



César Henrique Bernabé

# **Evolução da Arquitetura do Zanshin – um framework para análise de requisitos em tempo de execução**

Vitória, ES

2017

César Henrique Bernabé

## **Evolução da Arquitetura do Zanshin – um framework para análise de requisitos em tempo de execução**

Monografia apresentada ao Curso de Ciência da Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Departamento de Informática

Orientador: Prof. Dr. Vítor E. Silva Souza

Vitória, ES

2017

---

César Henrique Bernabé

Evolução da Arquitetura do Zanshin – um framework para análise de requisitos em tempo de execução/ César Henrique Bernabé. – Vitória, ES, 2017-  
54 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Vítor E. Silva Souza

Monografia (PG) – Universidade Federal do Espírito Santo – UFES  
Centro Tecnológico  
Departamento de Informática, 2017.

1. Engenharia de Requisitos Orientada a Objetivos. 2. Zanshin. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. Evolução da Arquitetura do Zanshin – um framework para análise de requisitos em tempo de execução

CDU 02:141:005.7

---

César Henrique Bernabé

## **Evolução da Arquitetura do Zanshin – um framework para análise de requisitos em tempo de execução**

Monografia apresentada ao Curso de Ciência da Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Vitória, ES  
2017

# Agradecimentos

AGRADECIMENTOS

*“Sou contra a violência, pois quando parece fazer o bem,  
o bem é apenas temporário.  
O mal que causa é permanente.  
(Gandhi)*

*Alguns de nós pensam que aguentar nos faz fortes.  
Mas às vezes, é desistir.  
(Herman Hesse)*

# Resumo

Com o avanço da tecnologia, sistemas de software são executados em ambientes cada vez mais dinâmicos, sendo assim obrigados a lidar com requisitos cada vez mais complexos. Nesse universo tecnológico, onde sistemas precisam reagir a diversos tipos de condições destaca-se o uso de sistemas que modificam seu comportamento e performance em tempo de execução para satisfazer requisitos mesmo em caso de falha, podendo se replanejar e reconfigurar a medida que novas exigências são encontradas e precisam ser atendidas. *Zanshin* é uma abordagem baseada em Engenharia de Requisitos Orientada a Objetivos (Goal-Oriented Requirements Engineering ou simplesmente GORE) criada para apoiar o desenvolvimento de sistemas adaptativos. Baseado no fato de que todo sistema possui, ainda que implícito, um ciclo de retroalimentação, o framework utiliza dessa premissa para monitorar e avaliar o comportamento do sistema alvo, enviando assim instruções de adaptação para o mesmo. Para auxiliar o processo de modelagem dos sistemas adaptativos são definidas duas novas classes de Requisitos: - Requisitos de Percepção (*Awareness Requirements* ou *s*), que especificam os indicadores de convergência que o sistema deve atingir para que determinado objetivo seja cumprido; - Requisitos de Evolução (*Evolution Requirements* ou *EvoReqs*), que definem as medidas a serem seguidas pelo processo de adaptação, ou seja, as instruções que serão enviadas ao sistema alvo mediante falha de um objetivo; Este trabalho discute uma nova proposta para o metamodelo operacional do *Zanshin*, que foi reformulado buscando compreender as restrições de *GORE* de uma forma mais precisa do que na adotada anteriormente. Além disso, para apoiar o processo de modelagem dos requisitos dentro desta abordagem, é apresentada a ferramenta CASE Unagi, que implementa um editor gráfico para modelos de sistemas adaptativos e oferece um sistema de integração com o *Zanshin*, permitindo assim que modelos criados sejam importados diretamente para a plataforma de apoio a sistemas adaptativos.

**Palavras-chaves:** Engenharia de Requisitos, GORE, Zanshin, Unagi...

# Lista de ilustrações

Figura 1 – Exemplo de modelos de objetivos (SOUZA, 2012) . . . . .	18
Figura 2 – Exemplo de modelos de objetivos de um sistema de despacho de ambulâncias (SOUZA, 2012) . . . . .	21
Figura 3 – SAE - CORE - Diagrama de Casos de Uso. . . . .	25
Figura 4 – SAE - CORE - Diagrama de Casos de Uso. . . . .	25
Figura 5 – SAE - PUBLIC - Diagrama de Casos de Uso. . . . .	27
Figura 6 – SAE - CORE - Diagrama de Classes. . . . .	28
Figura 7 – SAE - PUBLIC - Diagrama de Classes. . . . .	29
Figura 8 – Pacotes que formam a arquitetura do SAE. . . . .	30
Figura 9 – SAE - Arquitetura - Sistema (??). . . . .	31
Figura 10 – SAE - Implementação - Pacotes. . . . .	31
Figura 11 – SAE - Implementação - Páginas Web . . . . .	32
Figura 12 – FrameWeb - sae.core - Modelo Domínio. . . . .	36
Figura 13 – FrameWeb - Public - Modelo Domínio. . . . .	37
Figura 14 – FrameWeb - <i>nemo-utils</i> - Modelo Navegação (??). . . . .	38
Figura 15 – FrameWeb - Consultar Depoimentos - Modelo Navegação. . . . .	38
Figura 16 – FrameWeb - <i>nemo-utils</i> - Modelo Aplicação (??). . . . .	39
Figura 17 – FrameWeb - Public - Modelo Aplicação. . . . .	39
Figura 18 – FrameWeb - Core - Modelo Aplicação. . . . .	40
Figura 19 – FrameWeb - <i>nemo-utils</i> - Modelo Persistência (??). . . . .	41
Figura 20 – FrameWeb - Public - Modelo Persistência. . . . .	41
Figura 21 – FrameWeb - Core - Modelo Persistência. . . . .	42
Figura 22 – SAE - Tela Login. . . . .	42
Figura 23 – SAE - Erro Login. . . . .	43
Figura 24 – SAE - Tela inicial Egresso. . . . .	43
Figura 25 – SAE - Tela Mobile de Administrador. . . . .	44
Figura 26 – SAE - Lista de Curso. . . . .	44
Figura 27 – SAE - Tela cadastro de Curso. . . . .	45
Figura 28 – SAE - Tela exclusão de Curso. . . . .	45
Figura 29 – SAE - Telas mobile. . . . .	46
Figura 30 – SAE - Telas mobile Gráfico. . . . .	46
Figura 31 – SAE - Tela Consulta Faixa Salarial. . . . .	47
Figura 32 – SAE - Tela Depoimentos á serem analisados. . . . .	47
Figura 33 – SAE - Tela Análise de Depoimentos. . . . .	48
Figura 34 – SAE - Tela Consulta Depoimentos. . . . .	48
Figura 35 – FrameWeb Modelo de Aplicação Sugerido . . . . .	51



# Lista de tabelas

Tabela 1 – Atores . . . . .	24
-----------------------------	----

# Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
HTML	HyperText Markup Language
Java EE	Java Platform, Enterprise Edition
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	eXtensible Markup Language
GORE	Goal Oriented Requirements Engineering
EMF	Eclipse Modeling Framework

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>12</b>
<b>1.1</b>	<b>Objetivos</b>	<b>12</b>
<b>1.2</b>	<b>Metodologia</b>	<b>13</b>
<b>1.3</b>	<b>Organização do Texto</b>	<b>14</b>
<b>2</b>	<b>REFERENCIAL TEÓRICO</b>	<b>16</b>
<b>2.1</b>	<b>Engenharia de Requisitos Orientada a Objetivos</b>	<b>16</b>
2.1.1	Modelos de Objetivos em Tempo de Execução	19
2.1.2	Exemplo de Modelagem de Caso de Uso	20
<b>2.2</b>	<b>Zanshin</b>	<b>21</b>
<b>2.3</b>	<b>Desenvolvimento Orientado a Modelos</b>	<b>21</b>
2.3.1	Sirius	21
<b>3</b>	<b>ESPECIFICAÇÃO DE REQUISITOS</b>	<b>22</b>
<b>3.1</b>	<b>Descrição do Escopo</b>	<b>22</b>
<b>3.2</b>	<b>Diagrama de Casos de Uso</b>	<b>24</b>
3.2.1	Atores	24
3.2.2	Subsistema sae.core	25
3.2.3	Subsistema sae.public	26
<b>3.3</b>	<b>Diagrama de Classes</b>	<b>28</b>
3.3.1	Subsistema sae.core	28
3.3.2	Subsistema sae.public	29
<b>4</b>	<b>PROJETO ARQUITETURAL E IMPLEMENTAÇÃO</b>	<b>30</b>
<b>4.1</b>	<b>Arquitetura do Sistema</b>	<b>30</b>
4.1.1	Camada de Apresentação	31
4.1.2	Camada de Negócio	33
4.1.3	Camada de Acesso a Dados	33
<b>4.2</b>	<b>Framework <i>nemo-utils</i></b>	<b>34</b>
<b>4.3</b>	<b>Modelos FrameWeb</b>	<b>35</b>
4.3.1	Modelo de Domínio	35
4.3.2	Modelo de Navegação	37
4.3.3	Modelo de Aplicação	38
4.3.4	Modelo de Persistência	40
<b>4.4</b>	<b>Apresentação do Sistema</b>	<b>42</b>

<b>5</b>	<b>CONSIDERAÇÕES FINAIS . . . . .</b>	<b>49</b>
<b>5.1</b>	<b>Conclusões . . . . .</b>	<b>49</b>
<b>5.2</b>	<b>Limitações e Perspectivas Futuras . . . . .</b>	<b>50</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>52</b>
	<b>APÊNDICES</b>	<b>54</b>

# 1 Introdução

O avanço da tecnologia nas ultimas décadas permitiu que a complexidade das atividades realizadas por computadores se tornasse cada vez maior, demandando que projetos de sistemas de software passassem a abranger ainda mais os detalhes de domínio do ambiente em que os programas computacionais seriam executados (ANDERSSON et al., 2009; BRUN et al., 2009). Esse fator motivou estudos na área de modelagem e projeto de sistemas, fazendo com que novas pesquisas buscassem abranger os processos de projeto, construção e teste. Entretanto, para garantir a estabilidade dos sistemas, fazia-se uso majoritariamente da intervenção humana (ANDERSSON et al., 2009), o que rapidamente tornou-se inviável à medida que os sistemas cresciam (ANDERSSON et al., 2009) para atender o aumento na demanda de usuários. Assim, a utilização de sistemas adaptativos vem tornando-se a solução mais viável e prática para a atual conjuntura do desenvolvimento de softwares. Além disso, o aumento do número de diferentes dispositivos e situações em que os softwares podem ser executados faz com que esses passem a enfrentar uma grande diversidade de contextos (muitas vezes imprevisíveis) de execução, fundamentando ainda mais a pesquisa na área de softwares adaptativos (KEPHART; CHESS, 2003).

Sistemas adaptativos são dotados da capacidade de tomar decisões para se ajustar e se reconfigurar mediante mudanças de contexto, permitindo assim que os requisitos elicitados continuem a ser atendidos de forma satisfatória (SOUZA; LAPOUCHNIAN; MYLOPOULOS, 2012). Entretanto, poucas soluções desse tipo consideram a modelagem das características adaptativas no sistema desde a fase de levantamento de requisitos de software, como por exemplo o *Zanshin* (SOUZA, 2012), uma abordagem que baseia-se em modelos de requisitos para projetar características adaptativas em sistemas através de novos requisitos chamados Requisitos de Adaptação.

## 1.1 Objetivos

Este trabalho possui dois objetivos principais, a reconstrução do metamodelo operacional do *Zanshin* e a criação de uma ferramenta gráfica usada para modelar sistemas adaptativos através da Engenharia de Requisitos Orientada a Objetivo (*Goal Oriented Requirements Engineering* ou *GORE*). O primeiro refere-se ao metamodelo de objetivos baseado em  $i^*$ , usado pelo framework para verificar se os requisitos do modelo de domínio específico estão sendo atendidos satisfatoriamente. Para ambas as atividades, foram utilizados os conceitos aprendidos ao longo do curso de Ciência da Computação. São objetivos específicos deste projeto:

- Levantar as deficiências dos metamodelo atual do Zanshin, identificando os pontos em que as relações entre os elementos deveriam ser modificadas para representarem mais fielmente as formalidades da Engenharia de Requisitos Orientada a Objetivos, bem como pontos onde as decomposições entre elementos deveria tornar-se restrita, por exemplo.
- Elaborar um metamodelo atualizado que, além de representar mais rigorosamente a hierarquia dos elementos *GORE*, também reflita as necessidades da arquitetura do *Zanshin*, como por exemplo as relações de contenimento de elementos.
- Modificar o código fonte do *framework* para que o mesmo possa utilizar o novo metamodelo desenvolvido e executar o mecanismo de adaptações considerando esse novo metamodelo.
- Desenvolver uma ferramenta que permita ao usuário criar uma representação gráfica do modelo do sistema alvo e implementar, dentro dessa ferramenta, um módulo que permita converter o modelo gráfico representado em arquivos XML que podem ser importados diretamente para o *Zanshin*.
- Apresentar os trabalhos desenvolvidos, juntamente com as perspectivas futuras de otimização de ambos os sistemas apresentados nesse trabalho.

