos de dados independentes. Além disso, o sistema deveria incluir outras funcionalidades que não eram atendidas pelos mecanismos previamente usados. Em função desse contexto, surgiu o Sistema de Apoio à Pós-Graduação (SAPOS) (AMARO; FERREIRA, 2013).

Com o desejo de obter um desenvolvimento menos burocrático e que entregasse ao cliente uma pequena parte do sistema funcionando ao fim de um curto período de tempo, foi escolhida uma metodologia de desenvolvimento ágil. Nessa metodologia, o usuário acompanha de perto o desenvolvimento do *software*, permitindo que seus desejos sejam rapidamente incorporados no produto.

Foram necessárias escolhas de tecnologias que auxiliassem o desenvolvimento no menor tempo possível. Com este propósito, foi escolhida a linguagem *Ruby* com o auxílio do *framework Ruby on Rails*, que ganharam espaço no cenário atual exatamente por esta característica.

Ao fim do desenvolvimento da primeira versão do SAPOS, apesar de todos os requisitos desenvolvidos, o sistema ainda não atendia todas as demandas da coordenação da Pós-Graduação em Computação da UFF. Foi requisitado então o desenvolvimento de uma segunda versão do SAPOS, produto deste trabalho.

Como objetivos do SAPOS 2, podemos citar: a melhoria dos controles de etapas, prorrogações e pontos de orientação, que já existiam na primeira versão do sistema; o desenvolvimento de novas buscas avançadas; a criação de mais relatórios que pudessem ser extraídos no formato PDF; o desenvolvimento do controle de credenciamento de professores; e a criação do controle de disciplinas, que é composto de entidades como turmas, áreas de pesquisa, inscrições, entre outras.

Este trabalho está dividido em quatro capítulos além desta introdução. O Capítulo 2 apresenta outros sistemas relacionados a este trabalho. O Capítulo 3 relata as novas funcionalidades e melhorias desenvolvidas na segunda versão do SAPOS. O Capítulo 4 expõe a metodologia e o processo de desenvolvimento do sistema, além de apresentar algumas das ferramentas utilizadas. O Capítulo 5 contém a conclusão do trabalho, no qual serão discutidos os objetivos alcançados e apresentadas novas possibilidades para a expansão do sistema.

CAPÍTULO 2 – SISTEMAS DE APOIO À PÓS-GRADUAÇÃO

As secretarias e coordenações da Universidade Federal Fluminense têm a necessidade de consultar e manipular uma grande quantidade de dados relacionados à Pós-Graduação. Alguns exemplos são matrículas de alunos, credenciamento de professores, realização de etapas, alocação de bolsas e disciplinas cursadas. O problema é agravado pela descentralização desses dados, que muitas vezes não estão disponíveis em formato digital.

A solução proposta pela UFF é o armazenamento de uma base de dados digital, que pode ser acessada pelas pessoas envolvidas através da interface de um sistema *Web*. Neste capítulo serão descritos dois sistemas que realizam a gerência dos dados acadêmicos relacionados à Pós-Graduação da UFF: (i) SisPós, desenvolvido pela Superintendência de Tecnologia da Informação (STi); e (ii) a primeira versão do SAPOS, desenvolvida para atender às demandas do Programa de Pós-Graduação em Computação.

2.1 SISPÓS

Para a gestão dos dados relacionados à Pós-Graduação está sendo desenvolvido pela STI o Sistema de Gestão da Pós-Graduação (SisPós). Acessado por coordenadores dos cursos, funcionários da coordenação e administradores da PROPPI, o SisPós atua na gerência de diversos dados acadêmicos. Exemplos desses dados são: alunos, editais, cursos e disciplinas, como apresentado na Figura 2.1.

Além deste módulo principal, que faz a gestão dos dados acadêmicos, também estão na fase de aprovação os seguintes módulos:

Inscrição em Editais – Tem como finalidade fornecer uma interface onde os candidatos que pretendem cursar uma Pós-Graduação na UFF possam se inscrever.

Acesso a Alunos – Visa fornecer acesso aos alunos já inscritos na Pós-Graduação, para que possam gerar declarações, alterar seus dados cadastrais e realizar a inscrição em disciplinas.

Existe ainda um módulo chamado “Currículo e Disciplinas”, que já tem algumas funcionalidades prontas, mas ainda está na fase de desenvolvimento. Este módulo estará encarregado do controle de disciplinas, currículos e da vinculação de alunos.

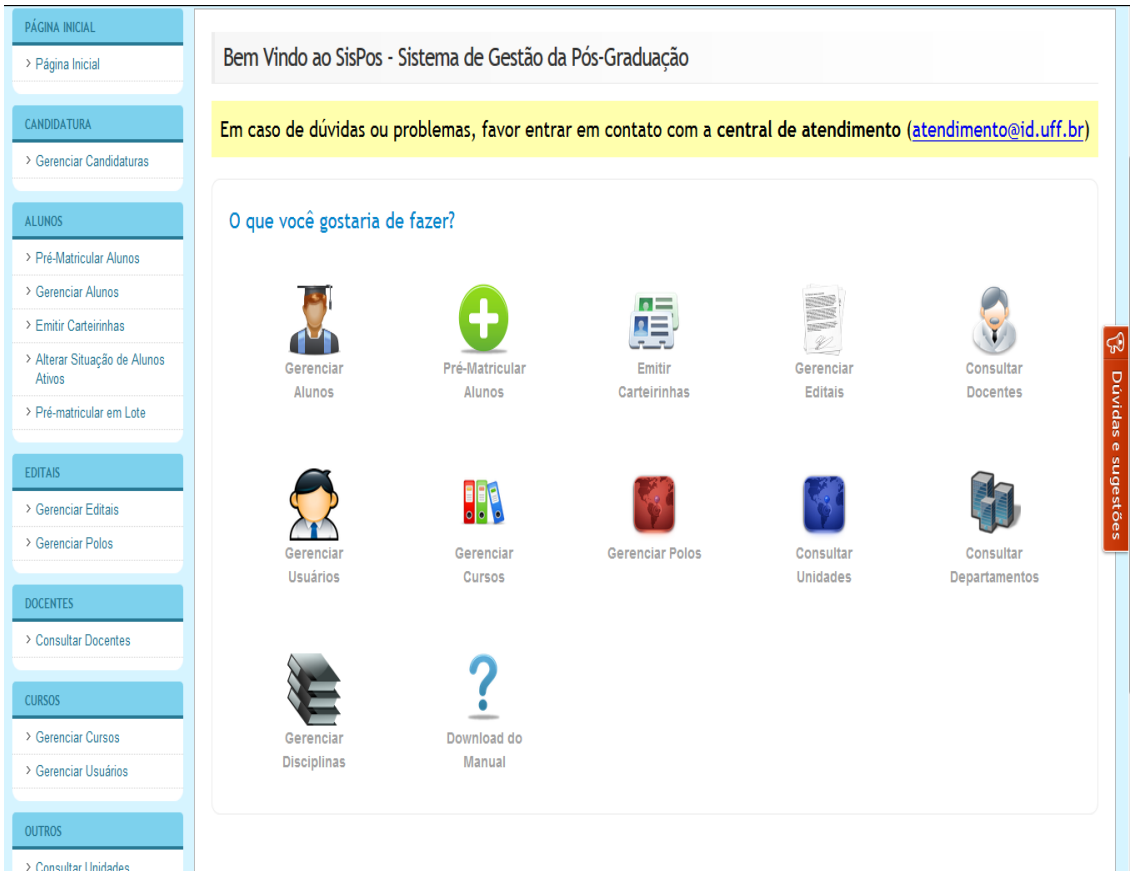


Figura 2.1: SisPós

Outros módulos do sistema já estão previstos, mas ainda não passaram pela fase de análise. São eles:

Pesquisador Colaborador – Cuidará do cadastro de pesquisadores, colaboradores e alunos externos que estejam envolvidos em um programa de Pós-Graduação;

Quadro de Horários – Permitirá a gestão de turmas, o lançamento de notas e a extração do histórico escolar.

Banca e Diplomação – Permitirá a gestão de bancas de avaliação, que podem ser compostas por professores da UFF e professores de outras universidades.

2.2 PRIMEIRA VERSÃO DO SAPOS

Para suprir as necessidades que não são atendidas e nem previstas pelo SisPós, foi desenvolvida a primeira versão do SAPOS. O sistema é acessado somente pelos coordenadores e pela secretaria do Programa de Pós-Graduação em Computação da UFF. Ele faz o controle de sete grupos de dados:

Alunos – Esse grupo controla os dados de alunos, suas matrículas e seus desligamentos do curso. Também é possível cadastrar os tipos de matrículas, motivos de desligamentos e níveis dos cursos oferecidos pela instituição de ensino.

Professores – É responsável pelo cadastro de professores e pelas orientações realizadas pelos mesmos. O sistema na sua primeira versão não guardava dados de credenciamento de professores. Isto fazia com que o número de pontos de orientação concedidos para os professores fosse o mesmo para os credenciados e os não credenciados.

Bolsas – Esse grupo cuida do cadastro de agências de fomento que oferecem bolsas para os alunos de Pós-Graduação. Também é responsável pelo cadastro dos tipos de bolsas e alocação dessas bolsas para os alunos.

Etapas – É responsável pelo controle de realizações de etapas e prorrogações concedidas para adiar o prazo final para a realização das mesmas. O sistema na sua primeira versão fazia o controle de prazos de etapas e prorrogações apenas usando como unidade os meses. Esse fato causava a carência de precisão no controle de etapas e prorrogações, que tem prazos diferenciados, como quantidade de semestres letivos e dias.

Formação – Cuida do controle de instituições de ensino e cursos. Com estas informações, é possível conhecer o histórico de formação de um aluno antes de ingressar no Programa de Pós-Graduação. Por exemplo, é possível saber o em qual instituição um aluno cursou a sua Graduação e seu Mestrado antes de ser aceito como aluno de Doutorado no Programa de Pós-Graduação em Computação da UFF.

Localidades – Nesta seção são armazenados dados de cidades, estados e países. Com esses dados é possível guardar informações como a naturalidade e residência atual de alunos.

Configurações – Esta seção é destinada a tratar de todas as configurações necessárias para a administração do sistema. Na primeira versão do SAPOS, esta seção comporta apenas o controle de usuários autorizados a acessar o sistema.

O sistema oferece dois tipos de interface para buscar registros em suas telas: a busca simples, e a busca avançada. A busca simples, ilustrada na Figura 2.2, procura pelo texto digitado na caixa de busca em todas as colunas dos registros. Caso o texto seja encontrado em pelo menos uma coluna de um registro, esse registro é exibido na tela como resultado da busca. Pela sua facilidade técnica, esse tipo de busca é o mais utilizado no sistema.

A busca avançada, ilustrada na Figura 2.3, procura por condições específicas de cada coluna escolhida para a consulta, permitindo que ela seja mais precisa. Esta busca é mais poderosa, pois enquanto a busca simples faz apenas a consulta de um texto em qualquer coluna, a busca avançada tem a capacidade de fazer a união da consulta específica para cada coluna dos registros. Por exemplo, com a busca avançada podemos consultar as bolsas ativas no nível "Doutorado" cedidas por uma determinada Agência de Fomento, o que não seria possível com a busca simples.

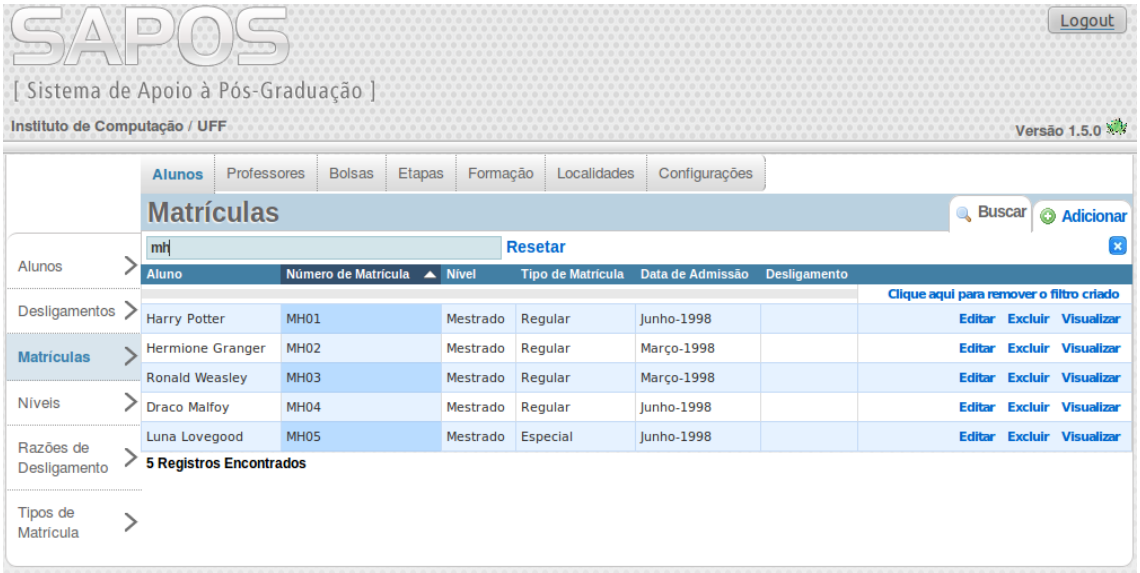
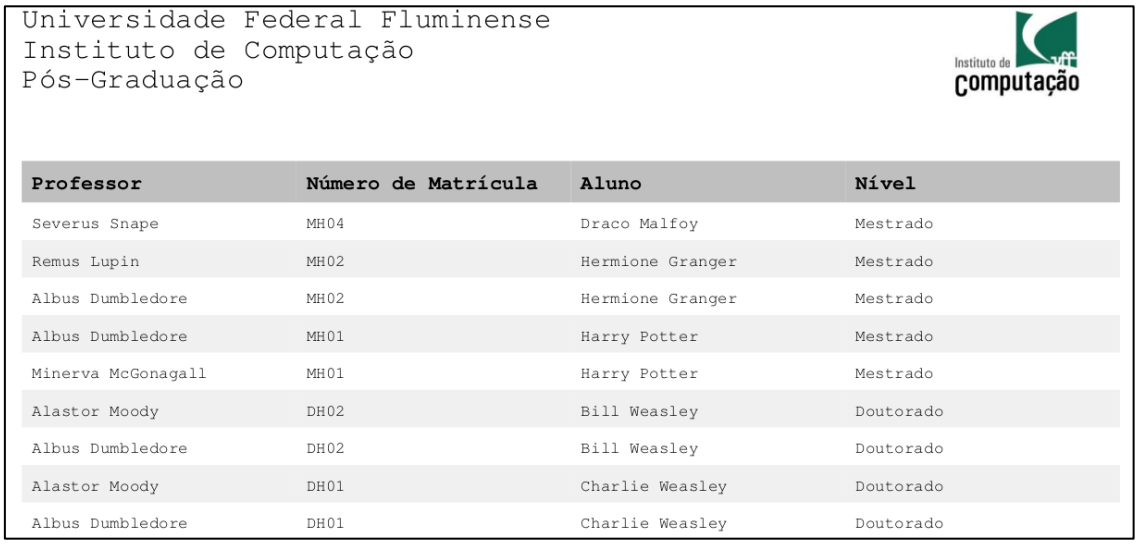


Figura 2.2: Busca Simples

Apenas as telas de bolsas, alocação de bolsas e orientações possuíam a busca avançada. O sistema na sua primeira versão possuía outras telas que necessitavam consultas complexas, mas não dispunham da busca avançada. Um exemplo é a tela de matrículas, que possui uma grande quantidade de informação para ser consultada, mas apresentava somente a opção de busca simples.

Figura 2.3: Busca Avançada

É possível extrair relatórios no formato PDF nas telas de bolsas, alocação de bolsas e orientações. Os dados exportados para esses relatórios podem ser filtrados por meio da busca e, assim, gerar um documento somente com as informações relevantes para cada caso. A Figura 2.4 ilustra um exemplo de relatório gerado pelo sistema no formato PDF.



Universidade Federal Fluminense
Instituto de Computação
Pós-Graduação

Professor	Número de Matrícula	Aluno	Nível
Severus Snape	MH04	Draco Malfoy	Mestrado
Remus Lupin	MH02	Hermione Granger	Mestrado
Albus Dumbledore	MH02	Hermione Granger	Mestrado
Albus Dumbledore	MH01	Harry Potter	Mestrado
Minerva McGonagall	MH01	Harry Potter	Mestrado
Alastor Moody	DH02	Bill Weasley	Doutorado
Albus Dumbledore	DH02	Bill Weasley	Doutorado
Alastor Moody	DH01	Charlie Weasley	Doutorado
Albus Dumbledore	DH01	Charlie Weasley	Doutorado

Figura 2.4: Relatório de orientações

2.3 DISCUSSÃO

Neste capítulo foram apresentados o sistema SisPós e o sistema SAPOS na sua primeira versão. Ambos os sistemas fazem o controle de uma grande quantidade de dados, mas isoladamente os sistemas ainda não conseguem satisfazer a todas as necessidades da coordenação do Programa de Pós-Graduação em Computação da UFF. Existe a necessidade de gerenciar dados referentes a disciplinas, inscrição em turmas, alocação de salas, entre outros. Esses dados ainda precisam ser cadastrados e mantidos em outras vias, como formulários em papel e bases de dados digitais dispersas. A Tabela 1 mostra a comparação das funcionalidades atendidas pela primeira versão do SAPOS e pelo SisPós.

Funcionalidade/Sistema	SAPOS	SisPós
Controle de Alunos	Sim	Sim
Controle de Professores	Sim	Sim
Controle de Pontos de Orientação de Professores	Sim	Não
Controle de Bolsas	Sim	Não
Controle de Etapas	Sim	Não
Controle de Editais	Não	Sim
Controle de Cursos	Não	Sim
Controle de Polos	Não	Sim
Controle de Unidades	Não	Sim
Controle de Departamentos	Não	Sim

Tabela 1: Comparação entre a primeira versão do SAPOS e o SisPós

No próximo capítulo são apresentadas as novas funcionalidades e melhorias implementadas no SAPOS, gerando sua segunda versão, que buscaram atender uma parcela maior da necessidade atual do Programa de Pós-Graduação em Computação da UFF.

CAPÍTULO 3 – SAPOS 2

Como discutido anteriormente, a primeira versão do SAPOS foi desenvolvida para centralizar informações referentes à Pós-Graduação em Computação da UFF. Informações estas que eram mantidas em forma de papel ou armazenadas em bancos de dados no *Microsoft Excel* e *Access*.

Este capítulo apresenta as melhorias e novas funcionalidades implementadas no sistema, que ao fim geraram o SAPOS 2. A Seção 3.1 apresenta o novo controle de Etapas e Prorrogações. A Seção 3.2 apresenta a busca avançada implementada na tela de Matrículas. A Seção 3.3 apresenta o controle de Credenciamentos. A Seção 3.4 apresenta o controle de Disciplinas. A Seção 3.5 apresenta os novos relatórios que podem ser extraídos. Por fim, a Seção 3.6 apresenta a discussão sobre o capítulo.

3.1 NOVO CONTROLE DE ETAPAS E PRORROGAÇÕES

O controle de etapas e prorrogações do SAPOS na sua primeira versão limitava-se a contabilizar os prazos apenas em meses, o que fazia com que o controle de prazos tivesse algumas limitações.

No caso do IC, os semestres letivos são iniciados nos meses de Março e Agosto de cada ano. Por esse fato, os semestres letivos que têm início em Março têm duração de 5 meses, enquanto os semestres letivos iniciados em Agosto têm duração de 7 meses. Para as etapas que têm duração de um semestre letivo, por exemplo, era necessário o cadastro de duas instâncias de etapas: uma com duração de 5 meses, para atender aos casos dos semestres letivos iniciados em Março e outra com duração de 7 meses, para atender aos casos dos iniciados em Agosto. No SAPOS 2 foi aprimorado o controle de semestres letivos, no qual é possível saber, para um dado semestre, a data de início, a data de fim e realizar operações como somar e subtrair semestres letivos. A informação de um semestre letivo é composta de um ano e um semestre.

A data de início de um semestre letivo é identificada da seguinte forma: se o semestre é o primeiro semestre do ano, a data de início do semestre letivo é o dia 1 de Março do ano em questão; caso contrário, a data de início do semestre letivo é o dia 1 de Agosto do ano em questão. Por exemplo, para o segundo semestre do ano 2012, a data de início seria 1 de Agosto de 2012.

A data de fim de um semestre letivo é o dia anterior ao início do próximo semestre letivo. Com essa abordagem, a data de fim do segundo semestre de 2011, seria identificada como 29 de Fevereiro de 2012.

Para adição de semestres a um semestre letivo escolhido, é utilizado o algoritmo em pseudocódigo é o apresentado na Figura 3.1.

```
1   Se N é par
2       RESULTADO.ano = ANOLETIVO.ano + N/2
3       RESULTADO.semestre = ANOLETIVO.semestre
4   Senão
5       Se ANOLETIVO.semestre é 1
6           RESULTADO.ano = ANOLETIVO.ano + N/2
7           RESULTADO.semestre = 2
8       Senão
9           RESULTADO.ano = ANOLETIVO.ano + (N/2) + 1
10          RESULTADO.semestre = 1
11      fim-Se
12  fim-Se
```

Figura 3.1: Algoritmo para adição de semestres letivos

Existem ainda casos de etapas e prorrogações que têm prazos que não são possíveis de contabilizar apenas com meses e semestres letivos. Por exemplo, uma etapa ou prorrogação que tenha um prazo de 45 dias não poderia ser atendida com o modelo anterior.

Com a implementação atual é possível criar etapas e prorrogações com prazos flexíveis, fazendo a união das unidades “dias”, “meses” e “semestres letivos”. Por exemplo, pode ser criada uma etapa ou prorrogação que tenha prazo de quatro semestres letivos, três meses e quinze dias, como mostra a Figura 3.2.

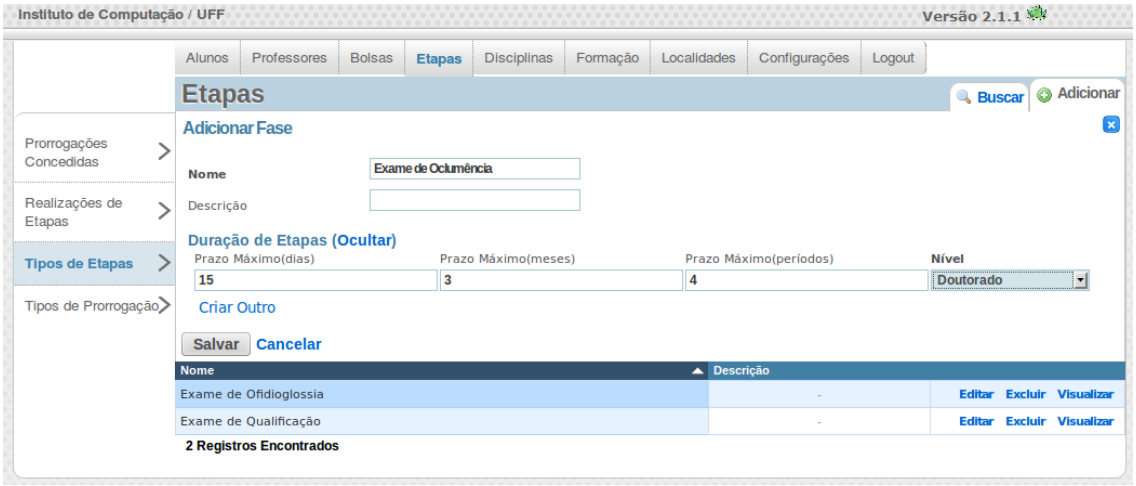


Figura 3.2: Nova criação de etapas

3.2 NOVA BUSCA AVANÇADA

Como visto no capítulo anterior, a busca simples é muito limitada por não conseguir unir duas ou mais consultas e nem buscar em uma coluna específica da tabela. Como a entidade Matrícula tem um grande número de conexões com outras entidades do sistema, é fundamental que possa ser feita uma busca detalhada na tela de Matrículas.

Com a busca avançada que foi implementada, é possível fazer consultas relacionadas apenas aos dados da matrícula e consultas relacionadas a outras entidades do sistema. Fazem parte do grupo de dados da própria matrícula: o número da matrícula, nível, tipo de matrícula e data de admissão. A interface da busca avançada na tela de matrículas é apresentada na Figura 3.3. A Figura 3.4 apresenta o modelo simplificado da entidade Matrícula (*Enrollment*).