## 1.2 Metodologia

O trabalho realizado compreendeu as seguintes atividades:

1. *Revisão Bibliográfica*: Estudo sobre Engenharia de Requisitos Orientada a Objetivos, Desenvolvimento Orientado a Modelos, de publicações acadêmicas sobre *Zanshin* e sobre desenvolvimento de sistemas adaptativos.
2. *Estudo das Tecnologias*: Por fim, levantamento das tecnologias disponíveis para *Eclipse Modeling Framework* (EMF) que permitam o desenvolvimento de editores gráficos dentro da plataforma Eclipse<sup>TM</sup>. Após o levantamento das tecnologias que deveriam ser utilizadas para o desenvolvimento de editores gráficos iniciou-se estudo das tecnologias EMF, Sirius<sup>TM</sup> e do código fonte do *Zanshin*.
3. *Elaboração do novo Metamodelo*: Nessa etapa o novo metamodelo a ser usado foi elaborado gradativamente a partir das informações obtidas dos documentos estudados e das discussões realizadas em reuniões com grupo de estudos de Engenharia de Requisitos Orientada a Objetivos na Ufes.
4. *Adequação do Zanshin ao novo Metamodelo*: Após finalização do metamodelo, iniciou-

se processo de adequação do *Framework* para que o mesmo pudesse operar adequadamente de acordo com a nova proposta.

5. *Implementação da Ferramenta de Modelagem*: Uma primeira versão da ferramenta foi desenvolvida no contexto de trabalho de Iniciação Científica, entretanto a mesma usava o metamodelo antigo do *Zanshin*. Essa etapa consiste, portanto, no desenvolvimento de ferramenta gráfica que permite a modelagem de sistemas adaptativos seguindo as formalidades do novo metamodelo proposto para o sistema *Zanshin*, bem como a criação de módulo java que permite a exportação do modelo desenvolvido nessa ferramenta para arquivo XML adequado aos padrões do *Framework*.
6. *Redação da Monografia*: Escrita da monografia, etapa obrigatória do processo de elaboração do Projeto de Graduação. Para a escrita desta, foi utilizada a linguagem *LaTeX*<sup>1</sup> utilizando o template *abnTeX*<sup>2</sup> que atende os requisitos das normas da ABNT (Associação Brasileira de Normas Técnicas) para elaboração de documentos técnicos e científicos brasileiros. Para apoiar este processo, foi utilizado o aplicativo *TeXstudio*<sup>3</sup>.

## 1.3 Organização do Texto

Este texto está dividido em quatro partes principais além desta introdução, que seguem:

- **Capítulo 2** – Referencial Teórico: apresenta discussão acerca de *GORE* e *Model Driven Development*, focando na relação desses tópicos com sistemas adaptativos e com o processo de desenvolvimento da ferramenta *Unagi*. Ademais, é discutida a arquitetura do sistema *Zanshin* e suas características.
- **Capítulo 3** – Proposta: nesse capítulo são apresentados os processos e decisões que levaram a elaboração do novo metamodelo do *Zanshin*, bem como as modificações decorrentes dessas modificações na arquitetura da plataforma. Ademais, na seção seguinte é abordado o processo de desenvolvimento da ferramenta *Unagi* e os pormenores da implementação de todos os módulos da mesma.
- **Capítulo 4** – Revisão: é feita validação do metamodelo evoluído do *Zanshin* e da implementação da ferramenta *Unagi*.
- **Capítulo 5** – Considerações Finais: apresenta as conclusões obtidas ao final deste trabalho, tal como as dificuldades encontradas e as perspectivas de trabalhos futuros

<sup>1</sup> LaTeX – <http://www.latex-project.org/>

<sup>2</sup> abnTeX – <http://www.abntex.net.br>

<sup>3</sup> [www.texstudio.org](http://www.texstudio.org)

para esse contexto.



## 2 Referencial Teórico

Este capítulo apresenta os principais conceitos teóricos que fundamentaram a evolução do metamodelo de requisitos do *Zanshin* e do desenvolvimento da ferramenta *Unagi*. A seção 2.1 aborda a Engenharia de Requisitos Orientada a Objetivos, destacando os principais conceitos dessa área que foram utilizados ao longo deste trabalho. A seção ?? apresenta o sistema *Zanshin* e os detalhes do metamodelo original do *framework*. A seção ?? apresenta as principais ferramentas que foram utilizadas durante o desenvolvimento do *Unagi*, como as funcionalidades de Desenvolvimento Orientado a Modelos (*Model Driven Development* ou MDD) do Eclipse<sup>TM</sup>, o plugin Sirius<sup>TM</sup>, dentre outros.

### 2.1 Engenharia de Requisitos Orientada a Objetivos

A Engenharia de Software é uma área da Ciência da Computação voltada ao estudo dos processos, métodos, técnicas, ferramentas e ambientes de suporte ao desenvolvimento de software, apoiando-se principalmente nas práticas e aplicações da área de Gerência de Projetos com o objetivo de promover melhor organização, produtividade e qualidade em todo o processo de desenvolvimento de um software (FALBO, 2014).

Dentro da área de Engenharia de Software, destaca-se uma importante subárea, a área de Engenharia de Requisitos de Software, focada no processo de elicitação de requisitos, considerados fatores determinantes no sucesso do desenvolvimento de um software (FALBO, 2017). Requisitos podem ser entendidos como a definição do que o sistema pode prover, ou também entendidos como o que o sistema é capaz de fazer para atingir um determinado objetivo (PFLEEGER, 2004).

Devido ao fato de requisitos estarem diretamente ligados aos objetivos do sistema, destaca-se também a **Engenharia de Requisitos Orientada a Objetivos**, uma parte subárea de Engenharia de Requisitos. Objetivos são parte importante do processo Engenharia de Requisitos, seu propósito é indicar as principais necessidades que justificam a criação de um determinado sistema, demonstrando os casos em que as funcionalidades do mesmo satisfarão as necessidades elicítadas, além de dizer como o sistema deve ser construído para satisfazê-las (ROSS; SCHOMAN, 1977). Em uma descrição geral e resumida do processo de identificação de objetivos, pode-se dizer que o potencial software é analisado nos ambientes organizacional, operacional e técnico, onde são assim identificados os problemas de contexto e as oportunidades de solução desses problemas. Então, os objetivos são criados com foco na resolução dos problemas e das oportunidades identificadas. Tendo em mãos os objetivos do sistema devidamente refinados, os requisitos do sistema são então elaborados para que esses objetivos sejam devidamente atendidos. Além de apoiar no

processo de modelagem de requisitos, objetivos são usados para apoiar outros propósitos como gerenciamento de conflitos e o processo de verificação (LAMSWEERDE, 2001). De acordo com (LAMSWEERDE, 2001), objetivos podem ser reformulados em diferentes níveis de abstração, dependendo do tipo de necessidade que o sistema alvo deve atender, abrangendo desde interesses referentes a estratégias de negócios até conceitos técnicos de atividades, podendo assim referirem-se a requisitos funcionais e não-funcionais.

A necessidade de uso de objetivos no processo de modelagem de sistemas de software vem se tornando cada vez mais clara a medida que analistas percebem que:

- Objetivos provêm critérios claros de completude dos requisitos do sistema, permitindo também que requisitos desnecessários sejam descartados.
- Objetivos facilitam o processo de entendimento dos requisitos por parte dos *stakeholders*.
- Melhora a legibilidade de documentos de especificação de requisitos, pois permite que engenheiros possam enxergar com mais clareza as alternativas de desenvolvimento dos requisitos do sistema. Além de facilitar o processo de gerenciamento de conflitos.
- Objetivos dirigem parte do processo de elicitação de requisitos, facilitando a identificação de boa parte deles.

Diferentemente dos requisitos, objetivos podem precisar da cooperação entre diferentes tipos de refinamentos para que sejam atendidos de forma suficiente (DARDENNE; LAMSWEERDE; FICKAS, 1993). Em outras palavras, um objetivo diretamente relacionado ao sistema a ser criado torna-se um requisito, enquanto um objetivo sob responsabilidade de um agente do ambiente em que o software será executado torna-se uma Pressuposição de Domínio (ou *Domain Assumptions*) e, nesse caso, são satisfeitos devido a uma regra de negócio (LAMSWEERDE, 2001; LAMSWEERDE; DARIMONT; LETIER, 1998). Objetivos funcionais podem ser classificados como objetivos rígidos (*Hard Goals*) e objetivos fracos (*Soft Goals*), estes não possuem critérios claros de satisfação, entretanto são úteis quando deseja-se comparar os melhores refinamentos ao objetivo estudado, enquanto aqueles são objetivos cujo critério de satisfação pode ser atendido de forma técnica (DARDENNE; LAMSWEERDE; FICKAS, 1993). Para que *Soft Goals* tenham um parâmetro claro de satisfabilidade, são adicionados a eles as *Quality Constraints*, critérios que operacionalizam os ***Soft Goals***. Por exemplo, um *Soft Goal* “Baixo Custo” pode ser refinado na *Quality Constraint* “Custo deve ser menor que mil reais”. Por fim, (JURETA; MYLOPOULOS; FAULKNER, 2008) define outro tipo de refinamento para especificar a atendibilidade de um objetivo: as tarefas ou *Taks*, que são os passos a serem tomados para que um determinado objetivo seja cumprido. Em outras palavras, tarefas são definidas por funcionalidades do sistemas que, se executadas com sucesso, são consideradas satisfeitas

(SOUZA; LAPOUCHNIAN; MYLOPOULOS, 2012).

Objetivos relacionam-se um com o outro através de refinamentos. Segundo (DARDENNE; FICKAS; LAMSWEERDE, 1991; DARDENNE; LAMSWEERDE; FICKAS, 1993), objetivos podem ser refinados usando grafos E/OU (*AND/OR*). O critério de satisfabilidade de objetivos refinados em “E” ou “OU” segue os conceitos da lógica booleana: refinamentos do tipo “E” implicam que para que um objetivo seja considerado satisfeito, todos os sub-objetivos refinados a partir dele devem ser satisfeitos, enquanto refinamentos do tipo “OU” relacionam o objetivo principal com um conjunto de alternativas, ou seja, basta que um de seus refinamentos seja atendido para que ele também seja considerado alcançado. Objetivos são refinados até atingirem um level de granularidade em que são refinados apenas por tarefas que podem ser completado com sucesso por um ator (humano ou outro sistema) (SOUZA et al., 2013). Refinamentos podem acontecer entre Objetivos e outros Objetivos, *Softgoals*, Tarefas e Pressuposições de Domínio.

Em questões de representação gráfica, os modelos de objetivos discutidos nesse texto são grafos ordenados com as exigências das partes interessadas no topo do modelo e abaixo, objetivos (e tarefas) mais refinados. A simbologia utilizada é baseada na sintaxe de  $i^*$  (YU et al., 2011). Um exemplo de modelo de objetivos representando um sistema de despacho de ambulâncias é mostrado na Figura 1.

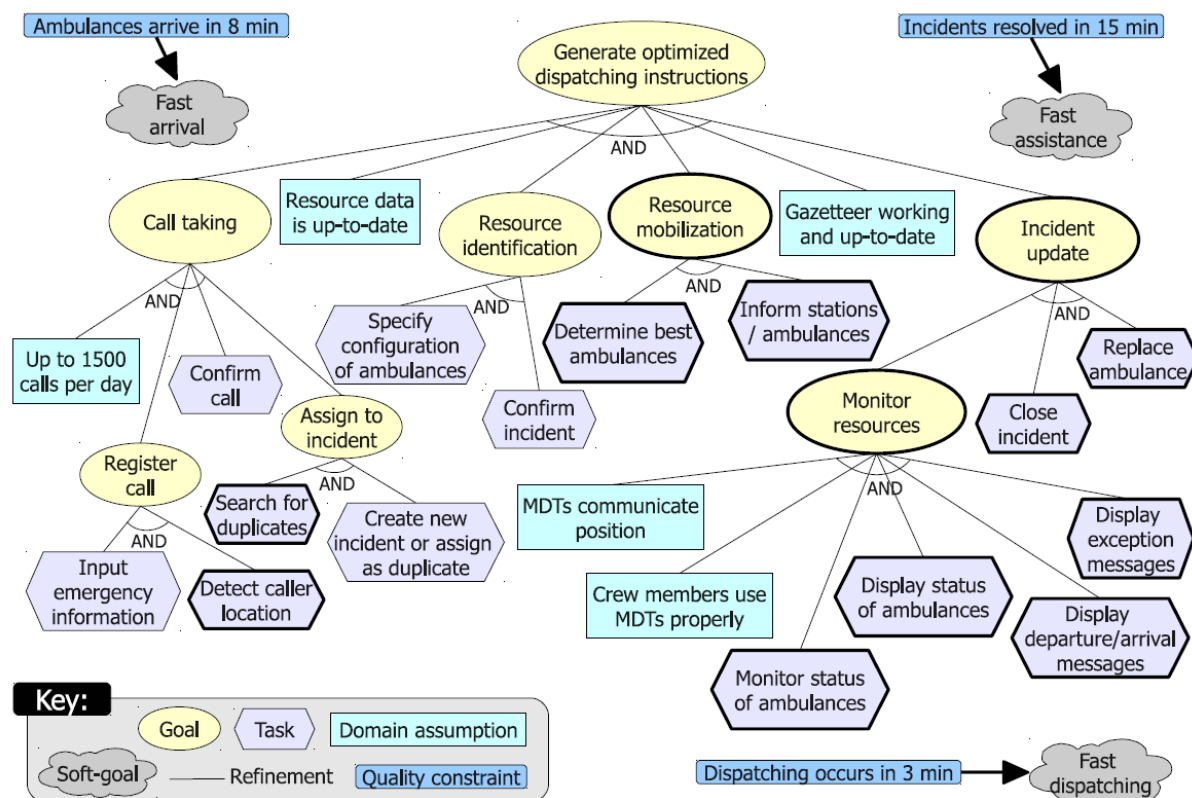


Figura 1 – Exemplo de modelos de objetivos (SOUZA, 2012)

### 2.1.1 Modelos de Objetivos em Tempo de Execução

Muitas vezes os requisitos de um *software* precisam ser modificados durante o ciclo de execução do mesmo. Além disso, durante o processo de especificação as partes interessadas no sistema podem apresentar requisitos condicionais, ou seja, que assumem diferentes configurações dependendo da ocorrência de determinada situação (SOUZA; LAPOUCHNIAN; MYLOPOULOS, 2012). Em outras palavras, há a necessidade de sistemas que possam se automonitorar e, caso necessário, se adaptarem para que seus objetivos continuem sendo satisfeitos (DALPIAZ et al., 2013). Esse tipo de sistema geralmente é composto por duas partes principais: a primeira sendo o sistema em si, que executa uma tarefa para cumprir um objetivo desejado e a segunda sendo um sistema de monitoramento do primeiro, que envia ao primeiro sistema instruções de modificação de suas configurações para que seus objetivos continuem sendo atendidos (SOUZA et al., 2013).