As buscas relacionadas a outras entidades do sistema são:

Nome do aluno: Faz uma consulta pelo nome do aluno, que está armazenado na entidade Aluno e não em Matrícula;

Ativo: Consulta se a matrícula tem algum desligamento associado;

Possui bolsa: Busca se a matrícula está associada a alguma bolsa ativa;

Orientador: Consulta pelo nome dos professores que são orientadores da matrícula;

Realização de Etapa: Faz uma busca nas etapas realizadas da matrícula. Caso esteja registrado que a matrícula cumpriu a etapa selecionada em uma data igual ou anterior à data indicada, a matrícula é retornada pela busca.

Etapa atrasada: Consulta as etapas realizadas e as prorrogações concedidas da matrícula. O prazo para cumprimento de uma etapa é contado a partir da data de admissão da matrícula. As prorrogações concedidas adicionam tempo extra para cumprimento das etapas. Caso o prazo para o cumprimento da etapa selecionada,

acrescido do tempo concedido pelas prorrogações, seja maior que a data selecionada, a matrícula é retornada pela busca.

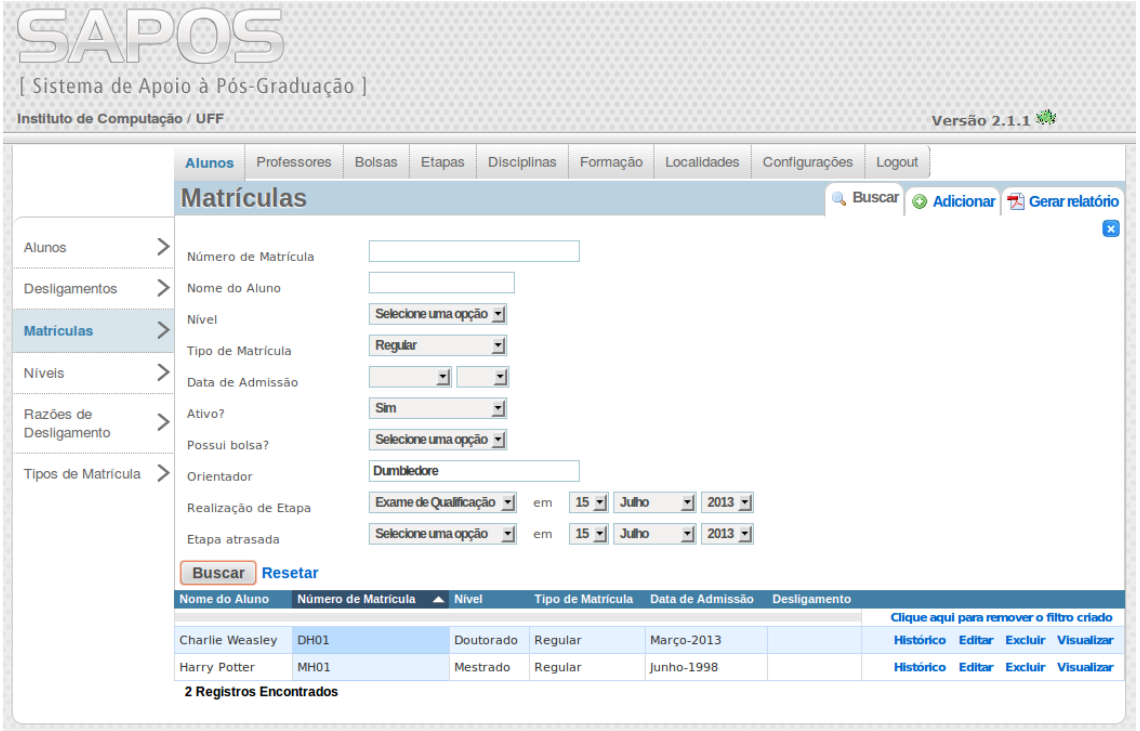


Figura 3.3: Nova Busca Avançada

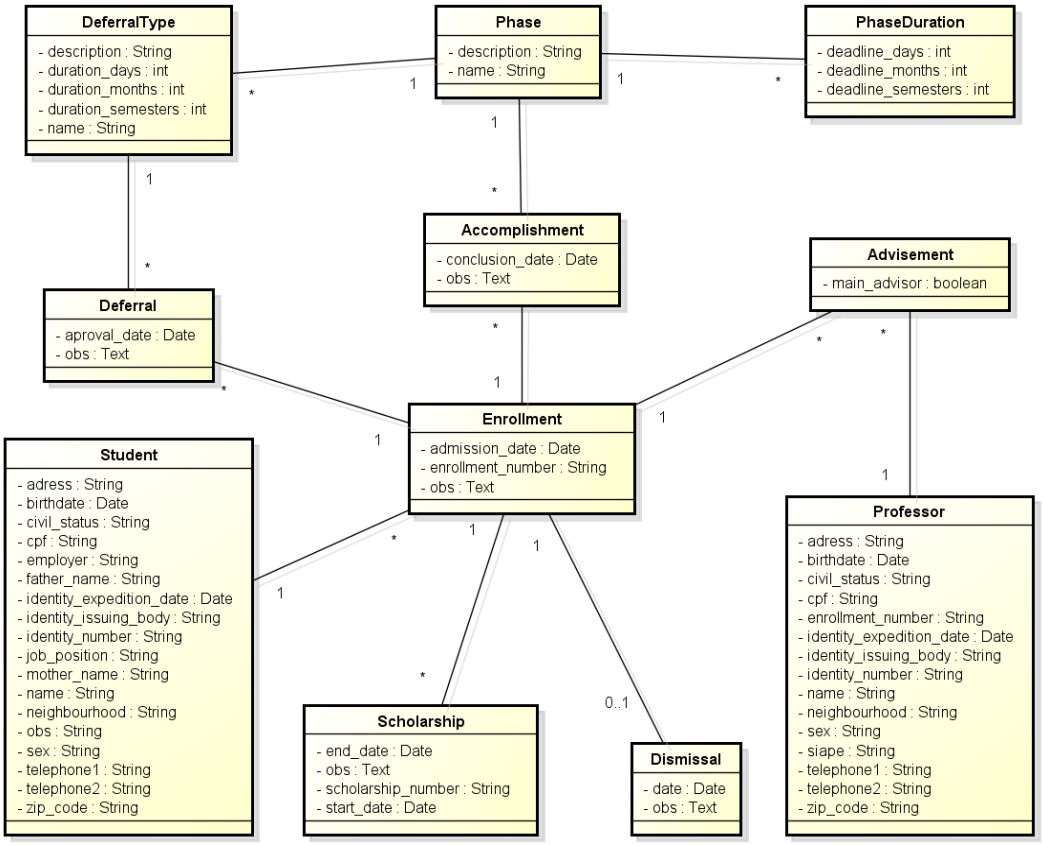


Figura 3.4: Modelo simplificado de Matrícula

3.3 CONTROLE DE CREDENCIAMENTOS

Para controlar o número máximo de alunos que cada professor pode orientar, foi convencionado pelo colegiado do Programa de Pós-Graduação em Computação da UFF o uso de pontos de orientação. Porém, a primeira versão do SAPOS assumia que todos os professores cadastrados eram credenciados no Programa de Pós-Graduação. Por este fato, se professores externos atuassem na orientação de alunos, e esses professores fossem cadastrados no SAPOS, o cálculo dos pontos de orientação era feito de maneira incorreta, dividindo os pontos com o professor externo. A Figura 3.5 apresenta o modelo criado para o controle de credenciamentos.

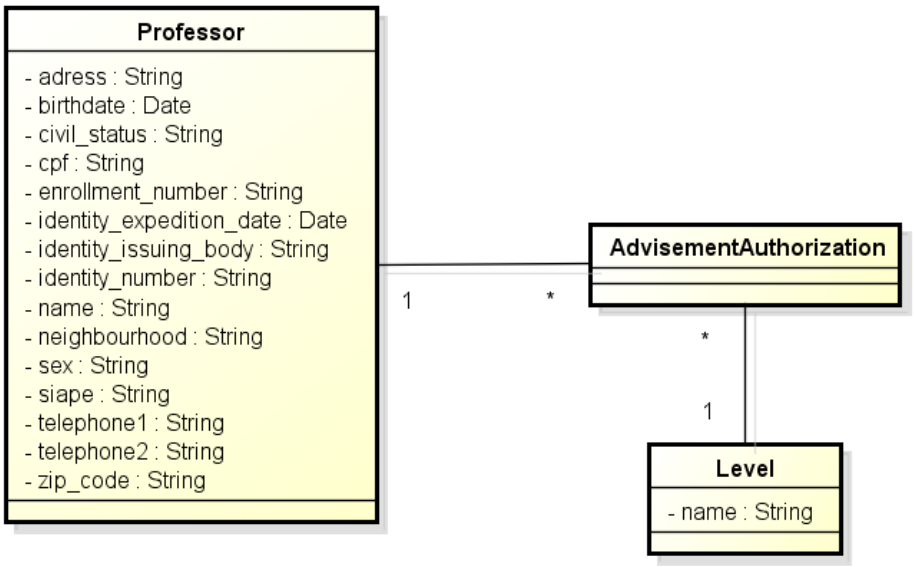


Figura 3.5: Modelo de credenciamentos

Para o cálculo correto dos pontos de orientação, é necessário o conhecimento de informações de credenciamento de professores. Na primeira versão do SAPOS, os pontos de orientação eram calculados para cada orientação da seguinte maneira: se o aluno fosse orientado por apenas um professor, este ganharia 1 ponto de orientação; se o aluno fosse orientado por dois ou mais professores, cada um deles receberia 0,5 ponto de orientação.

Pela regra de pontos de orientação em vigor no Programa de Pós-Graduação em Computação da UFF, um professor só deve receber pontos de orientação se for credenciado. Com o cadastro de dados de credenciamentos de professores passou a ser possível então realizar o cálculo de pontos de orientação da maneira correta: se o aluno for orientado por apenas um professor credenciado, mesmo que seja co-orientado por

outro professor não credenciado, o professor credenciado recebe 1 ponto de orientação; se o aluno é orientado por dois ou mais professores credenciados, cada um deles recebe 0,5 ponto de orientação.

Para possibilitar esse controle foram construídas interfaces de listagem, cadastro, visualização, edição e exclusão de credenciamentos. Para a consulta dos dados cadastrados, foi implementada a busca avançada na listagem de credenciamentos, com a qual é possível buscar os credenciamentos pelo nome do professor e pelo nível desejado. Um exemplo da interface de credenciamentos é exibido na Figura 3.6.

Figura 3.6: Controle de Credenciamentos

Foi alterada também a interface de visualização de professores, para que ficassem mais claros os dados sobre as orientações de um dado professor. A forma de exibição implementada na primeira versão do SAPOS era a listagem dos nomes de todos os alunos orientados pelo professor com suas respectivas matrículas, exibida na Figura 3.7. Para uma visualização mais organizada e fácil de ler, foi inserida uma tabela com as mesmas informações anteriores e adicionada a quantidade de pontos de orientação referente a cada orientação. Além disso, essa visualização agora considera apenas os alunos ativos. Isso significa que no momento em que o aluno é desligado, os pontos de orientação relativos a ele não são mais computados. Um exemplo é ilustrado na Figura 3.8.

Orientandos **MH01 - Harry Potter, DH02 - Bill Weasley, MH02 - Hermione Granger, DH01 - Charlie Weasley**

Figura 3.7: Lista de orientandos na primeira versão do SAPOS

Credenciamentos
Orientandos

Doutorado, Mestrado

Nome do Aluno	Número de Matrícula	Pontos de Orientação
Harry Potter	MH01	0.5
Bill Weasley	DH02	0.5
Hermione Granger	MH02	1.0
Charlie Weasley	DH01	0.5
Total		2.5

Figura 3.8: Tabela de Orientações

3.4 CONTROLE DE DISCIPLINAS

O Controle de Disciplinas é uma seção nova que engloba o controle de diversos dados referentes a disciplinas, como áreas de pesquisa, tipos de disciplinas, inscrição de alunos em turmas e alocação de horários e salas para as turmas criadas. O modelo de classes criado para este controle é apresentado na Figura 3.9.

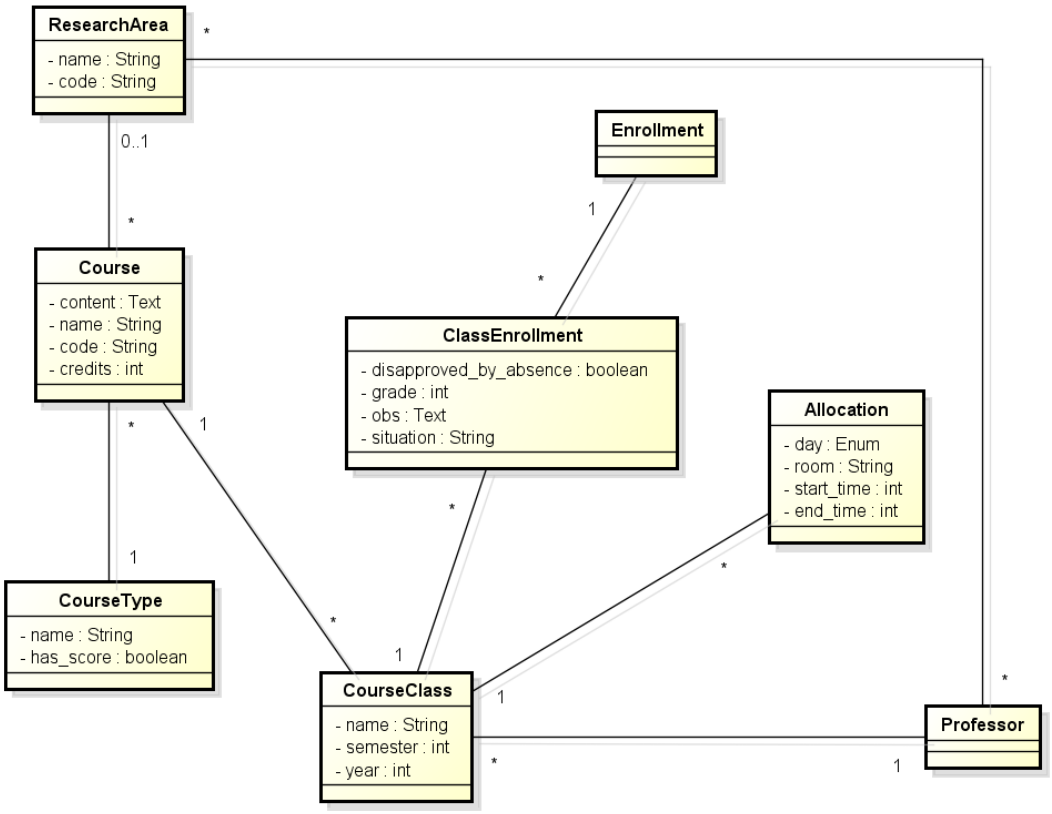


Figura 3.9: Modelo do controle de disciplinas

A seção de Disciplinas, ilustrada na Figura 3.10, é dividida nas seguintes áreas:

Áreas de Pesquisa: Faz o controle das áreas de pesquisa em que as disciplinas são agrupadas. Uma área é identificada pelo seu nome e código;

Tipos de Disciplinas: É responsável pelo controle dos diferentes tipos de disciplinas, e suas propriedades, como, por exemplo, se a disciplina requer lançamento de nota ou não. Exemplos de tipos de disciplinas são: obrigatórias do curso; obrigatórias de uma determinada área de pesquisa; e estudo dirigido;

Disciplinas: É o centro da seção, no qual são controladas as disciplinas oferecidas. Cada disciplina é associada a um tipo de disciplina e pode ser associada a uma área de pesquisa. Além disso, uma disciplina armazena outros dados como nome, código e quantidade de créditos;

Turmas: Representam as turmas criadas em cada semestre letivo para as disciplinas, associadas aos professores que as lecionam;

Inscrições: Faz o controle da inscrição de um aluno em uma turma, armazenando dados como a nota e situação atual. Foi criado um controle para impedir o cadastro de dados inválidos como, por exemplo, cadastrar a nota 10 para a situação “Reprovado” ou marcar o campo “Reprovado por falta” para a situação “Aprovado”;

Alocações: Armazenam dados de horários e identificação das salas para as turmas criadas. Foi criado um controle para evitar problemas como cadastrar duas alocações da mesma turma em horários conflitantes ou inválidos, fazendo com que os dados cadastrados sejam válidos. Com esse controle, o sistema não permite que uma alocação da turma “A”, na segunda-feira das 9 às 11 horas, se já existisse uma alocação da turma “A”, na segunda feira das 8 às 10 horas. O sistema também não permite que seja cadastrada uma alocação com a hora de início maior do que hora de fim.

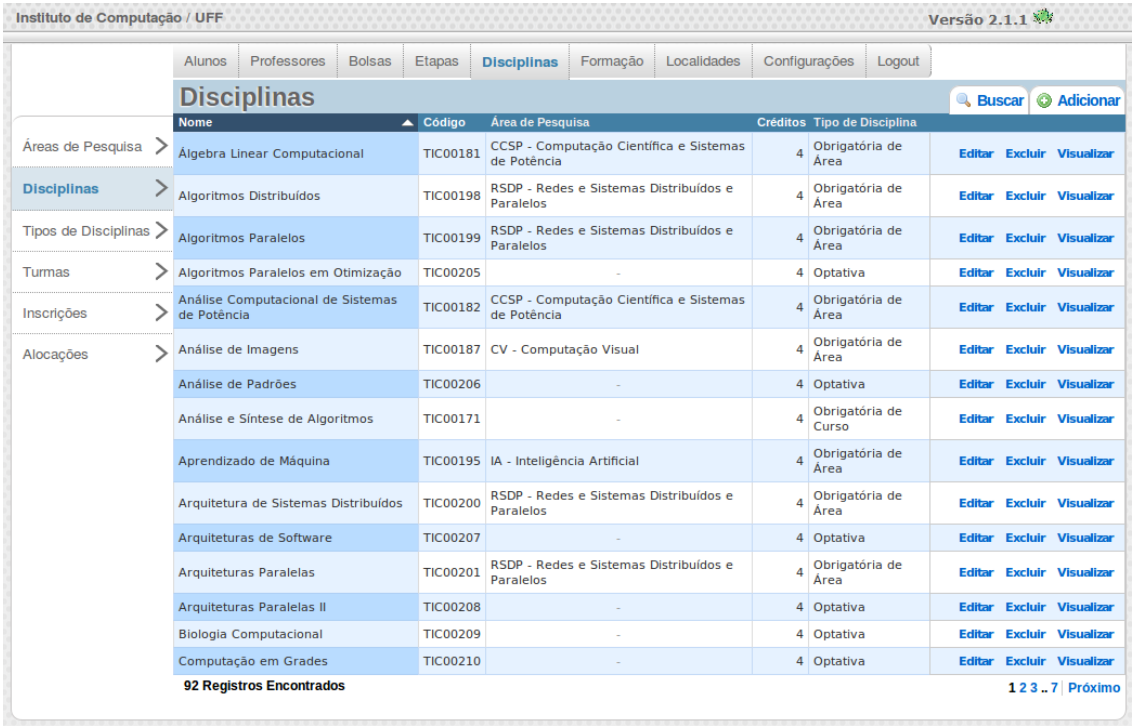


Figura 3.10: Seção de Disciplinas

3.5 NOVOS RELATÓRIOS

Foram criados três novos relatórios no formato pdf, para que permitam a impressão de forma mais organizada dos dados cadastrados no banco de dados do sistema.

O primeiro deles é o relatório da listagem de matrículas. Assim como o relatório que já existia na listagem de bolsas na primeira versão do SAPOS, este relatório transfere para um arquivo no formato pdf os dados da listagem da página, incluindo os filtros aplicados.

O segundo relatório é a pauta de turmas. Os professores das respectivas turmas o preenchem no final do semestre letivo com dados como a situação e nota de cada aluno. Após ser entregue na secretaria, esta cadastra os dados no sistema. A Figura 3.11 apresenta o exemplo de uma pauta de turma extraída do sistema.

O último deles é o histórico dos alunos, ilustrado na Figura 3.12. Ele contém os dados de todas as disciplinas cursadas. Como o sistema ainda não é acessado por alunos, é necessário que aquele que necessitar de seu histórico se dirija à secretaria para requisitá-lo.

Universidade Federal Fluminense
Instituto de Computação
Pós-Graduação

RESUMO SEMESTRAL

NOME DA DISCIPLINA	SEMESTRE/ANO	AULAS DADAS
Defesa Contra as Artes das Trevas	1º/2013	

Nº	Matrícula	Nome do Aluno	Nota Final	Freq S/I	Situação	Obs.:
1	DH01	Charlie Weasley	10.0	S	Aprovado	
2	DH02	Bill Weasley			Incompleto	
3	MH01	Harry Potter			Incompleto	
4	MH02	Hermione Granger			Incompleto	
5	MH03	Ronald Weasley			Incompleto	
6	MH04	Draco Malfoy			Incompleto	
7	MH05	Luna Lovegood	0.0	I	Reprovado	

Severus Snape

Figura 3.11: Pauta extraída do SAPOS 2

Universidade Federal Fluminense
Instituto de Computação
Programa de Pós-Graduação em Computação

Histórico Escolar

Aluno: Charlie Weasley

Matrícula: DH01 Naturalidade: Não informado Data de Nascimento: Não informado

Identidade: Não informado Órgão Expeditor: Não informado

C.P.F.: 00000000000

Curso: Doutorado em Computação

Mês/Ano de ingresso no curso: Março/2013

Código	Nome da Disciplina	Nota	Créditos	Período
001	Laboratório de Poções Mágicas I	9.0	4	1/13
002	Defesa Contra as Artes das Trevas	10.0	6	1/13
003	Vôo I	AP	2	1/13
	Total		12	

Etapas obrigatórias concluídas
Maio/2013 Exame de Ofidioglossia
Julho/2013 Exame de Qualificação

Orientadores: Alastor Moody, Albus Dumbledore

Niterói, 15 de Julho de 2013

Este documento só é válido sem rasuras, com selo da UFF e com a assinatura do Coordenador.

ASSINATURA DO COORDENADOR

Pág. nº 1

Figura 3.12: Histórico extraído do SAPOS 2

3.6 DISCUSSÃO

Neste capítulo vimos as novas funcionalidades implementadas no SAPOS 2. O sistema agora atende uma maior porção das necessidades de armazenamento de dados do escopo da Pós-Graduação em Computação da UFF. Além disso, foram criadas novas interfaces para facilitar o entendimento dos dados já presentes na primeira versão do SAPOS e criados novos relatórios no formato PDF. A Tabela 2 mostra a comparação das funcionalidades presentes na primeira e segunda versão do SAPOS.

Funcionalidade/Sistema	SAPOS	SAPOS 2
Controle de Alunos	Sim	Sim
Controle de Professores	Sim	Sim
Controle de Pontos de Orientação de Professores	Sim	Sim
Controle de Credenciamento de Professores	Não	Sim
Controle de Pontos de Orientação de Professores levando em conta seus credenciamentos	Não	Sim
Controle de Bolsas	Sim	Sim
Controle de Etapas e Prorrogações com prazos contabilizados em meses	Sim	Sim
Controle de Etapas e Prorrogações com prazos contabilizados em semestres letivos e dias	Não	Sim
Controle de Disciplinas	Não	Sim

Tabela 2: Comparação das funcionalidades presentes nas duas versões do SAPOS

No capítulo a seguir é discutido o processo e metodologia de desenvolvimento utilizados durante a implementação dessas funcionalidades no SAPOS 2.

CAPÍTULO 4 – DESENVOLVIMENTO

Com as diversas tecnologias para desenvolvimento de sistemas *Web* existentes no mercado, é necessário que façamos escolhas das ferramentas que mais se aplicam ao contexto do sistema a ser desenvolvido. Entre essas escolhas estão: a linguagem de programação; os *frameworks* e bibliotecas disponíveis para a linguagem de programação escolhida; e os padrões de projeto a serem implementados.

Além das escolhas referentes à programação em si, ainda devemos fazer outras escolhas para definir o processo de desenvolvimento do sistema. Entre essas estão o planejamento do projeto, o monitoramento e controle do projeto e gerência de configuração, que engloba o controle de versões e o controle de modificações.

Este capítulo tem como objetivo apresentar e justificar as escolhas feitas para o processo de desenvolvimento do SAPOS 2 e está organizado como segue. A Seção 4.1 apresenta a metodologia utilizada. A Seção 4.2 explica o padrão de projeto MVC. A Seção 4.3 relata detalhes da implementação. A Seção 4.4 expõe as ferramentas auxiliares. A Seção 4.5 apresenta os testes automatizados implementados. Por fim, a Seção 4.6 contém discussões finais do capítulo.