O sistema de monitoramento é construído fundamentado na premissa de que todo sistema possui um ciclo de retroalimentação (*feedback loop*) (BRUN et al., 2009), e assim realizam o processo de adaptação com base nesse ciclo, aplicando controladores de *feedback* que monitoram o comportamento do sistema e injetam estratégias de adaptação (SOUZA et al., 2013). O módulo adaptador do sistema verifica, de acordo com as saídas do sistema alvo, se os objetivos internos a esse estão sendo atendidos e, para isso, necessita importar o modelo de objetivos (SOUZA et al., 2013) enriquecido de elementos que indicam os requisitos a serem observados e as estratégias de adaptação relativas.

Modelos de sistemas adaptativos incluem requisitos autoconscientes, ou seja, requisitos definidos em relação ao sucesso, falha ou qualidade de serviço de outros requisitos (SOUZA et al., 2013). Assim, esses requisitos são considerados “requisitos especiais” já que sua operacionalização está relacionada a mudança de outros requisitos (SOUZA; LAPOUCHNIAN; MYLOPOULOS, 2012). Ademais, o comportamento do sistema é caracterizado por eventos que ocorrem em tempo de execução e que estão diretamente ligados a instâncias de objetivos (DALPIAZ et al., 2013). Assim, é importante observar que essa abordagem é considerada orientada a objetivos já que os requisitos mencionados são derivados do refinamento de objetivos elicitados para o sistema.

Requisitos autoconscientes são divididos em dois tipos principais: Requisitos de Percepção (*Awareness Requirements* ou *AwReqs*) (SOUZA et al., 2013) e Requisitos de Evolução (*Evolution Requirements* ou *EvoReqs*) (SOUZA; LAPOUCHNIAN; MYLOPOULOS, 2012). *AwReqs* são requisitos que referem-se ao estado de outros requisitos em tempo de execução, representando situações onde as partes interessadas desejam que o sistema se adapte (SOUZA; LAPOUCHNIAN; MYLOPOULOS, 2012). Podem se referir a qualquer tipo de elemento, sejam objetivos, *Softgoals*, tarefas e pressuposições de domínio. Além disso, indicam o quão crítico um requisito pode ser ao descrever o grau de

tolerância a falhas do mesmo (SOUZA; LAPOUCHNIAN; MYLOPOULOS, 2012). Antes da execução de um sistema, os requisitos estão em estado “não decidido” (*undecided*), e então pode assumir os estados “Sucesso” (*Succeeded*), “Falha” (*Failed*), e no caso de objetivos e tarefas, “Cancelado” (*Canceled*) (SOUZA et al., 2013). É facilmente notável que o processo de elicitação de requisitos de percepção só acontece depois que o modelo de objetivos é levantado, e assim como o processo de construção de objetivos, *AwReqs* devem ser sistematicamente criados.

*EvoReqs* são requisitos que modificam o espaço de comportamento do sistema, permitindo que novas alternativas de requisitos sejam usadas, baseando-se em um conjunto pré-definido de etapas de evolução para os requisitos monitorados (SOUZA; LAPOUCHNIAN; MYLOPOULOS, 2012). Isto é, *EvoReqs* são requisitos que especificam uma série de operações primárias em relação a outros requisitos diante de determinadas situações, dizendo ao sistema como adaptar-se (SOUZA; LAPOUCHNIAN; MYLOPOULOS, 2012).

Em suma, *AwReqs* especificam quando um determinado objetivo precisa de mudanças para continuar a ser atendido, enquanto *EvoReqs* especificam como executar tais mudanças. A seguir, o modelo de exemplo apresentado na seção 2.1 é novamente apresentado, porém com novos requisitos de adaptação que são devidamente discutidos na próxima sessão.

### 2.1.2 Exemplo de Modelagem de Caso de Uso

Na Figura 2 é apresentado o modelo completo do sistema de despacho de ambulâncias (*Ambulance Computer Aided Dispatch System* ou *ACAD*), nele observa-se o objetivo principal "Gerar Instruções de Despacho Otimizadas", representado por uma elipse, que é imediatamente refinado em outros objetivos e em uma pressuposição de domínio, representada por um quadrado, o tipo de refinamento é “E”. Assim, verifica-se que o primeiro nível de refinamento do objetivo principal é composto de:

- Objetivo “Gerenciar Chamadas”
- Objetivo “Identificação de Recursos”
- Objetivo “Mobilização de Recursos”
- Objetivo “Obtenção de Mapas”
- Pressuposição de Domínio “Dados sobre recursos está sempre atualizado”

O processo de refinamento do modelo então segue até que todos os objetivos sejam completamente refinados em tarefas ou pressuposições de domínio. Os requisitos de percepção são representados por um círculo oco. O modelo da 2 possui ao todo 16

tipos de *AwReqs*. O *AwReq* identificado por “AR5”, indica que a pressuposição de domínio mencionada acima deve “Nunca falhar”. Entretanto os *EvoReqs* referentes a cada um não são representados nesse modelo, e devem ser especificados em forma de tabela. Essa escolha visa facilitar a legibilidade do modelo.

Por fim, *Softgoals* são representados por nuvens e refinados em critérios de operacionalização (os *Quality Constraints*) representados por quadrados de cantos arredondados. Para exemplificar, tem-se o *Softgoal* “Chegada Rápida”, que é operacionalizado por “Ambulâncias chegam em oito minutos”.

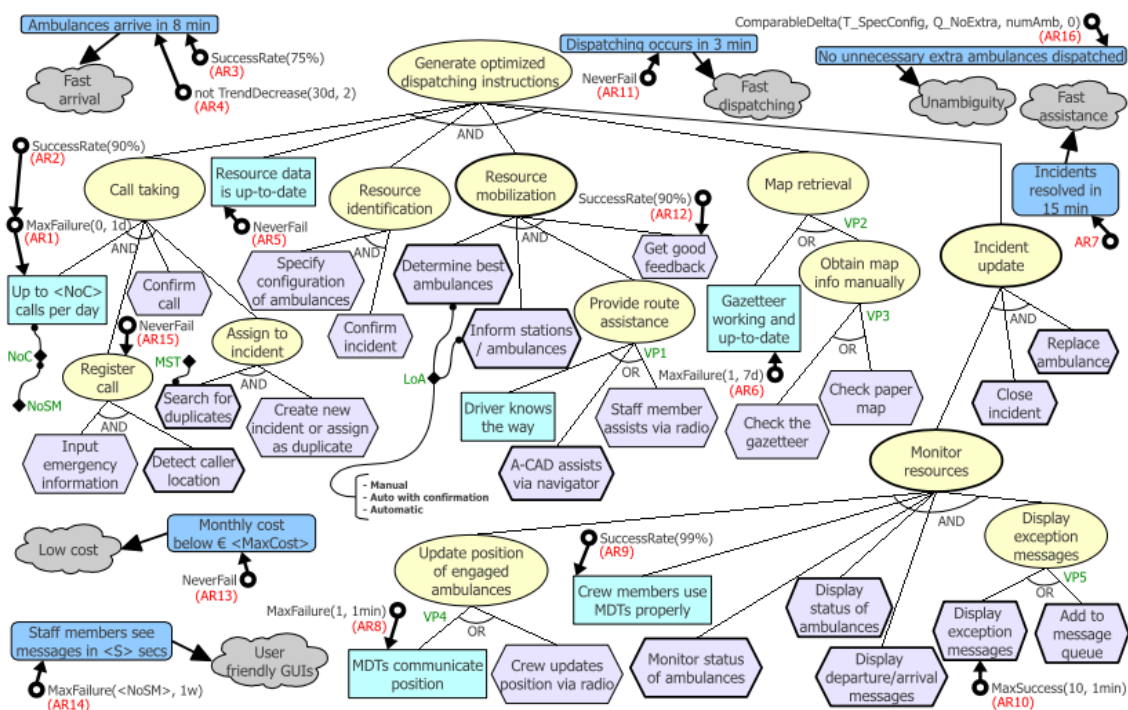


Figura 2 – Exemplo de modelos de objetivos de um sistema de despacho de ambulâncias (SOUZA, 2012)

## 2.2 Zanshin

## 2.3 Desenvolvimento Orientado a Modelos

### 2.3.1 Sirius



## 3 Especificação de Requisitos

Este capítulo aborda alguns resultados da Engenharia de Requisitos para a construção do sistema SAE. Na seção 3.1, é apresentado o escopo do projeto; na seção 3.2, são apresentados diagramas de casos de uso e na seção 3.3, são apresentados os diagramas de classes. Os requisitos funcionais, requisitos não funcionais e regras de negócio podem ser encontrados no **Documento de Especificação de Requisitos** que está disponível no Apêndice ao final desta monografia.

### 3.1 Descrição do Escopo

O DI/Ufes deseja um sistema de informação para acompanhar seus alunos egressos dos cursos de graduação (Ciência da Computação e Engenharia de Computação) e de pós-graduação (Mestrado em Informática e Doutorado em Ciência da Computação).

Para poder acessar o sistema, os egressos terão um pré-cadastro realizado por um administrador do sistema. Somente poderão ser pré-cadastrados ex-alunos que tenham se formado em algum curso oferecido pelo DI/Ufes. Para efetuar o pré-cadastro o administrador buscará os dados do egresso junto à Ufes, a saber: nome, data de nascimento, sexo, e-mail, identidade, CPF, naturalidade e nacionalidade. Também serão informados o curso em que o egresso se formou, o número de sua matrícula, o ano de ingresso e o ano de término.

Assim que o pré-cadastro for realizado, o sistema deverá enviar um e-mail ao egresso com um link que o leva diretamente para uma página onde pode definir sua senha. Para aumentar a segurança, esta página solicita o CPF ou a matrícula do egresso para efetivar a definição de senha. Caso o egresso perca este e-mail, poderá receber outro, devendo para isso entrar no site e informar o seu CPF/matrícula. Reconhecendo o egresso, o sistema enviará o e-mail.

Assim que for criada a senha, o sistema levará o egresso a uma página onde ele preencherá um formulário com os seguintes campos: faixa salarial, área de atuação, se atua na área em que se formou, nível de escolaridade e se reside no ES. Para cada nível de escolaridade deve dizer o título obtido, o ano, a instituição, o estado e o país.

O tempo médio exigido para o preenchimento deste formulário deve ser inferior a 5 minutos. A cada 2 anos o sistema deverá enviar um e-mail para que o usuário atualize esses dados, sendo armazenado o histórico dos mesmos.

Os egressos escolherão a sua área de atuação dentre as seguintes opções: empreendedor; funcionário público; funcionário privado; professor; ou pesquisador. E informarão se

atuam em Informática, área afim ou área não correlata. Será perguntado se a formação acadêmica adquirida no curso da Ufes contribuiu para a sua atividade atual.

Os egressos escolherão a faixa salarial, dividida da seguinte forma: até 3 salários mínimos; de 3 a 5 salários mínimos; de 5 a 10 salários mínimos; de 10 a 15 salários mínimos; de 15 a 20 salários mínimos; e acima de 20 salários mínimos. Poderão também optar por assuntos de interesse para recebimento de e-mail. A princípio os assuntos serão: Redes de Computadores e Sistemas Distribuídos; Computação de Alto Desempenho; Inteligência Computacional; Sistema de Informação; e Otimização.

Egressos poderão postar depoimentos sobre o curso que realizaram. Esses depoimentos ficarão acessíveis a todos que acessarem o site, depois de serem avaliados e liberados pelo coordenador do curso a fim de evitar críticas gratuitas depreciativas. O egresso poderá optar por aparecer seu nome no depoimento ou se ele quer que fique anônimo. De um depoimento deseja-se saber a data de envio, sobre qual curso, o autor e o conteúdo.

Assim como no caso dos depoimentos, os egressos também poderão mandar comentários ou sugestões sobre o curso que realizaram. Estes serão enviadas para o coordenador do curso para que possa respondê-los e também auxiliar em melhorias a serem feitas nos cursos.

Administradores do sistema poderão cadastrar seminários, informando o assunto, o título, a data e horário, o local e o palestrante. Caso não tenha palestrante ainda, o administrador terá a opção de enviar um e-mail aos egressos convidando-os a serem o palestrante. Caso alguém responda ao chamado (por e-mail, externo ao sistema), o administrador terminaria o cadastro do seminário. Assim que a palestra estiver confirmada, o sistema enviará um e-mail para todos os egressos que tenham interesse pelo assunto, convidando-os para participarem. Os egressos também teriam a opção de sugerir um assunto em que tenham interesse em ser o palestrante. Neste caso o administrador confirmaria com ele e cadastraria o seminário no sistema.

No site, ficarão disponíveis para consulta relatórios sobre dados estatísticos. Estes dados serão mostrados na forma de gráficos, assim os usuários poderão escolher um curso e optar pelos seguintes gráficos:

- **Faixa Salarial:** mostra a porcentagem de egresso em cada faixa salarial.
- **Área de Atuação:** mostra a porcentagem de egresso em cada área: (Empreendedor), (Func. Público), (Func. Privado), (Professor) e (Pesquisador).
- **Atuação do Egresso:** mostra a porcentagem de egressos que atuam na área da informática, a porcentagem dos que atuam em áreas afins e a porcentagem dos que atuam em áreas não correlatas.
- **Escolaridade:** mostra a porcentagem de egressos em cada nível de escolaridade.



- **Reside no ES:** mostra a porcentagem de egressos que moram no Estado.
- **Sexo:** mostra a porcentagem de egressos do sexo masculino e feminino.

Os usuários também poderão consultar todos os egressos, que serão mostrados na forma de lista.

## 3.2 Diagrama de Casos de Uso

Este projeto foi dividido em dois subsistemas `sae.core` e `sae.public`, sendo que o subsistema `sae.core` envolve toda a funcionalidade relacionada com o administrador do sistema, abrangendo controle de seminários, cursos, assuntos de interesse, envio de e-mail automático e pré-cadastro de egresso. O subsistema `sae.public` envolve toda a funcionalidade relacionada com consultas a serem realizadas no site, e com as interações que os egressos poderão fazer, tais como cadastrar depoimentos e sugestões. Veremos na Subseção 3.2.2 os casos de uso do subsistema `sae.core` e na Subseção 3.2.3 os casos de uso do subsistema `sae.public`.

### 3.2.1 Atores

O modelo de casos de uso visa capturar e descrever as funcionalidades que um sistema deve prover para os atores que interagem com o mesmo. A Tabela 1 descreve cada um dos atores identificados no sistema.

Tabela 1 – Atores

Ator	Descrição
Administrador	Profissional da Ufes responsável pela parte administrativa do sistema.
Coordenador	É um administrador responsável por um curso, avaliando depoimentos e sugestões enviadas pelos egressos.
Egresso	Ex-alunos da Ufes que tenham se formado em algum curso oferecido pelo DI/Ufes.
Visitante	Qualquer pessoa que acessar o site.

A Figura 3 apresenta o diagrama de herança entre os atores do sistema, de modo que essas heranças não serão mostradas nos outros diagramas para evitar a poluição visual.

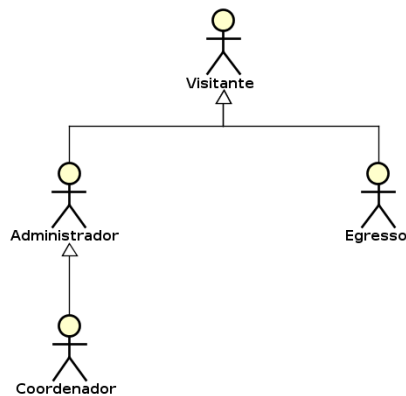


Figura 3 – SAE - CORE - Diagrama de Casos de Uso.

### 3.2.2 Subsistema sae.core

A Figura 4 mostra os casos de uso do subsistema **sae.core** que serão descritos a seguir. O subsistema **sae.core** foi criado para gerenciar as funcionalidades que só os administradores podem realizar. Os casos de uso **Gerenciar Cursos**, **Gerenciar Administradores**, **Gerenciar Egressos**, **Gerenciar Assuntos de Interesse** e **Gerenciar Seminário** são do tipo cadastrais e incluem alteração, inclusão, consulta e exclusão.

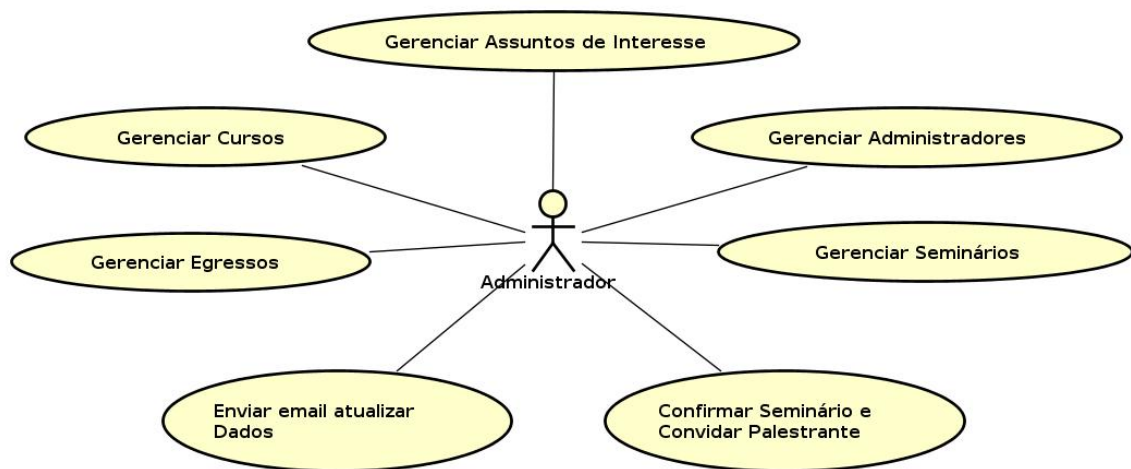


Figura 4 – SAE - CORE - Diagrama de Casos de Uso.

O DI/Ufes possui cursos de graduação e de pós-graduação. Então, foi criado o caso de uso **Gerenciar Cursos** para que o administrador possa inserir novos cursos, possa também consultar, alterar e até mesmo excluir um curso. Para inserir um novo curso, basta informar o código, o nome e o coordenador do curso.

Uma informação crucial para o sistema são os Administradores, que serão controlados pelo caso de uso **Gerenciar Administradores**. Nesse caso, deve ser informado: nome, e-mail (será o login para acessar o sistema), CPF e matrícula.

Os egressos são uma das partes fundamentais do sistema, assim serão controlados

pelo caso de uso **Gerenciar Egresso**. As informações necessárias de um egresso são: nome, e-mail (será o login para acessar o sistema), data de nascimento, sexo, identidade, CPF, naturalidade e nacionalidade, também são necessários informar o curso, a matrícula, o ano de início e de término do curso.

Os egressos poderão escolher assuntos de interesse para recebimento de e-mail. A princípio os assuntos serão: Redes de Computadores e Sistemas Distribuídos; Computação de Alto Desempenho; Inteligência Computacional; Sistema de Informação; e Otimização. Assim foi criado o caso de uso **Gerenciar Assuntos de Interesse** para realizar esse controle. A informação de um assunto é o nome.

Seminários também poderão ser cadastrados no sistema. Assim teremos os casos de uso **Gerenciar Seminário**, **Confirmar Seminário** e **Convidar Palestrante** para fazer o controle. O caso de uso *Gerenciar Seminário* será cadastral, enquanto o *Confirmar Seminário* e *Convidar Palestrante* envolve atividades como enviar e-mail a todos os egressos que tenham interesse no assunto do seminário assim que este for confirmado, enviar e-mail aos egressos convidando a serem o palestrante de seminário cujo assunto é de seu interesse.

Para manter os dados dos egressos atualizados será enviado, a cada 2 anos, um e-mail para todos os egressos solicitando que estes façam a atualização de seus dados. O caso de uso **Enviar e-mail atualizar dados** será responsável por este envio de e-mail.

### 3.2.3 Subsistema sae.public

A Figura 5 mostra os casos de uso do subsistema **sae.public** que serão descritos abaixo. Este subsistema foi criado para gerenciar as funcionalidades relacionadas com consultas a serem realizadas no site e com as interações que os egressos poderão fazer, tais como cadastrar depoimentos e sugestões através dos casos de uso **Gerenciar Depoimentos**, **Gerenciar Sugestões**, **Gerenciar Escolaridades** e **Gerenciar Históricos**. Os casos de uso de consulta são **Consultar Todos Egressos**, **Consultar Depoimento** e **Consultar dados Estatísticos**, que poderão ser realizado por qualquer usuário do sistema.

No caso de uso **Consultar Todos Egressos** as consultas poderão ser feitas de forma geral onde serão mostrados todos os egressos, ou por curso, onde serão mostrados apenas os egressos que formaram naquele curso. Será exibido na tela para ao usuário o nome do egresso, o curso que realizou, o ano de início e o ano de término.

No caso de uso **Consultar Depoimento** as consultas aos depoimentos poderão ser realizadas de forma geral onde serão mostrados todos os depoimentos, ou por curso, onde serão mostrados apenas depoimentos sobre o curso escolhido. Será exibido na tela o conteúdo, o autor e a data de postagem. Somente serão mostrados nesta consulta depoimentos que tenham sido analisados e aprovados pelo coordenador do curso que se

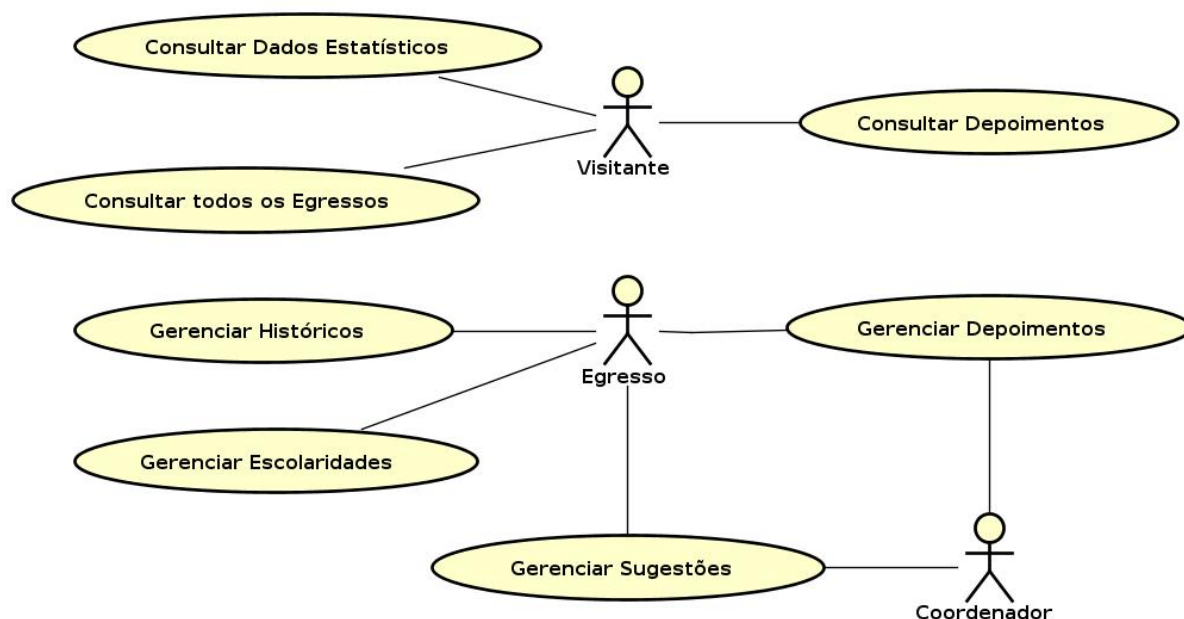


Figura 5 – SAE - PUBLIC - Diagrama de Casos de Uso.

refere o depoimento.

No caso de uso **Consultar dados Estatísticos** as consultas serão feitas com bases nos dados mais atuais dos egressos. Alguns exemplos de gráficos que poderão ser gerados nesta consulta são *Faixa Salarial*, *Área de Atuação*, *Escolaridade e Sexo*.

Egressos poderão postar depoimentos sobre o curso que realizaram. Esses depoimentos ficarão acessíveis a todos que acessarem o site, depois de serem avaliados e liberados pelo coordenador do curso a fim de evitar críticas gratuitas depreciativas, assim foi criado o caso de uso **Gerenciar Depoimentos** para fazer esse controle.

Assim como no caso dos depoimentos, os egressos também poderão mandar comentários ou sugestões sobre o curso que realizaram. Estes serão enviadas para o coordenador do curso para que possa respondê-los o caso de uso responsável por fazer esse controle é **Gerenciar Sugestões**.

No caso de uso **Gerenciar Históricos** o egresso informará sua faixa salarial, área de atuação que pode ser funcionário no setor público ou no setor privado, empreendedor, professor ou pesquisador, informará também se atua na área da informática, se reside no Espírito Santo e o seu maior nível de escolaridade. Com essas informações será possível criar o perfil dos egressos.