4.1 METODOLOGIA

Existem diversas atividades no desenvolvimento de um *software*. Segundo PRESSMAN (2010), elas são cinco:

1. **Comunicação:** Uma parte ou a totalidade da equipe que irá desenvolver o *software* se comunica com o usuário ou um representante direto do mesmo para que sejam levantados os requisitos do sistema.

2. **Planejamento:** É estimado o tempo para implementação de cada um os requisitos identificados na fase de comunicação. Os requisitos são agendados de acordo com a duração estimada. Nesta fase também é feita a análise de riscos.

3. **Modelagem:** É feita a análise de cada requisito e, a partir dela, é gerado um modelo técnico para o requisito.

4. **Construção:** Os requisitos são codificados e testados a partir dos modelos especificados previamente.

5. **Implantação:** Os requisitos construídos são apresentados ao usuário para que ele valide se está de acordo com o que foi imaginado inicialmente por ele. O usuário responde à equipe notificando se o requisito foi construído corretamente e pedindo ajustes para se adequar à solução ideal.

Essas atividades podem ser realizadas sequencialmente ou em pequenos ciclos, dependendo da metodologia de desenvolvimento adotada. Há uma divisão em dois grandes grupos de metodologias para desenvolvimento de *software*: tradicionais e ágeis.

Nos anos 70, surgiu a metodologia Cascata, uma das mais conhecidas metodologias tradicionais. A Figura 4.1 ilustra o processo na metodologia Cascata.

Figura 4.1: Processo na metodologia Cascata

As metodologias ágeis sugerem que as atividades sejam realizadas em iterações. As atividades devem ser executadas para apenas poucos requisitos que são apresentados ao usuário no fim de um curto período de tempo. Esta prática tem a finalidade de identificar o mais cedo possível se o *software* que está sendo construído de fato é o *software* que o usuário quer. A Figura 4.2 apresenta um processo típico de uma metodologia ágil.

A prática de projetar o sistema por inteiro para só depois começar a codificá-lo é obsoleta no cenário atual (PRESSMAN, 2010). Dessa forma, a metodologia utilizada no desenvolvimento desse projeto foi aderente a metodologia ágil. Apesar de não terem sido utilizadas iterações de tamanho fixo, os requisitos implementados foram apresentados para os usuários logo após o seu desenvolvimento, evitando assim que todas as funcionalidades fossem apresentadas no fim do projeto e com grande chance de

não serem aprovados pelos usuários. O processo utilizado foi a repetição das seguintes tarefas:

1 - Levantamento do requisito, feito com os usuários;

2 – Análise do requisito;

3 – Desenvolvimento e testes;

4 – Apresentação do requisito para os usuários;

5 – Pedidos de ajustes, feitos pelos usuários;

6 – Desenvolvimento e teste dos ajustes requisitados.

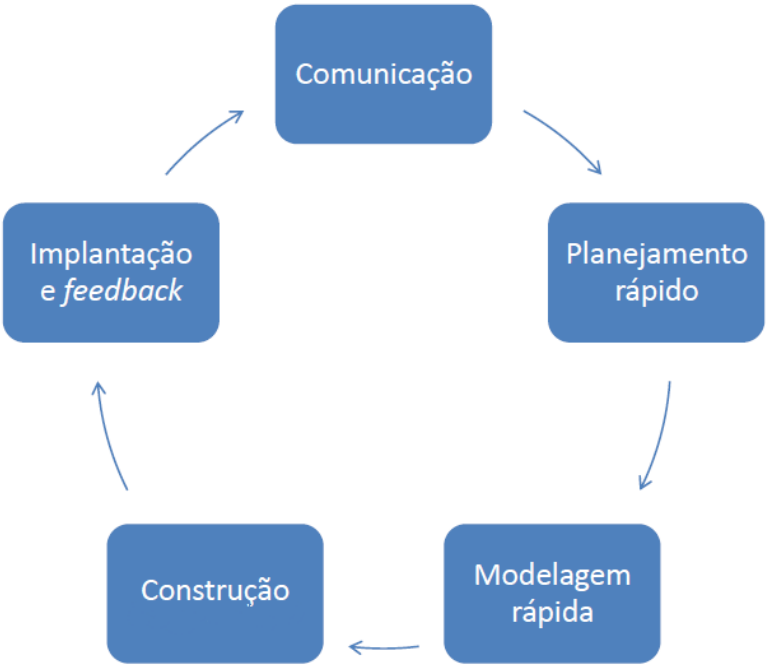


Figura 4.2: Processo típico de metodologia ágil

Foram realizadas algumas reuniões presenciais dos desenvolvedores com os usuários, mas a maior parte da comunicação foi feita por troca de e-mails entre eles. Também foi utilizado o *Redmine* (LANG, 2006), que é apresentado na Seção 4.4 deste capítulo.

4.2 PADRÃO MVC

Para entender o uso do *framework Ruby on Rails*, que é o principal *framework* utilizado no desenvolvimento do projeto, é preciso entender o padrão MVC. O padrão MVC, ou Modelo-Visão-Controle, é um estilo arquitetural que tem como objetivo

separar o código que contém a lógica da aplicação do código que contém a interface com o usuário (KRASNER; POPE, 1988). Essa separação é feita visando uma melhor organização e a independência dos três componentes, que são:

(i) **Modelo:** Componente que é responsável por armazenar as entidades e regras de negócio da aplicação.

(ii) **Visão:** É responsável pela apresentação da interface da aplicação com o usuário.

(iii) **Controle:** Faz a interação entre a camada do Modelo e da Visão. No caso de um sistema *web* é responsável por receber as requisições feitas à aplicação.

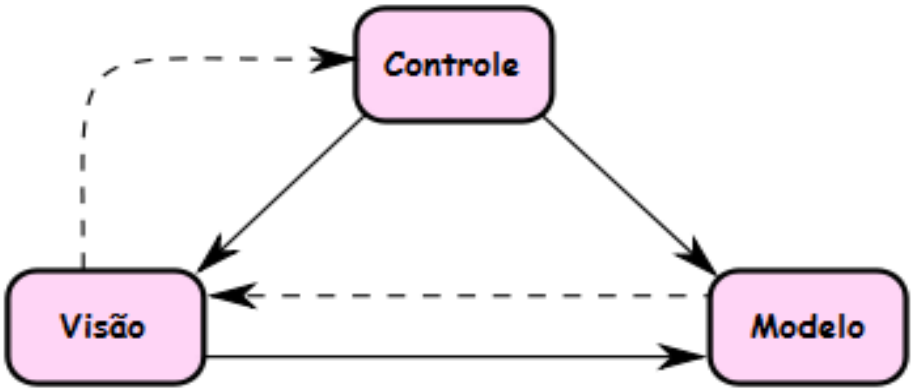


Figura 4.3: Interação entre as camadas no padrão MVC[1]

A interação entre as três camadas, representada na Figura 4.3, pode acontecer das seguintes maneiras:

- O usuário interage com a Visão
- O Controlador pede ao Modelo para modificar seu estado
- O Controlador pede para a Visão se modificar
- O Modelo notifica a Visão quando seu estado é modificado
- A Visão requisita um estado ao Modelo

[1] Adaptado de http://www.htmlgoodies.com/img/2010/11/mvc.png

4.3 IMPLEMENTAÇÃO

Para viabilizar o desenvolvimento, é necessário escolher as ferramentas a serem utilizadas. A linguagem de programação, seus *frameworks* e bibliotecas auxiliares são algumas das escolhas mais importantes a serem feitas. Com o objetivo de agilizar o desenvolvimento do projeto, foi escolhida na primeira versão do SAPOS a linguagem *Ruby* em conjunto com o *framework Ruby on Rails* e suas bibliotecas auxiliares. Devido ao sucesso alcançado, estes componentes foram mantidos no SAPOS 2.

4.3.1 RUBY

A linguagem *Ruby* (MATSUMOTO, 1995) é orientada a objetos que ficou conhecida por sua alta legibilidade e facilidade de desenvolvimento. Algumas características dessa linguagem são a tipagem dinâmica e o fato de todos os elementos, até mesmo os tipos primitivos, serem considerados objetos.

4.3.2 RUBY ON RAILS

Ruby on Rails (THOMAS; HANSSON, 2011) é um *framework* que visa facilitar o desenvolvimento de sistemas *web* orientados a bancos de dados seguindo o padrão MVC.

Os princípios mais presentes no *framework* são:

- CoC (*Convention over Configuration*, ou Convenção sobre Configuração): Tem o objetivo de diminuir a necessidade de configurações por meio da padronização de regras de organização e nomenclatura.
- DRY (*Don't Repeat Yourself*, ou Não Se Repita): Visa reduzir a repetição e aumentar o reuso de informações. Segundo HUNT e THOMAS (1999), cada porção do conhecimento deve ter uma representação não ambígua e única dentro de um sistema.

Uma técnica de desenvolvimento muito utilizada pelo *framework Ruby on Rails* é o ORM (*Object-Relational Mapping*, ou Mapeamento Objeto-Relacional). O ORM é utilizado para representar tabelas do banco de dados como classes do sistema. Da mesma forma, as tuplas do banco de dados são representadas como instâncias das classes. A interface do ORM encapsula os comandos em SQL em seus métodos,

fazendo com que o programador apenas chame um método da interface para realizar uma consulta no banco de dados e transformar o seu resultado em instâncias de objetos da linguagem utilizada.

O *Ruby on Rails* é constituído por diversos módulos. Um deles é o *Active Record*, que é responsável pela implementação do ORM. Com o auxílio dessa ferramenta, o *framework* utiliza arquivos de migrações do banco de dados quando é necessário manipular a estrutura do banco de dados. Muitas vezes, esses arquivos de migrações são criados automaticamente pela execução de comandos de geração do próprio *Ruby on Rails*. Por exemplo, para a criação da tabela de Inscrições (Class Enrollment), representada na Figura 4.4, foi gerada automaticamente a migração apresentada na Figura 4.5.

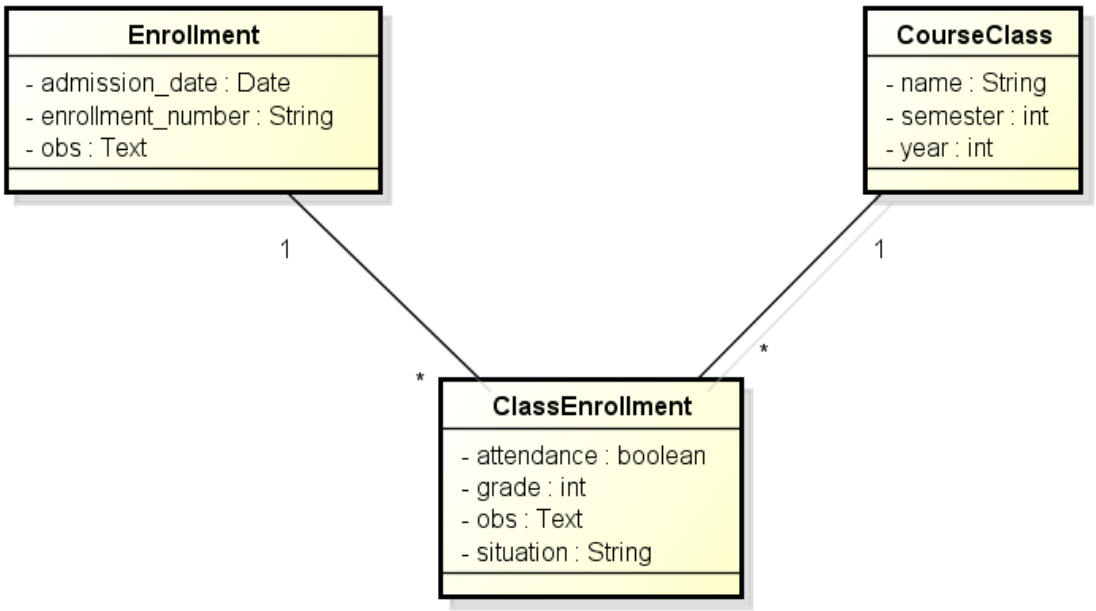


Figura 4.4: Modelo de Inscrições

É possível a utilização de todos os tipos de dados mais comuns em bancos de dados. Além dos tipos comuns de dados, é possível ver na Figura 4.5 o uso do termo *references*, nas linhas 8 e 9. Quando este método é utilizado, é criada automaticamente a coluna do tipo inteiro e uma chave estrangeira para a tabela referenciada. O método *timestamps* cria duas colunas do tipo *DateTime*: *created_at* (data de criação) e *updated_at* (data de atualização).

Também é possível observar que são gerados dois métodos para cada migração: *up* e *down*. O método *up*, declarado nas linhas 2 a 13, é executado para avançar o estado

do banco de dados para a nova versão e o método *down*, declarado nas linhas 15 a 17, é executado para retornar o banco de dados ao estado que se encontrava antes da execução do método *up*.

```
1  class CreateClassEnrollments < ActiveRecord::Migration
2    def self.up
3      create_table :class_enrollments do |t|
4        t.text :obs
5        t.integer :grade
6        t.boolean :attendance
7        t.string :situation
8        t.references :course_class
9        t.references :enrollment
10
11       t.timestamps
12     end
13   end
14
15   def self.down
16     drop_table :class_enrollments
17   end
18 end
```

Figura 4.5: Migração utilizada para a tabela de Inscrições.

Para a construção e execução de tarefas, o framework *Ruby on Rails* utiliza uma biblioteca chamada *Rake*[2].

Existe uma tabela chamada *schema_migrations*, que armazena um código relativo a cada migração já executada. Para atualização do banco de dados, a tarefa *rake db:migrate* é utilizada, que executa o método *up* de todos os arquivos de migração que não estão cadastrados na tabela *schema_migrations*. Para reversão do banco de dados, é utilizada a tarefa *rake db:rollback*, que executa o método *down* da última migração inserida na tabela *schema_migrations*. Após a reversão do banco de dados, a tupla relativa à migração executada é removida da tabela *schema_migrations*. No caso de utilizarmos o banco de dados *MySQL* (AXMARK; LARSSON; WIDENIUS, 1995), a consulta em SQL gerada por esta migração é exposta na Figura 4.6.

Para buscas simples de registros no banco de dados, não há a necessidade de escrever a consulta em SQL. É possível utilizar os comandos *where* e *joins*, presentes

[2] https://github.com/jimweirich/rake

no *Active Record*. Por exemplo, se desejarmos encontrar todos os registros de Inscrições, buscando pelo número de matrícula de um aluno, seria executado o comando apresentado na Figura 4.7. No caso do banco de dados *MySQL* , este comando iria gerar a consulta em SQL apresentada na Figura 4.8.

```
==  CreateClassEnrollments: migrating ================================
-- create_table(:class_enrollments)
SHOW CREATE TABLE `class enrollments`                              `
CREATE TABLE `class enrollments` (`id` int(11) DEFAULT NULL
auto_increment PRIMARY KEY, `obs` text, `grade` int(11), `attendance`
tinyint(1), `situation` varchar(255), `course_class_id` int(11), `
enrollment_id` int(11), `created_at` datetime NOT NULL, `updated_at`
datetime NOT NULL, CONSTRAINT fk_class_enrollments_course_class_id FOREIGN
KEY (`course_class_id`) REFERENCES `course_classes` (`id`), CONSTRAINT
fk_class_enrollments_enrollment_id FOREIGN KEY (`enrollment_id`)
REFERENCES `enrollments` (`id`)) ENGINE=InnoDB
CREATE INDEX `fk__class_enrollments_course_class_id` ON
`class_enrollments` (`course_class_id`)
CREATE INDEX `fk__class_enrollments_enrollment_id` ON `class_enrollments`
(`enrollment_id`)

==  CreateClassEnrollments: migrated (0.7607s) =======================
INSERT INTO `schema_migrations` (`version`) VALUES ('20130404150626')
```

Figura 4.6: SQL gerado pela migração da entidade Inscrição

```
ClassEnrollment.joins(:enrollment).
  where({:enrollments => {
        :enrollment_number => "M001"}})
```

Figura 4.7: Exemplo de consulta

```
SELECT `class_enrollments`.* FROM
`class_enrollments` INNER JOIN `enrollments` ON `enrollments
`.`id` = `class_enrollments`.`enrollment_id` WHERE `enrollments
`.`enrollment_number` = 'M001'
```

Figura 4.8: Exemplo de SQL gerado em consulta pelo ORM

O retorno desse comando é uma coleção de objetos da classe Inscrição, que podem ser utilizados, por exemplo, para geração do histórico do aluno.

Em casos nos quais as consultas não são tão simples, podemos escrever toda ou boa parte da consulta em SQL. O exemplo exibido na Figura 4.9, retorna o número de professores que orientam uma determinada matrícula:

```
enrollment.advisements.joins(:professor).
where("professors.id in
       (SELECT advisement_authorizations.professor_id
  from advisement_authorizations)").
count
```

Figura 4.9: Exemplo de consulta ORM com SQL

É importante observar que as migrações, assim como os outros elementos referentes ao ORM, são independentes do banco de dados que é utilizado na aplicação. Para a conversão dos comandos em SQL são utilizadas bibliotecas adaptadoras do banco de dados desejado. No SAPOS, utilizamos a biblioteca Mysql 2[3] para realizar a adaptação para o banco de dados *MySQL*.

4.3.3 BIBLIOTECAS AUXILIARES

Foram utilizadas diversas bibliotecas auxiliares, que no caso da linguagem *Ruby* são chamadas de "*gems*". Nesta seção, são apresentadas duas *gems* que já estavam sendo utilizadas na primeira versão do SAPOS e 5 novas *gems* adicionadas no SAPOS 2. Algumas *gems* que já eram utilizadas na primeira versão do SAPOS são *Active Scaffold* [4] e *Prawn*[5].

Active Scaffold

Fornece uma interface para a criação de CRUDs (Create, Read, Update, and Destroy ou Criação, Leitura, Atualização e Destruição), incluindo funcionalidades como buscas, paginação e controle de layout. Com a execução de apenas um comando, é possível gerar todos os elementos necessários para a inclusão de uma nova entidade no domínio da aplicação.

Na execução do comando no exemplo apresentado na Figura 4.10, são gerados: (i) o arquivo com esqueleto do modelo, que contém as regras de negócio da entidade; (ii) o esqueleto do controlador, que por padrão já possui todas as ações do CRUD; (iii) o esqueleto do arquivo *helper*, que serve para personalizar as telas de criação, remoção, edição e visualização da entidade; (iv) o arquivo de migração, para criação da tabela no banco de dados da aplicação; (v) o esqueleto do arquivo de fábricas, que é explicado

[3] https://github.com/brianmario/mysql2
[4] https://github.com/activescaffold/active_scaffold
[5] https://github.com/prawnpdf/prawn

posteriormente; (vi) o esqueleto dos testes automatizados, (vii) e o cadastro das ações HTTP, chamadas rotas, necessárias para as ações do CRUD.

```
$ rails generate active_scaffold Professor name:string cpf:string birthdate:date
```

Figura 4.10: Comando de geração de entidade pelo Active Scaffold

As visões são tratadas de forma genérica, porém personalizáveis, portanto não é necessário gerar arquivos para aquelas que não fogem da organização padrão do CRUD.

Prawn

Oferece a geração de arquivos do tipo PDF, facilitando a inserção de imagens, tabelas e textos dinâmicos.

As *gems* adicionadas no SAPOS 2 são Rails ERD[6], Factory Girl Rails[7], Rspec Rails[8], Rcov[9] e Schema Plus[10].

Rails-ERD

A partir do comando *rake erd*, esta *gem* faz o mapeamento do código implementado nos modelos para um diagrama de classes UML no formato PDF. Ela ajuda na documentação do projeto, já que pode-se extrair o estado atual das entidades do sistema para um diagrama UML com um simples comando. O Apêndice A exibe o diagrama do SAPOS gerado a partir do Rails-ERD.

Factory Girl Rails

É uma biblioteca que dá auxilio aos testes automatizados. Com ela, é possível construir objetos válidos de qualquer classe, a partir das chamadas “fábricas”. Usando como exemplo a inscrição de alunos em turmas, seria possível criar uma fábrica chamada “inscrição de aluno aprovado”. Esta fábrica retornaria um objeto da classe Inscrição com a situação “Aprovado”, nota entre 6 e 10 e campo “Reprovado por falta”

[6] https://github.com/voormedia/rails-erd
[7] https://github.com/thoughtbot/factory_girl_rails
[8] https://github.com/rspec/rspec-rails
[9] https://github.com/relevance/rcov
[10] https://github.com/lomba/schema_plus

desmarcado, além de ter todos os outros campos preenchidos de forma que o objeto fosse válido e pronto para ser salvo no banco de dados.

Rspec Rails

É um *framework* de testes automatizados que permite testar qualquer elemento do sistema (*e.g.* modelos, controladores, visões) no ambiente *Ruby*, mesmo que o sistema não utilize o *framework Ruby on Rails*. Caso o sistema utilize o *Ruby on Rails*, é possível testar modelos, controladores, visões e até mesmo requisições e rotas do sistema (DAVID CHELIMSKY et al. 2010).

Rcov

Calcula a porção de código que possui testes automatizados e fornece um relatório em formato HTML que pode ser visualizado em qualquer navegador de internet. Esta *gem* funciona integrada com quaisquer *frameworks* de testes que o sistema esteja utilizando. É possível configurá-la para que apenas os arquivos mais relevantes sejam levados em consideração na geração do relatório.

Schema Plus

Esta *gem* é uma extensão do módulo *ActiveRecord* do *Rails*. Ela oferece recursos avançados para definição de tabelas no banco de dados, como a criação de chaves estrangeiras, índices e *views*. Além disso, é configurável para que todas as migrações gerem automaticamente chaves estrangeiras e índices para os campos mais comuns.

4.4 FERRAMENTAS

Foram utilizadas diversas ferramentas para auxiliar no desenvolvimento do SAPOS 2. Nessa seção, são apresentadas as ferramentas utilizadas para modelagem, controle de modificações e versionamento.

4.4.1 MODELAGEM

Um modelo prescritivo é utilizado para guiar o desenvolvimento do software, determinando como o sistema deve ser construído. Um modelo descritivo é utilizado quando se deseja saber como o sistema está de fato construído.

Para a obtenção do modelo descritivo, é utilizada a *gem Rails ERD*, vista anteriormente. Para a construção do modelo prescritivo, utilizou-se a ferramenta *Astah Community* (ZANIBONI, 1996). Esta ferramenta permite criar diversos diagramas UML, tais como diagrama de classes, casos de uso, atividades, entre outros. É possível exportar os diagramas como imagem para que pessoas que não possuem o programa instalado em seus computadores possam ver a modelagem gerada. A Figura 4.11 apresenta um diagrama de classes gerado pelo *Astah*.