No caso de uso **Gerenciar Escolaridades** o egresso poderá cadastrar todos seus cursos realizados a nível de graduação, especialização, mestrado, doutorado ou pós-doutorado. Para cada curso ele informará a instituição, o estado e o país, e o ano de conclusão.

Maiores informações e detalhes sobre os casos de uso poderão ser consultados no **Documento de Análise de Requisitos** que está disponível no Apêndice ao final dessa monografia.

### 3.3 Diagrama de Classes

Assim como os casos de uso na seção 3.2 os diagramas de classes estão divididos de acordo com a divisão dos subsistemas, na Subseção 3.3.1 estão as classes pertencentes ao subsistema *sae.core* e na Subseção 3.3.2 estão as classes pertencentes ao subsistema *sae.public*.

#### 3.3.1 Subsistema *sae.core*

A Figura 6 exibe o diagrama de classes do subsistema *sae.core*. Uma das classes mais importante é a **Egresso** que possui ligações com outras classes tanto no subsistema *sae.core* quanto no subsistema *sae.public*. É obrigatório que um *egresso* possua um *curso*, que será feito através da classe **Curso Realizado** visto que para ser egresso do DI/Ufes é preciso ter realizado um curso. Entretanto é opcional um *egresso* ter um *assunto de interesse*, podendo ter mais de um.

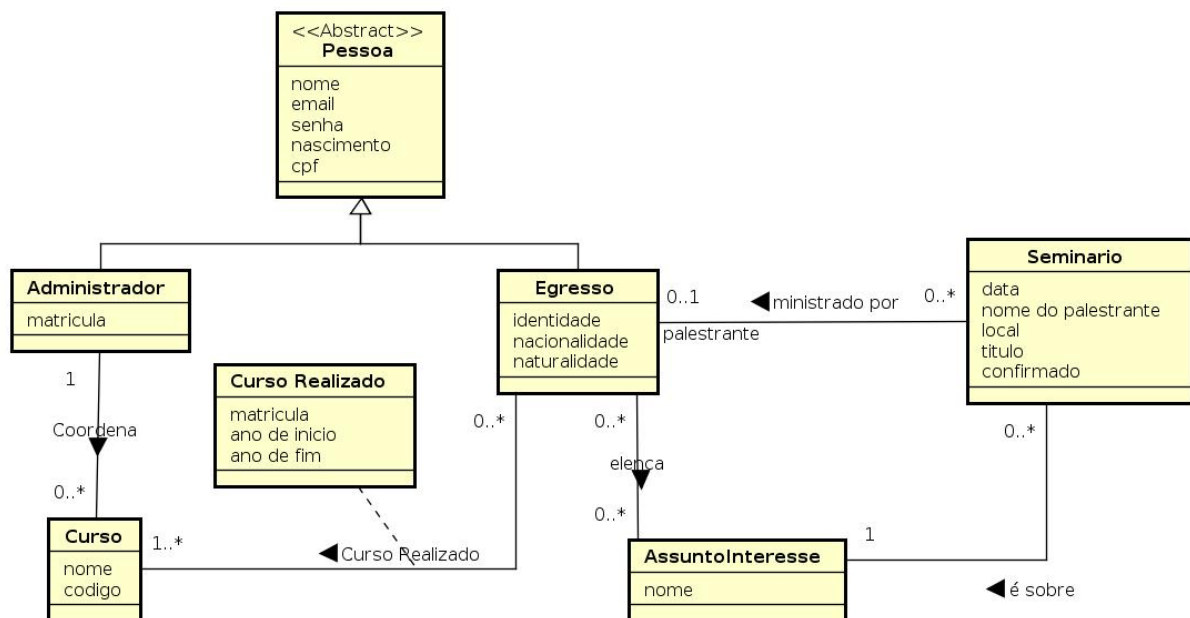


Figura 6 – SAE - CORE - Diagrama de Classes.

As classes **Administrador** e **Assunto de Interesse** podem ter registros no sistema e mesmo assim não estarem ligadas a nenhuma outra classe. Todo *curso* deve ter um *administrador* associado a ele, visto que este desempenhará o papel de coordenador do curso, sendo responsável por avaliar depoimento e responder sugestões do curso que coordena. Um *seminário* precisa ter, obrigatoriamente, um *assunto de interesse*.

### 3.3.2 Subsistema sae.public

A Figura 7 exibe o diagrama de classes do subsistema **sae.public**. Podemos notar que as classes **Egresso** e **Curso** foram referenciadas do subsistema **sae.core**. Portanto, fazem parte desse subsistema as classes **Depoimento**, **Sugestão**, **Escolaridade** e **Histórico do Egresso**. Uma *escolaridade* e um *histórico do egresso* devem estar associados a um *egresso*. Um *depoimento* e uma *sugestão*, além de um *egresso*, também devem ter um *curso* associado a eles.

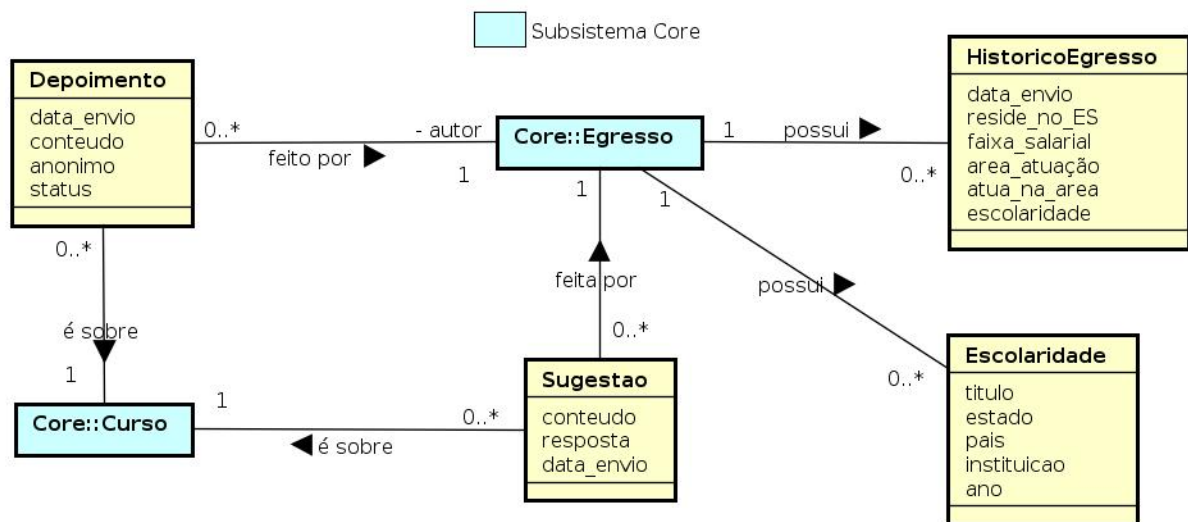


Figura 7 – SAE - PUBLIC - Diagrama de Classes.

Esse diagrama possui uma única restrição de integridade: uma sugestão feita por um egresso deve ser sobre um curso que este egresso tenha realizado, o mesmo vale para a classe **Depoimento**.

Maiores informações sobre o diagrama de classes poderão ser consultados no Documento de Especificação de Requisitos, disponível no Apêndice ao final dessa monografia.

## 4 Projeto Arquitetural e Implementação

Seguindo a fase de especificação e análise de requisitos ocorre a fase de projeto que envolve a modelagem de como será a implementação do sistema, incorporando aos requisitos as tecnologias a serem utilizadas.

Neste capítulo iremos mostrar a arquitetura do projeto, assim como algumas partes de sua implementação e apresentar as principais telas do sistema. Na seção 4.1, a arquitetura do sistema é descrita, na seção 4.2, as framework nemo-utils é apresentado, na seção 4.3 os modelos FrameWeb são apresentados. Por fim, na seção 4.4, são apresentadas algumas telas e características da ferramenta.

### 4.1 Arquitetura do Sistema

No projeto arquitetural, o SAE foi dividido em dois módulos principais (implementados como pacotes Java), seguindo a divisão de subsistemas feita na análise dos requisitos e apresentada no Capítulo ?? . A Figura 8 mostra os módulos que formam a arquitetura do SAE .

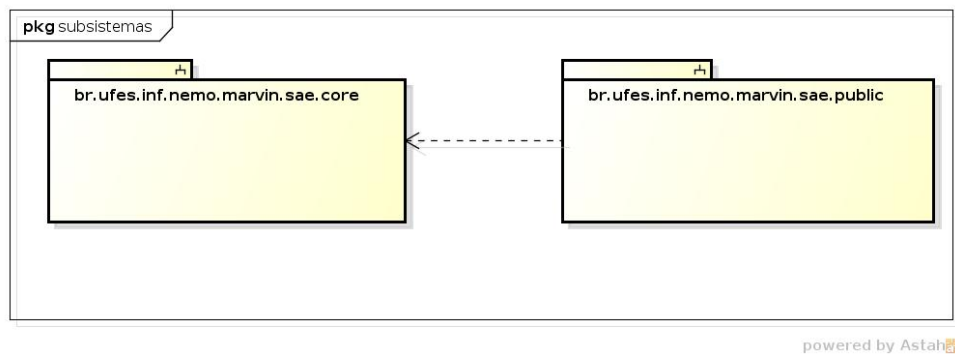


Figura 8 – Pacotes que formam a arquitetura do SAE.

O módulo `br.ufes.inf.nemo.marvin.sae.core` contém as funcionalidades do subsistema **sae.core**, enquanto o módulo `br.ufes.inf.nemo.marvin.sae.public` contém as funcionalidades do subsistema **sae.public**. Mais à frente iremos detalhar um pouco mais as subdivisões desses módulos.

Os módulos principais do SAE são ainda subdivididos em camadas segundo a arquitetura que pode ser verificada na Figura 9. O sistema SAE foi dividido em três camadas, sendo elas: apresentação (*Presentation Tier*), negócio (*Business Tier*) e acesso a dados (*Data Access Tier*). Esta Figura mostra, também, as tecnologias Java associadas a cada pacote. Tais tecnologias foram abordadas na Seção ??.



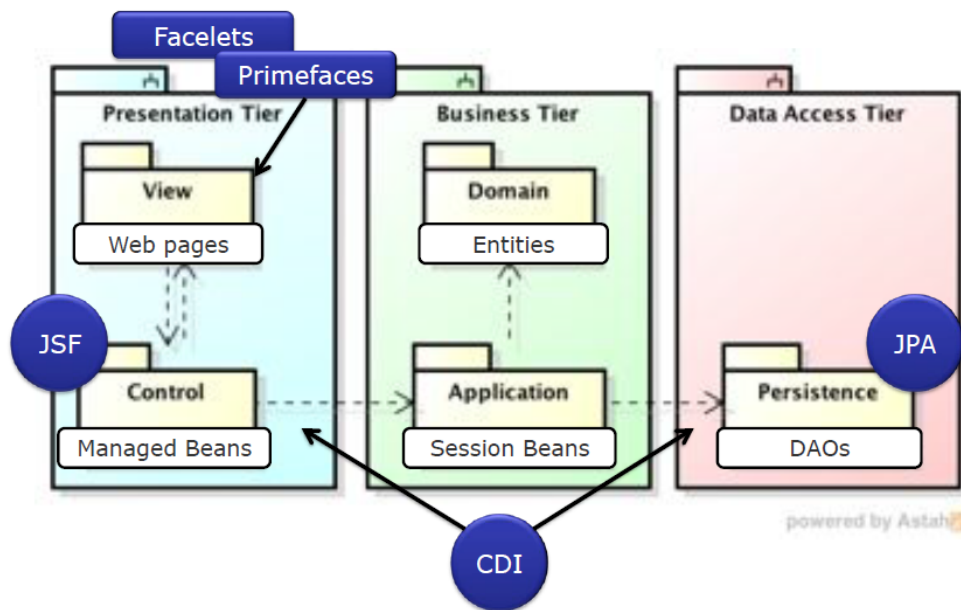


Figura 9 – SAE - Arquitetura - Sistema (??).

A Figura 10 exibe os pacotes do sistema SAE. Como podemos perceber, os pacotes foram agrupados pelos módulos principais e pelas camadas da arquitetura. A seguir iremos detalhar um pouco mais cada um deles.

```

▶ br.ufes.inf.nemo.marvin.sae.core.application
▶ br.ufes.inf.nemo.marvin.sae.core.control
▶ br.ufes.inf.nemo.marvin.sae.core.domain
▶ br.ufes.inf.nemo.marvin.sae.core.persistence
▶ br.ufes.inf.nemo.marvin.sae.publico.application
▶ br.ufes.inf.nemo.marvin.sae.publico.control
▶ br.ufes.inf.nemo.marvin.sae.publico.domain
▶ br.ufes.inf.nemo.marvin.sae.publico.persistence

```

Figura 10 – SAE - Implementação - Pacotes.

#### 4.1.1 Camada de Apresentação

A **camada de apresentação** foi subdividida em visão (*View*) e controle (*Control*). A parte da visão é formada pelas páginas Web. A parte de controle contém os controladores que realizam a comunicação entre a interface e a aplicação.