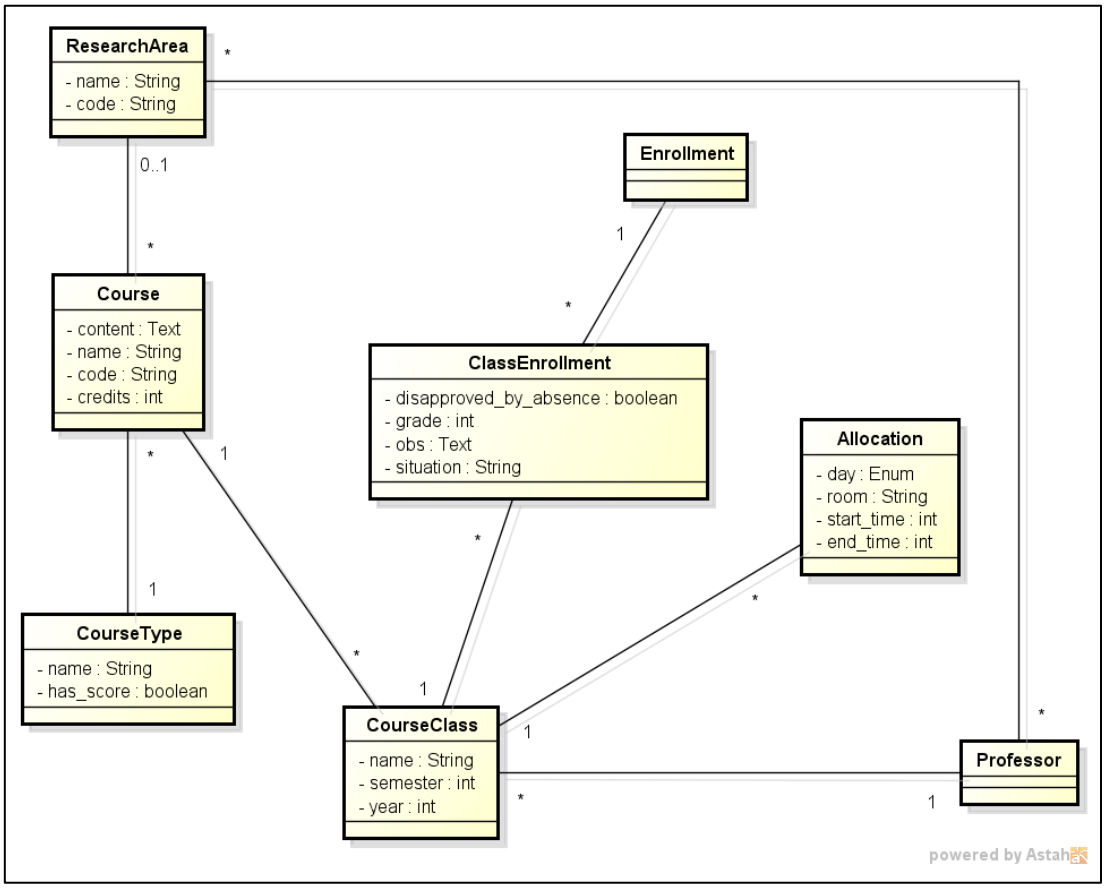


Figura 4.11 Diagrama de classes exportado pelo Astah

4.4.2 VERSIONAMENTO

Para o controle de versões e gerenciamento de código, foi utilizado o *Git* (TORVALDS, 2005), hospedado no sistema *Github* (WANSTRATH; HYETT; PRESTON-WERNER, 2008). O *Github* é um sistema *Web* onde é possível consultar dados do versionamento com uma interface mais amigável do que execução de comandos em um terminal. Nessa interface, é possível navegar pelos diversos ramos (do inglês *branches*) e etiquetas (do

inglês *tags*) geradas no versionamento, consultar o histórico de um arquivo, consultar o histórico de *commits* e diversas outras funcionalidades. A Figura 4.12 mostra a tela principal do repositório do SAPOS no *Github*.

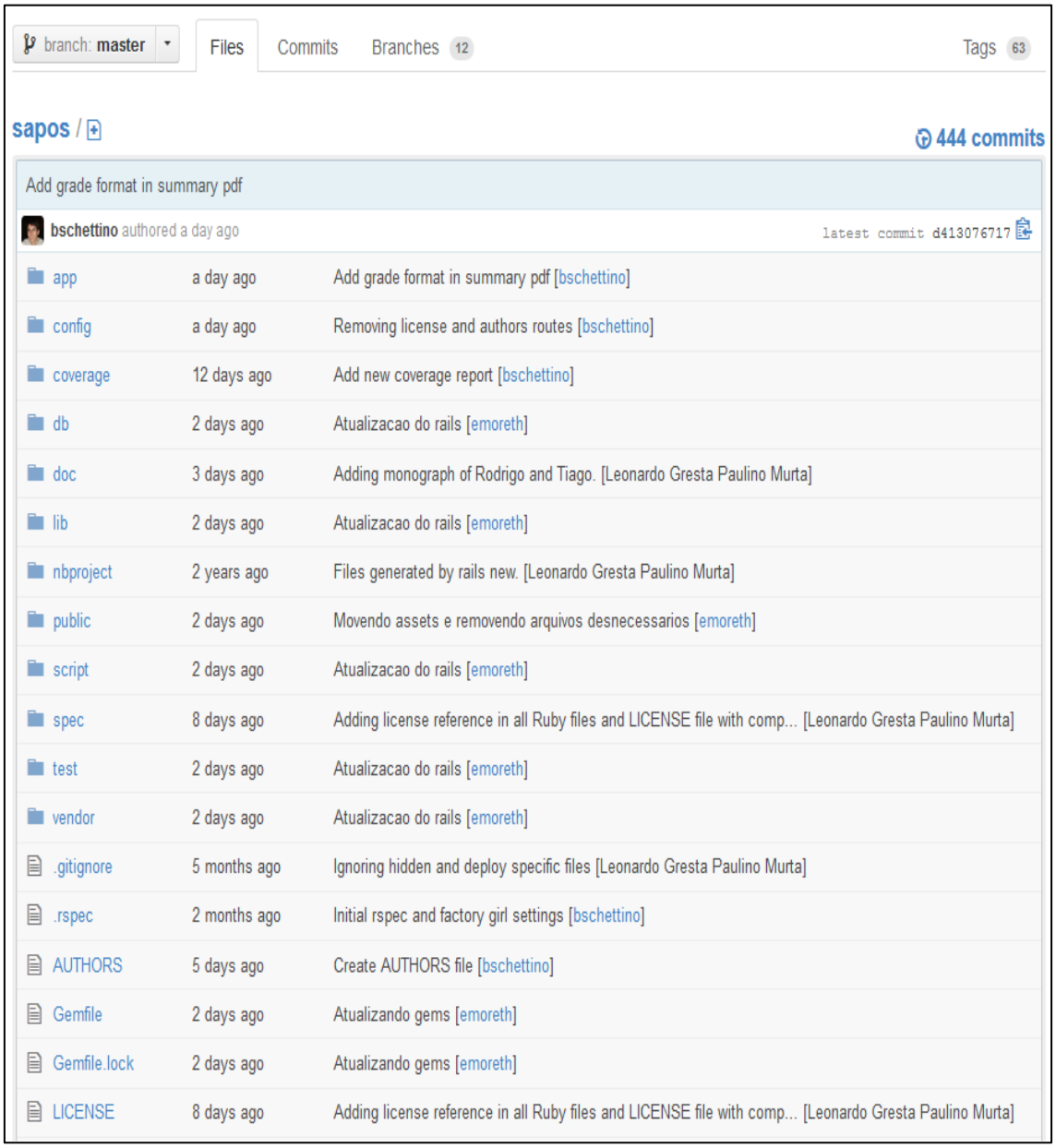


Figura 4.12: Tela principal do Github

Como a primeira versão do SAPOS foi produto de um trabalho de conclusão de curso feito em dupla, o versionamento foi fundamental. O trabalho de guardar cópias versionadas, que já seria inviável sem o uso de uma ferramenta, seria ainda mais difícil sabendo que os desenvolvedores deixariam o projeto ao fim do trabalho. Foram convocados dois alunos da Graduação da UFF, Everton Moreth e Bruno Schettino, para o desenvolvimento dessa segunda versão do SAPOS.

Foram utilizadas as estratégias de ramificação:

- Após a entrada dos novos membros, foi criado o ramo corretivo 1.5.x. Isso foi necessário porque os membros antigos precisavam corrigir *bugs* e fazer ajustes sem serem afetados por commits de novas funcionalidades.
- Foram criados ramos para o desenvolvimento de cada nova funcionalidade do sistema.
- As correções foram feitas em ramos corretivos.
- A integração de novas funcionalidades e correções foi feita no ramo *master*.

A Figura 4.13 apresenta exemplos de ramificações do sistema. Para a numeração das versões e nomeação das etiquetas, foi utilizada a forma M.m.b[11], onde M representa uma grande versão, m representa uma versão menor, na qual foi criada apenas uma nova funcionalidade e b representa correção de defeitos. Esse controle de etiquetas é feito da seguinte forma:

- Para cada nova funcionalidade lançada, o segundo dígito é incrementado e o terceiro dígito é zerado.
- Para cada nova correção ou ajuste, o terceiro dígito é incrementado.

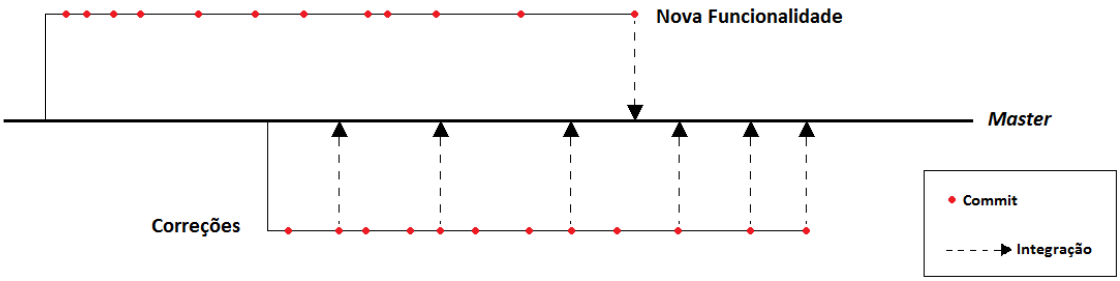


Figura 4.13: Exemplo de ramificações do sistema

4.4.3 CONTROLE DE MODIFICAÇÕES

O *Redmine* é uma ferramenta auxiliar para o controle de modificações. Trata-se de um sistema *Web* no qual podem ser cadastrados requisitos, tarefas e notificações de problemas. A **Erro! Fonte de referência não encontrada.** apresenta uma tela do sistema *Redmine*, na qual é possível ver as tarefas, requisitos e problemas notificados

[11] http://semver.org

com os seus respectivos responsáveis e situações. Também é possível realizar buscas nessa tela e assim listar apenas as informações mais relevantes.

O *Redmine* guarda uma estimativa de horas e data de previsão de início para cada tarefa e requisito cadastrado. Dessa maneira, o resultado da atividade de planejamento também pode ser mantido no sistema.

Figura 4.14: Interface do Sistema Redmine

4.5 TESTES AUTOMATIZADOS

Testes automatizados são importantes para um sistema, pois fornecem a segurança de que, após uma mudança em um componente do sistema, os demais componentes continuam funcionando corretamente. Exemplos de testes automatizados implementados no sistema podem ser vistos na Figura 4.15.

```
describe "enrollment" do
  context "should be valid when" do
    it "enrollment is not null" do
      accomplishment.enrollment = Enrollment.new
      accomplishment.should have(0).errors_on :enrollment
    end
  end
  context "should have error blank when" do
    it "enrollment is null" do
      accomplishment.enrollment = nil
      accomplishment.should have_error(:blank).on :enrollment
    end
  end
end
```

Figura 4.15: Testes automatizados

Foram implementados 252 diferentes testes automatizados, que totalizaram 83,51% de cobertura do total de linhas de código das regras de negócio do sistema. O

relatório é apresentado na Figura 4.16. Esse relatório é gerado de forma automática, utilizando a *gem* Rcov, mencionada anteriormente. Com ele, podemos ver a relação de linhas testadas por total de linhas do sistema e a relação de linhas testadas por linhas de código do sistema. A diferença dessas duas métricas é que a segunda inclui apenas linhas de código efetivamente e a primeira inclui também as linhas em branco e linhas com conteúdo que não são utilizados como código fonte do sistema, como comentários, por exemplo.

Cada arquivo do sistema tem um detalhamento da cobertura de testes. Um exemplo é apresentado na Figura 4.17. As linhas de código que foram executadas durante algum teste são exibidas na cor verde. As linhas na cor cinza representam aquelas que não têm código, ou seja, aquelas que contêm apenas espaços em branco ou comentários. Por fim, as linhas de código que não foram executadas por nenhum dos casos de teste são mostradas na cor vermelha.

NAME	TOTAL LINES	LINES OF CODE	TOTAL COVERAGE	CODE COVERAGE
app/models/dismissal.rb	14	10	100.00%	100.00%
app/models/dismissal_reason.rb	3	3	100.00%	100.00%
app/models/enrollment.rb	78	63	100.00%	100.00%
app/models/enrollment_status.rb	3	3	100.00%	100.00%
app/models/institution.rb	5	4	100.00%	100.00%
app/models/level.rb	4	4	100.00%	100.00%
app/models/major.rb	13	11	100.00%	100.00%
app/models/phase.rb	8	7	100.00%	100.00%
app/models/phase_duration.rb	19	15	100.00%	100.00%
app/models/professor.rb	76	60	47.37%	41.67%
app/models/professor_research_area.rb	4	4	100.00%	100.00%
app/models/research_area.rb	11	9	100.00%	100.00%
app/models/role.rb	9	8	100.00%	100.00%
app/models/scholarship.rb	20	15	100.00%	100.00%
app/models/scholarship_duration.rb	113	84	94.69%	92.86%
app/models/scholarship_type.rb	3	3	100.00%	100.00%
app/models/sponsor.rb	4	3	100.00%	100.00%
app/models/state.rb	8	7	100.00%	100.00%
app/models/student.rb	16	10	100.00%	100.00%
app/models/user.rb	42	34	42.86%	29.41%
app/models/year_semester.rb	137	120	100.00%	100.00%
TOTAL	**940**	**738**	**83.51%**	**84.28%**

Figura 4.16: Relatório de cobertura de código

```
1 class Advisement < ActiveRecord::Base
2   belongs_to :professor
3   belongs_to :enrollment
4
5   validates :professor, :presence => true
6   validates :enrollment, :presence => true
7
8   validates :professor_id, :uniqueness => {:scope => :enrollment_id}
9   validates :main_advisor, :uniqueness => {:scope => :enrollment_id}, :if => :main_advisor
10   validates :main_advisor, :presence => true, :unless => :enrollment_has_advisors
11
12
13   def to_label
14     "#{enrollment.enrollment_number} - #{professor.name}"
15   end
16
17   #defines if an certain advisement is active (An active advisement is an advisement which the student doesn't have a dismissal reason
18   def active
19     return false if enrollment.nil?
20     dismissals = Dismissal.where(:enrollment_id => enrollment.id)
21     return dismissals.empty?
22   end
23
24   def active_order
25     return active.to_s
26   end
27
```

Figura 4.17: Detalhamento de cobertura de código do modelo de Orientação

4.6 DISCUSSÃO

Foram apresentados nesse capítulo as técnicas e ferramentas utilizadas no desenvolvimento do SAPOS 2. O resultado do uso dessas técnicas e ferramentas é uma melhor organização do código fonte e do processo de desenvolvimento do sistema, visando que o *software* tenha a capacidade de crescer e estar de acordo com as necessidades da coordenação de Pós-Graduação em Computação da UFF.

CAPÍTULO 5 – CONCLUSÃO

O resultado deste trabalho é a ampliação do sistema que apoia na gestão de dados da Pós-Graduação em Computação da UFF. Com esta ampliação é possível atender uma maior porção das necessidades de gerência dos dados manipulados pela coordenação e secretaria, diminuindo a quantidade de dados armazenados em planilhas do *Microsoft Excel* e bancos de dados do *Microsoft Access*. Além disso, com a criação de testes automatizados, o sistema ganha mais confiabilidade e segurança para se expandir ainda mais.

O SAPOS está em fase de registro no INPI e se tornou um projeto de código aberto, utilizando a licença MIT[12]. Essa licença é permissiva e exige somente a manutenção de autoria, deixando claro que o SAPOS foi desenvolvido pela Universidade Federal Fluminense. O código do SAPOS pode ser obtido em https://github.com/gems-uff/sapos.

Fazendo uma análise da situação atual do sistema, foram identificados alguns pontos que podem ser melhorados em trabalhos futuros. É possível citar o relatório no formato PDF de histórico de alunos. Atualmente, o relatório conta somente com dados básicos das disciplinas cursadas pelo aluno. Para a geração de um histórico mais completo, são necessários alguns dados que ainda não estão sendo armazenados no sistema. Entre esses dados estão o título da tese ou dissertação do aluno e o conceito do programa de Pós-Graduação na CAPES, que dá uma noção da qualidade do curso para quem não o conhece.

A versão atual do sistema permite identificar todos os alunos que terão alguma etapa atrasada em uma determinada data. Atualmente, caso seja preciso notificar que um aluno estará com uma etapa atrasada em uma data próxima, é necessário que isto seja previamente consultado no sistema. É inviável que a secretaria realize essa busca no sistema recorrentemente e notifique a cada um dos alunos, aos seus respectivos orientadores e ao coordenador do curso. É desejável que o sistema realize essa busca automaticamente e envie um e-mail para o aluno, notificando-o de que ele deve pedir uma prorrogação, ou estará com a etapa atrasada em uma data próxima. Seria desejável, também, que ao fim dos envios dos e-mails para cada aluno, o sistema automaticamente enviasse outro e-mail para o orientador, com a listagem dos seus orientandos nessas

[12] http://opensource.org/licenses/MIT

condições, e para o coordenador do curso, com a relação completa dos alunos que estão nessas condições.

Como dito anteriormente, o SisPós é o sistema oficial da UFF para a gestão de Pós-Graduação. O SisPós e o SAPOS possuem o cadastro de alunos e matrículas. Devido a este fato, é necessária a comunicação entre o SAPOS e o SisPós, para que a consistência das bases de dados dos dois sistema seja mantida. Esta comunicação poderia ser feita através de *Web Services* e da interface do SisPós para a importação e exportação de dados via planilhas. Seria necessário, então, desenvolver no SAPOS: (i) a exportação de dados para planilhas compatíveis àquelas do SisPós; (ii) uma interface de importação de planilhas exportadas pelo SisPós; e (iii) o desenvolvimento de chamadas *Webservices* para a comunicação necessária com o SisPós.

Além disso, a versão atual do SAPOS contém apenas testes automatizados das regras de negócio do sistema. É interessante que sejam criados testes automatizados para a interface e controle de acesso do sistema, fazendo com que ele possa ser expandido com ainda mais segurança e confiabilidade.

REFERÊNCIAS BIBLIOGRÁFICAS

AXMARK, David, Allan LARSSON, e Michael WIDENIUS. 1995. *MySQL*. http://www.mysql.com/.

BARZEL, Ronen, e Michal LOMNICKI. 2013. "SchemaPlus". Acessado junho 23. https://github.com/lomba/schema_plus.

BEDRA, Aaron, Chad HUMPHRIES, e Jay MCGAFFIGAN. 2013. "RCov - Code Coverage for Ruby". Acessado junho 19. https://github.com/relevance/rcov.

BROWN, Gregory, Brad EDIGER, Daniel NELSON, Jonathan GREENBERG, e James HEALY. 2013. "Prawn". Acessado junho 18. https://github.com/prawnpdf/prawn.

CHELIMSKY, David. 2013. "Rspec-rails". Acessado junho 19. https://github.com/rspec/rspec-rails.

CHELIMSKY, DAVID, DAVE ASTELS, ZACH DENNIS, ASLAK HELLESOY, BRYAN HELMKAMP, e DAN NORTH. 2010. *The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber, and Friends.* Pragmatic Bookshelf.

Ferreira, Rodrigo, e Tiago Amaro. 2013. "SAPOS - Sistema de Apoio à Pós-Graduação". Monografia de Conclusão de Graduação, Niterói, RJ - Brasil: Ciência da Computação, IC-UFF.

FERRIS, Joe. 2013. "Factory Girl Rails". Acessado junho 23. https://github.com/thoughtbot/factory_girl_rails.

Hunt, Andrew, e David Thomas. 1999. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley.

"Interação entre as camadas no padrão MVC". 2013. Acessado julho 15. http://www.htmlgoodies.com/img/2010/11/mvc.png.

KRASNER, Glenn E., e Stephen T. POPE. 1988. *Stephen. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*. http://www.create.ucsb.edu/~stp/PostScript/mvc.pdf.

LANG, Jean-Philippe. 2006. *Redmine*. http://www.redmine.org/.

LOPEZ, Brian. 2013. "Mysql2". Acessado julho 15. https://github.com/brianmario/mysql2.

Pressman, Roger. 2010. *Software Engineering: A Practitioner's Approach.* 7th ed.

PRESTON-WERNER, Tom. 2013. “Semantic Versioning 2.0.0”. Acessado julho 16. http://semver.org/.

RUTHERFORD, Scott. 2013. “ActiveScaffold: A Ruby on Rails plugin from the makers of AjaxScaffold”. *GitHub*. Acessado junho 18. https://github.com/activescaffold/active_scaffold.

“The MIT License (MIT) | Open Source Initiative”. 1988. http://opensource.org/licenses/MIT.

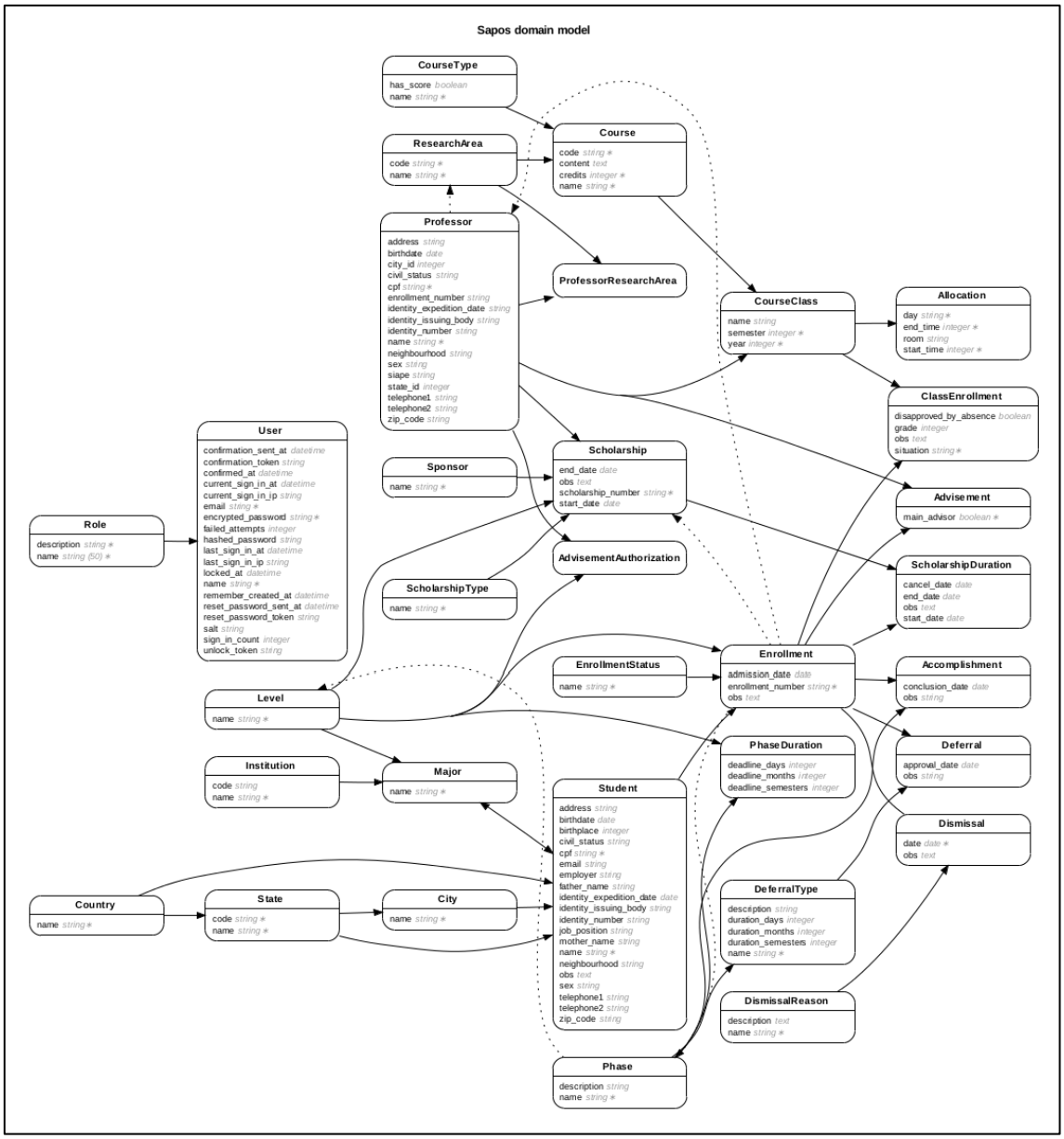
THOMAS, Dave, e David Heinemeier HANSSON. 2011. *Agile Web Development with Rails*. 4th ed. Pragmatic Bookshelf.

TIMMERMANS, Rolf. 2013. “Rails ERD. Generate Entity-Relationship Diagrams for Rails applications”. *GitHub*. Acessado julho 15. https://github.com/voormedia/rails-erd.

TORVALDS, Linus. 2005. *Git*. http://git-scm.com/.

WANSTRATH, Chris, PJ HYETT, e Tom PRESTON-WERNER. 2008. *GitHub*. https://github.com/.

WEIRICH, Jim. 2004. “Rake - Ruby Make”. *GitHub*. https://github.com/jimweirich/rake.

ZANIBONI, Massimo. 1996. *Astah Community*. Change Vision. http://astah.net/editions/community.

APÊNDICE A – MAPEAMENTO DAS CLASSES DO SISTEMA GERADO PELO RAILS ERD