A estrutura Web do sistema SAE, cujas páginas Web fazem parte da visão da camada de apresentação está organizada conforme a Figura 11. Existe uma pasta raiz chamada **WebContent** que contém todos os arquivos da visão. Ela possui duas subpastas que representam os módulos do SAE: **sae/core** e **sae/public**. Dentro de cada uma dessas, uma nova pasta foi criada para tratar cada caso de uso de forma separada. Isso ajuda na organização e caso seja necessário criar um novo caso de uso, basta adicionar uma nova



pasta e os arquivos necessários. A subpasta **sae/public** ainda foi dividida em **search**, para os casos de uso de consulta e **alumni** para os casos de usos relacionado aos egressos.

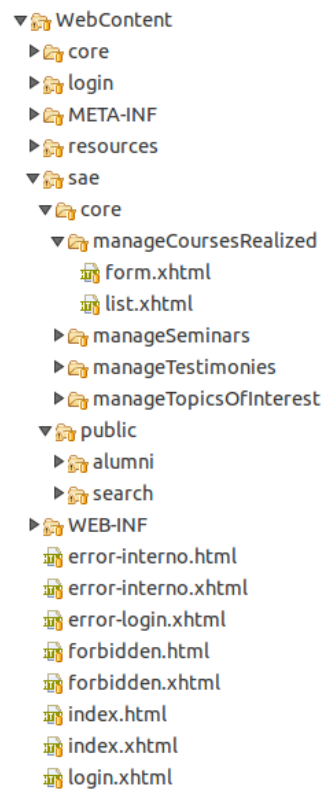


Figura 11 – SAE - Implementação - Páginas Web

As pastas que implementam os casos de uso seguem um padrão que foi definido no framework do *nemo-utils* (cf. Seção 4.2). Esse padrão de visão consiste em duas páginas, sendo a primeira chamada **form.xhtml**, que é responsável por elencar os dados das entidades para que possam ser modificados e armazenados no banco de dados. Já a página **list.xhtml** é responsável por recuperar e exibir para o usuário as informações da entidade que estão armazenadas no banco de dados.

Dentro da pasta raiz **WebContent**, temos as páginas **index.html** e **index.xhtml** que são as páginas iniciais do sistema. As páginas **error-interno.xhtml** e **error-interno.html** que serão utilizadas para tratar de erros internos do servidor. As páginas **forbidden.html** e **forbidden.xhtml** que são usadas para o caso do usuário tentar acessar uma página que não tenha acesso. Temos ainda as páginas **login.xhtml** e **error-login.xhtml** que são as páginas do formulário de login e de erro ao efetuar o login respectivamente.

O decorador será utilizado para definir o layout da página e o menu que está sendo utilizado. A pasta **resources** contém a subpasta **default** que contém o decorador.

### 4.1.2 Camada de Negócio

A **camada de negócio** foi subdividida em domínio (*Domain*) e aplicação (*Application*). A parte do domínio é formada pelas entidades do negócio, enquanto a aplicação contém as validações dos dados e implementação dos casos de uso (funcionalidades do sistema).

Os pacotes `sae.core.domain` e `sae.public.domain` contêm a definição das entidades do sistema SAE. Cada uma dessas entidades está definida em um arquivo `*.java` e já realiza o mapeamento objeto-relacional para o banco de dados. Através desse mapeamento, o JPA irá criar os objetos no banco de dados automaticamente, sem precisar de nenhuma intervenção do desenvolvedor. É nesse momento que é realizado também o relacionamento entre as classes do sistema utilizando as anotações `@OneToMany`, `@ManyToOne` ou `@ManyToMany` de JPA. Por fim, nesse pacote existe um arquivo para cada classe com o mesmo nome e um “\_” no final (chamada de *static meta-model* ou metamodelo estático), que declara os atributos que poderão ser utilizados para realizar as consultas no banco de dados utilizando os conceitos de Criteria API. Essas consultas serão implementadas na camada de persistência.

Os pacotes `sae.core.application` e `sae.public.application` contêm os componentes que fazem a comunicação entre a apresentação (controladores) e a persistência (DAOs), implementando as funcionalidades do sistema descritas em seus casos de uso (cf. Cap. ??). Faz também as validações das informações antes de chamar os métodos de acesso a dados. Essas validações serão feitas ao tentar criar ou modificar uma entidade.

### 4.1.3 Camada de Acesso a Dados

A **camada de acesso a dados** possui uma única parte responsável pela persistência (*Persistence*) representada pelos pacotes `sae.core.persistence` e `sae.public.persistence`, que contêm os objetos responsáveis por fazer a comunicação com o banco de dados. Esses objetos são conhecidos como DAO (cf. Seção ??) e serão responsáveis por armazenar e recuperar os dados do banco de dados.

Sobre a arquitetura do banco de dados, conforme explicado anteriormente, o sistema SAE utiliza o JPA para fazer o mapeamento objeto relacional e, através desse mapeamento, o próprio JPA irá criar os objetos no banco de dados automaticamente. Com isso, foi utilizada a anotação `@Entity` nas classes do domínio para realizar a persistência dos dados.

## 4.2 Framework *nemo-utils*

Nesta seção vamos falar um pouco sobre o framework *nemo-utils*<sup>1</sup>, que foi utilizado para implementar o sistema SAE. No Documento de Requisitos, a **RNF07** diz que “*O desenvolvimento do sistema deve explorar o potencial de reutilização de componentes, tanto no que se refere ao desenvolvimento com reuso quanto ao desenvolvimento para reuso*”. Assim, foi utilizado o framework *nemo-utils* que provê uma série de facilidades, pois ele já implementa as operações básicas entre a aplicação e o banco de dados de uma forma genérica, bastando ao desenvolver adaptar os códigos para as entidades do domínio do seu problema. Com isso, não foi necessário despendar tempo criando funcionalidades que já estavam implementadas no framework.

A maioria dos arquivos dos pacotes `sae.core.control` e `sae.public.control` herdam da classe `CrudController` do *nemo-utils*. Essa classe é responsável por armazenar temporariamente os dados das páginas Web e depois fazer a comunicação com a camada de aplicação. Em algumas páginas, também são responsáveis por carregar os dados dos componentes `selectOneMenu` do **PrimeFaces**. Além disso, realizam os filtros de pesquisa através do método `initFilters`.

A maioria dos arquivos dos pacotes `sae.core.application` e `sae.public.application` herdam da classe `CrudServiceBean` do *nemo-utils*. Essa classe é responsável por realizar as validações e por fazer a comunicação com a camada de acesso a dados. Essa classe possui alguns métodos responsáveis pelas validações. Estes métodos são vazios na classe `CrudServiceBean` e precisam ser implementados de acordo com as validações a serem realizadas em cada caso, são eles:

- **validateCreate** - responsável por fazer as validações ao tentar criar uma nova entidade no sistema. Também possui validações para evitar que dados duplicados sejam inseridos no sistema;
- **validateUpdate** - responsável por fazer as validações ao tentar atualizar os dados de uma entidade já existente no sistema. Também possui validações para evitar que dados duplicados sejam inseridos no sistema;
- **validateDelete** - responsável por fazer as validações ao tentar excluir os dados de uma entidade já existente no sistema. Em alguns casos, algumas classes não podem ser excluídas se tiverem algum relacionamento com outra classe no sistema. Por exemplo, não é possível excluir um professor que possua uma turma.

Utilizando o conceito de herança da programação orientada a objetos, quase todas as entidades do domínio herdam da classe `PersistentObjectSupport`, que é uma imple-

<sup>1</sup> *nemo-utils* – <https://github.com/nemo-ufes/nemo-utils>

mentação padrão para objetos persistentes que utiliza EJB 3 como padrão de anotações para persistência. Essas classes estão nos pacotes `sae.core.domain` e `sae.public.domain` e possuem os seguintes atributos: `serialVersionUID`, `uuid`, `id` e `version`. Nesse caso, é importante saber que o campo `id` será usado para identificar unicamente uma entidade no banco de dados, o campo `uuid` é um número gerado aleatoriamente para diferenciar unicamente uma entidade e o campo `version` identifica a versão da entidade.

Por último, os arquivos dos pacotes `sae.core.persistence` e `sae.public.persistence` herdam da classe `BaseJPDAO` do *nemo-utils*. Essa classe é responsável por realizar as operações no banco de dados, sendo elas: consulta, modificação, inserção e exclusão de dados. Todas as consultas são realizadas utilizando os conceitos de Criteria API do JPA. Como as consultas que foram implementadas são bem simples utilizando poucas restrições, grande parte do código foi reaproveitado para todas as classes, alterando apenas o tipo e os atributos.

## 4.3 Modelos FrameWeb

Nesta seção, serão exibidos os modelos FrameWeb que já foram citados anteriormente na Seção ???. Esses modelos também estão divididos nas camadas da arquitetura do sistema, conforme citado na Seção 4.1.

### 4.3.1 Modelo de Domínio

Os mapeamentos de persistência são meta-dados das classes de domínio que permitem que os *frameworks* ORM transformem objetos que estão na memória em linhas de tabelas no banco de dados relacional. Por meio de mecanismos leves de extensão da UML, como estereótipos e restrições, adicionamos tais mapeamentos ao diagrama de classes de domínio. Apesar de tais configurações serem relacionadas mais à persistência do que ao domínio, elas são representadas no Modelo de Domínio porque as classes que são mapeadas e seus atributos são exibidas neste diagrama. A Figura 12 mostra o modelo de domínio para o módulo `sae.core`.

Podemos observar nesta figura que quase todos atributos tem tamanho (`size`) definido. As classe **Egresso** e **Seminário** têm atributos do tipo data, na restrição destes atributos informamos se precisão vai ser *time*, armazenando no banco de dados somente a hora, *date*, apenas a data, ou *timestamp* armazenado ambos, data e hora. Neste último caso não é preciso colocar na restrição pois é a opção *default*.

A associação entre as classes **Egresso** e **AssuntoInteresse** tem uma restrição `fetch` que indica qual a estratégia de recuperação do banco de dados. Nesta associação está especificada a opção *lazy*, que significa que a recuperação vai ser no modo preguiçoso, ou seja, somente vai trazer do banco de dados quando precisar da informação.

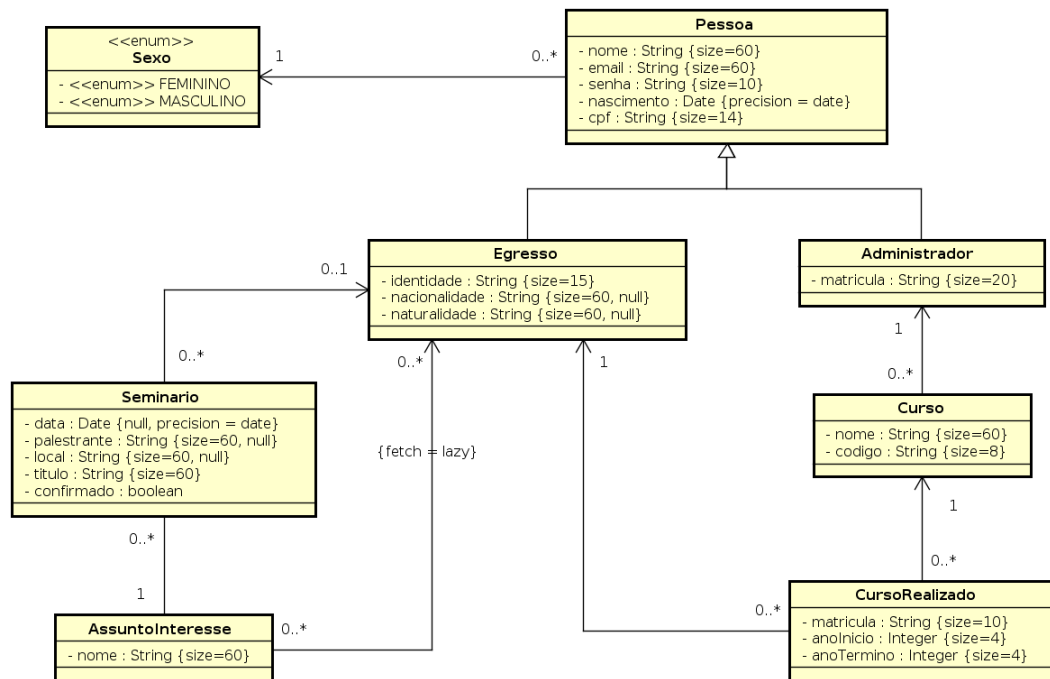


Figura 12 – FrameWeb - sae.core - Modelo Domínio.

A Figura 13 mostra o modelo de domínio para o módulo `sae.public`. As classes **Sugestão** e **Depoimento** possuem atributos marcados com estereótipo *lob*, isso significa que no banco de dados será criado um campo de tamanho grande como *clob* que pode ter até 4GB de dados para armazenamento de texto e *blob* que também pode ter até 4GB de dados binários, este último para armazenar informações digitais como imagens, áudios e vídeos.

Todas as classes de domínio estendem de *PersistentObjectSupport* do framework *nemo-utils*, sendo que essa herança não é mostrada no diagrama com o intuito de não poluí-lo com várias associações.

Diferente da abordagem original do FrameWeb original proposto em 2007, todos os atributos que são não nulos tiveram a *tag not null* omitida e os que são nulos tiveram a *tag null* acrescida de forma a diminuir a poluição visual com repetições desnecessárias no diagrama.

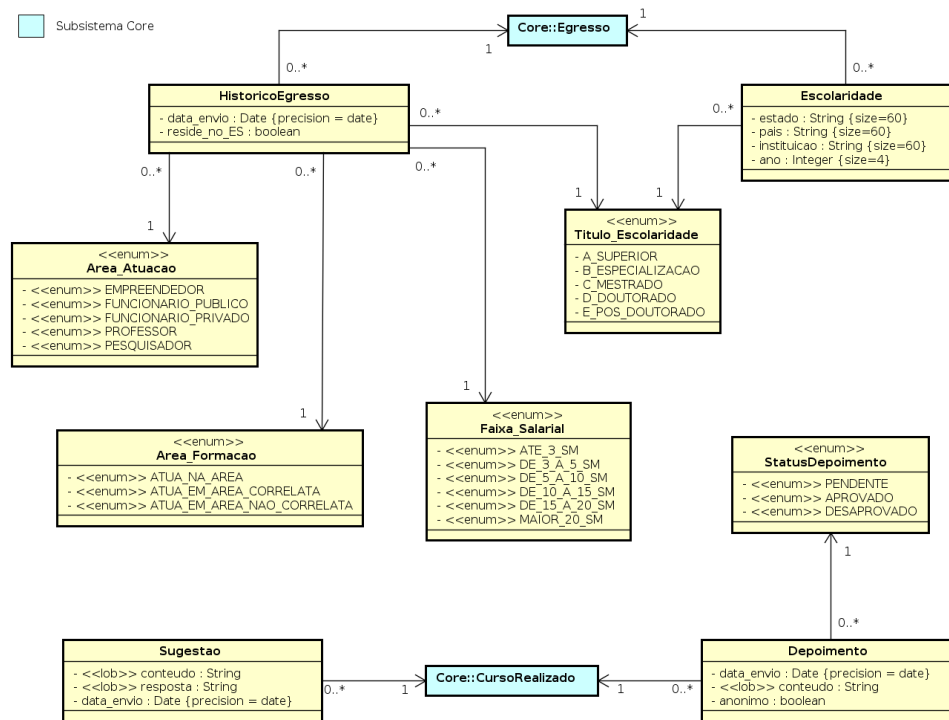


Figura 13 – FrameWeb - Public - Modelo Domínio.

### 4.3.2 Modelo de Navegação

O **Modelo de Navegação** é um diagrama de classe da UML que representa os diferentes componentes que formam a camada de Apresentação, como páginas Web, formulários HTML e classes de ação. Esse modelo é utilizado pelos desenvolvedores para guiar a codificação das classes e componentes dos pacotes **Visão** e **Controle**.

Em formulários HTML, atributos representam campos do formulário, que devem ter seus tipos definidos de acordo com a biblioteca de componentes utilizada, como neste trabalho foi utilizado PrimeFaces os tipos ficarão com o prefixo “p” (ex.: p.input, p.checkbox, p.button, etc.). A classe de ação é o principal componente do modelo. Suas associações de dependência ditam o controle de fluxo quando uma ação é executada.

As funcionalidades criar, editar, excluir e visualizar (abreviadas de CRUD, do inglês *create, read, update e delete*), seguem um mesmo fluxo de execução e de interação com o usuário. Tais funcionalidades são similares para todos os casos de uso cadastrais devido a utilização da ferramenta *nemo-utils*. Esse fluxo de execução similar é representado pela Figura 14 que é um modelo de apresentação genérico.

Para os casos de uso que apresentam funções diferentes das CRUDs, o modelo anterior não pode ser aplicado. A Figura 15 é um modelo de navegação para o caso de uso “Consultar Depoimento”.

Podemos perceber que o modelo possui duas páginas web marcadas com estereótipo «page», a página *index.xhtml* possui um formulário marcado com estereótipo «form» que

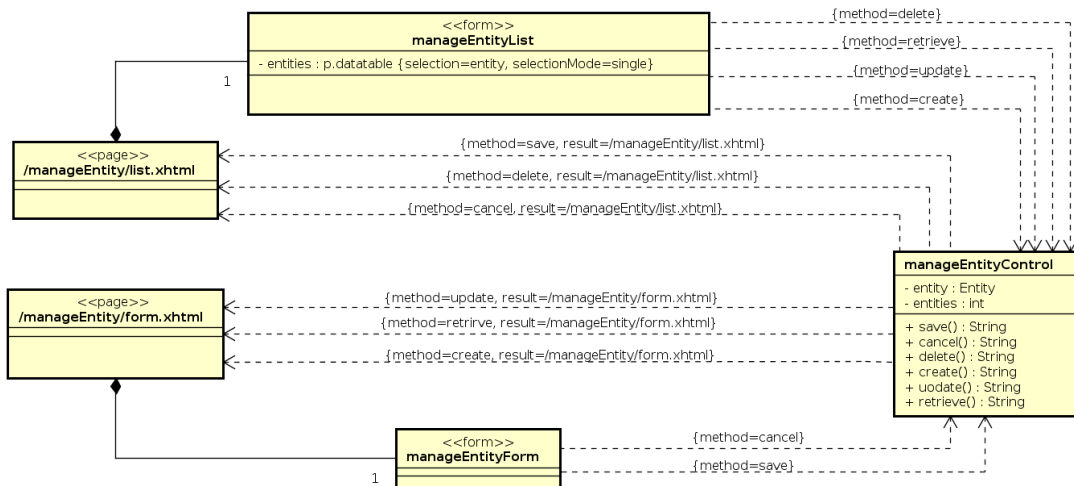
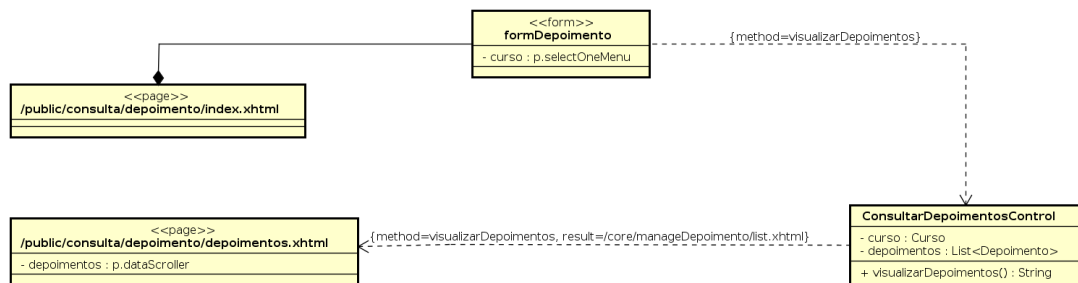
Figura 14 – FrameWeb - *nemo-utils* - Modelo Navegação (??).

Figura 15 – FrameWeb - Consultar Depoimentos - Modelo Navegação.

possui o atributo *curso*, este é injetado via EL (Expression Language) na classe chamada **ConsultaControl** que representa o controlador. Após selecionado o curso, o formulário aciona o método *visualizarDepoimento()* do controlador, o mesmo processa a requisição e mostra o resultado na página *depoimentos.xhtml*.

#### 4.3.3 Modelo de Aplicação

O **Modelo de Aplicação** é um diagrama de classes da UML que representa as classes de serviço, responsáveis pela codificação dos casos de uso, e suas dependências. Esse diagrama é utilizado para guiar a implementação das classes do pacote **Aplicação** e a configuração das dependências entre os pacotes **Controle**, **Aplicação** e **Persistência**, ou seja, quais classes de ação dependem de quais classes de serviço e quais DAOs são necessários para que as classes de serviço alcancem seus objetivos (??).

Todas as classes de aplicação que são cadastrais estendem de *CrudServiceBean* do pacote *nemo-utils*, porém com uma pequena alteração, foi adicionado a classe uma anotação `@PermitAll`, permitindo o controle de segurança. Tal classe está representada na Figura 16 de forma genérica (**Entity** é implementado como um politipo/tipo genérico **T** no código da classe). Da mesma forma dos diagramas anteriores essa herança não é mostrada

no diagrama acima com o intuito de não poluir o diagrama com várias associações.

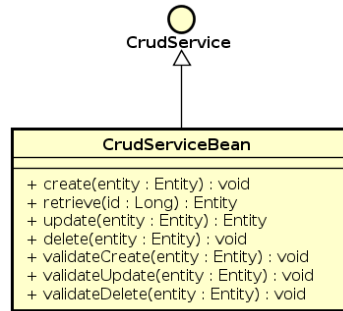


Figura 16 – FrameWeb - *nemo-utils* - Modelo Aplicação (??).

A Figura 17 mostra o modelo de aplicação para o módulo *sae.public*. Já a Figura 18 mostra o modelo de aplicação para o módulo *sae.core*.

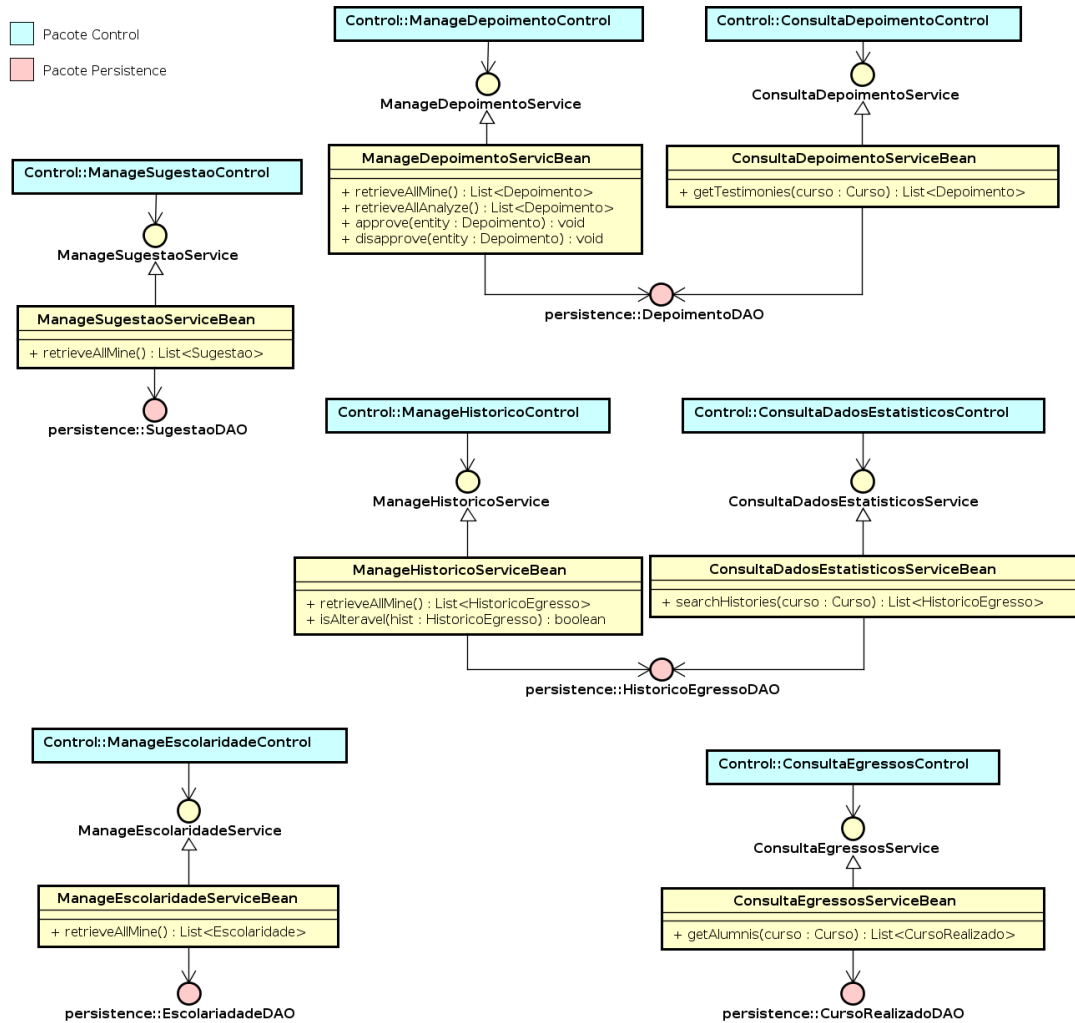


Figura 17 – FrameWeb - Public - Modelo Aplicação.



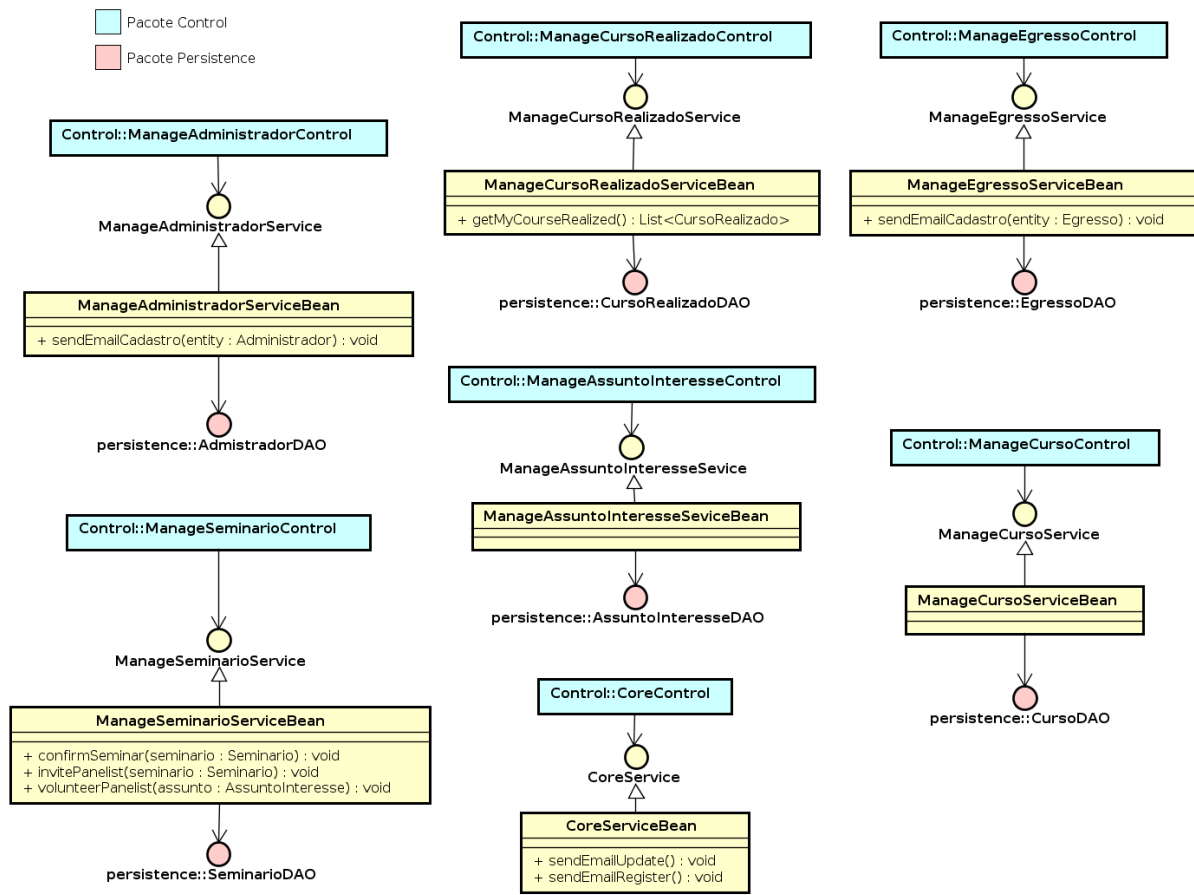


Figura 18 – FrameWeb - Core - Modelo Aplicação.

#### 4.3.4 Modelo de Persistência

O FrameWeb indica a utilização do padrão de projeto DAO para a construção da camada de acesso a dados. O **Modelo de Persistência** é um diagrama de classes da UML que representa as classes DAO existentes, responsáveis pela persistência das instâncias das classes de domínio. Esse diagrama guia a construção das classes DAO, que pertencem ao pacote de persistência.

Para que não seja necessário repetir em cada interface DAO operações que são comuns a todas elas (ex.: `save()`, `delete()`, `retrieveById()`, etc.), podemos apresentar DAOs base que declaram esses métodos – novamente, uma interface e várias implementações. Automaticamente, todas as interfaces DAO de todos os diagramas herdam as definições da interface base, ocorrendo o mesmo com as implementações concretas de cada tecnologia de persistência, sem que isso precise estar explícito no diagrama. A Figura 19 exibe as classes bases do *nemo-utils*.

Tanto a interface **BaseDAO** quanto a classe **BaseJPADA** são declaradas usando tipos genéricos, deixando a cargo de suas subinterfaces e subclasses a especificação da classe gerenciada por cada DAO. O DAO base define métodos para recuperar todos os objetos de uma determinada classe, recuperar um objeto dado seu identificador, salvar e excluir um

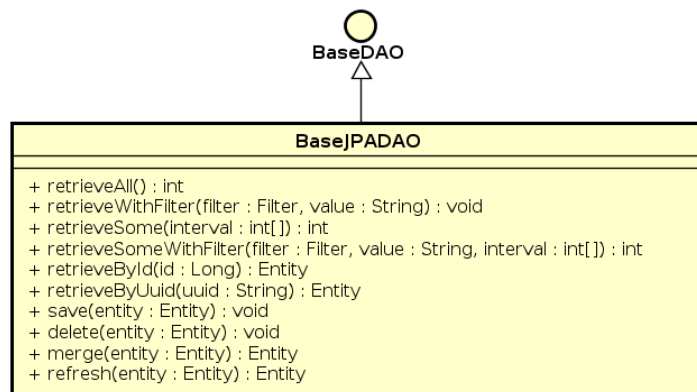


Figura 19 – FrameWeb - *nemo-utils* - Modelo Persistência (??).

objeto. Também não será necessário exibir os métodos do DAO na implementação e na interface, basta modelá-los em apenas um dos dois. No caso do DAO Base, subentende-se que todos os métodos públicos de BaseJPADAO são definidos na interface BaseDAO.

Segundo os padrões estabelecidos por FrameWeb, todas as interfaces DAO são subinterfaces de BaseDAO, enquanto todas as implementações JPA são subclasses de BaseJPADAO, herdando todos os métodos básicos, por exemplo: `retrieveAll()`, `save()`, `delete()`, `retrieveById()`. Os demais métodos que foram declarados no diagrama se referem a consultas específicas que devem ser disponibilizadas para o funcionamento de determinados casos de uso.

As Figuras 20 e 21 são os modelos de persistência para os módulos `sae.public` e `sae.core`, respectivamente.

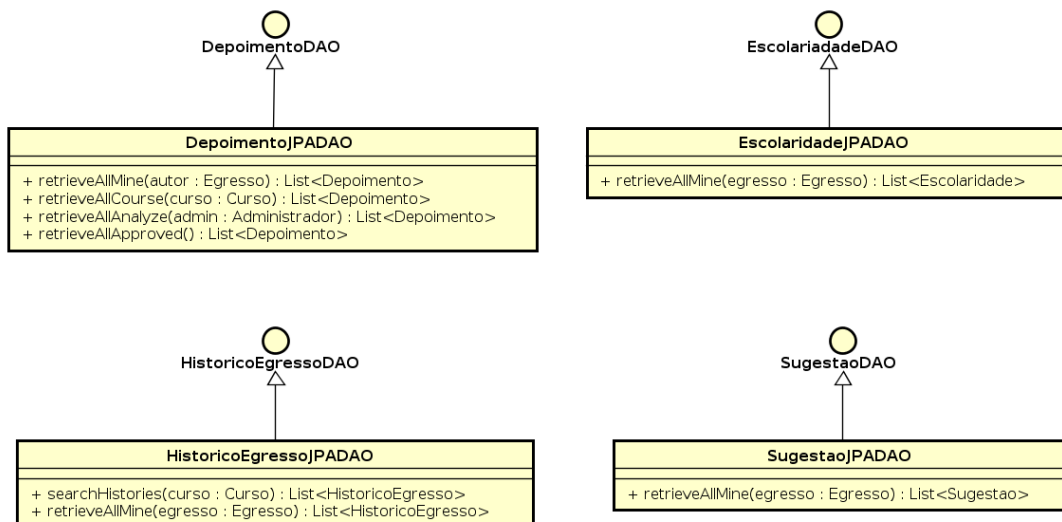


Figura 20 – FrameWeb - Public - Modelo Persistência.

Como é possível perceber, o Modelo de Persistência não define nenhuma extensão da UML para representar os conceitos necessários da camada de acesso a dados, mas

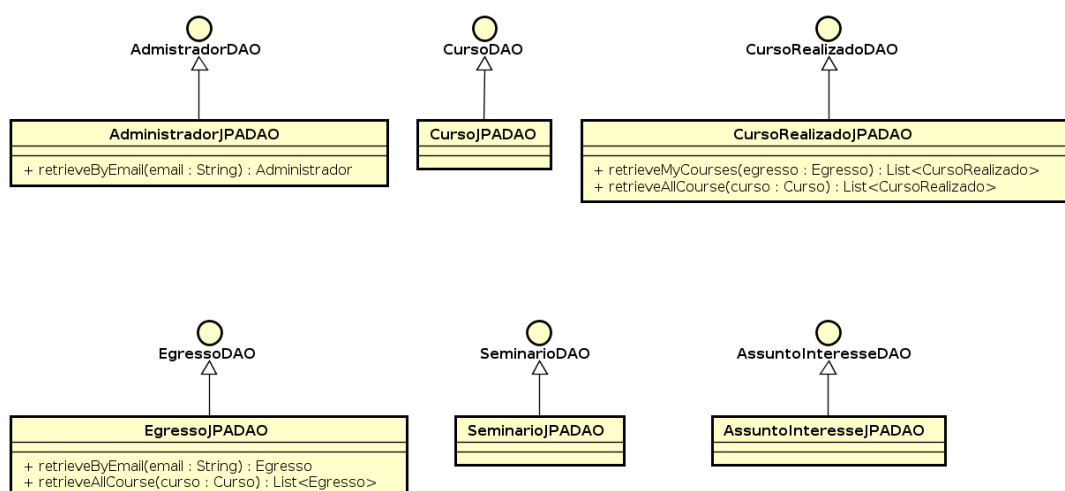


Figura 21 – FrameWeb - Core - Modelo Persistência.

apenas regras que tornam essa modelagem mais simples e rápida, por meio da definição de padrões.

## 4.4 Apresentação do Sistema

Nesta seção, apresentamos o sistema por meio de uma série de capturas de tela. A Figura 22 mostra a tela inicial de login no sistema.

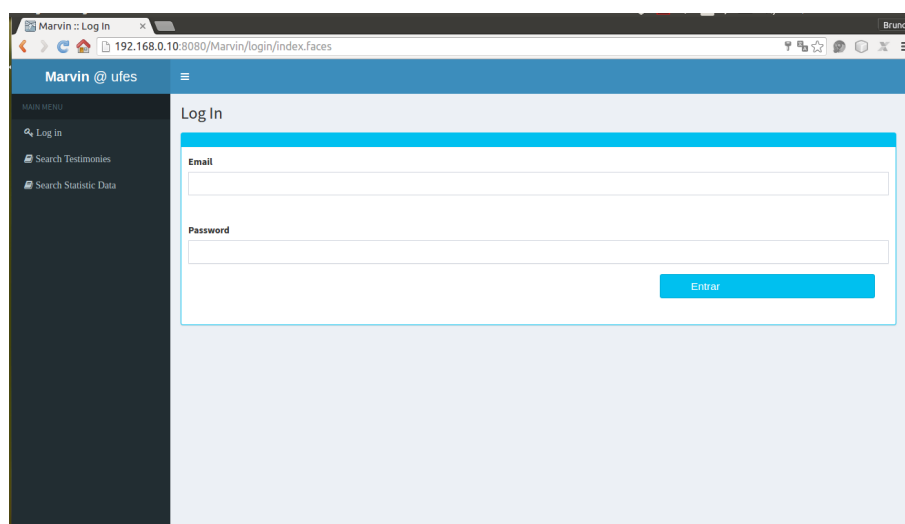


Figura 22 – SAE - Tela Login.

Podemos ver no canto esquerdo da tela uma barra com os menus do sistema, quando um usuário faz login no sistema aparecerão nesta barra as funções que ele tem acesso. A parte à direita é destinada a exibir as informações do sistema.

No Documento de Requisitos, a **RNF-4** diz que “O sistema deve controlar o acesso às funcionalidades”. Pensando nisso, o sistema SAE implementou login e senha para que

os seus usuários realizem o acesso e também trata a questão da sessão expirada. A seguir iremos explicar essas questões.

O login utilizará e-mail e senha. O campo do e-mail possui validação para verificar se o mesmo é válido. O campo da senha aceita qualquer caractere alfanumérico e possui tamanho máximo 15. Caso o e-mail e senha informados não correspondam a nenhum usuário, será redirecionado para uma página de erro de login, conforme a Figura 23.

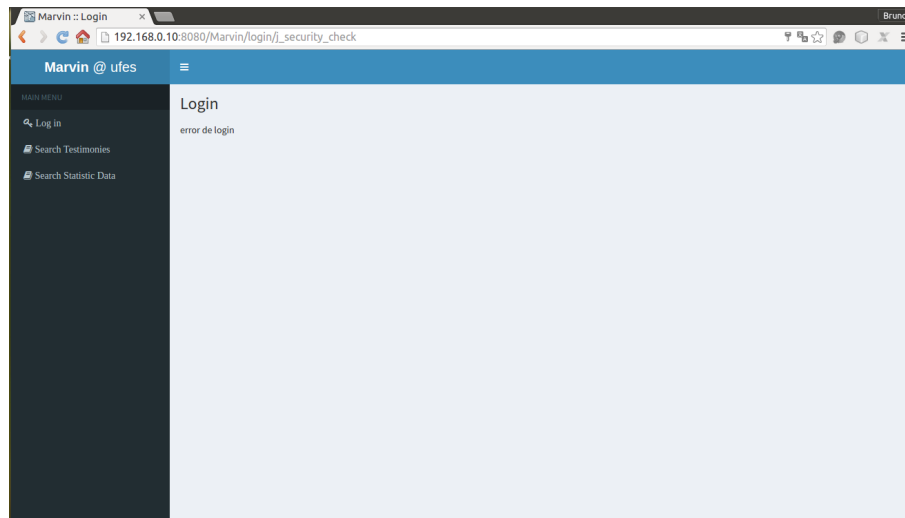


Figura 23 – SAE - Erro Login.

Os menus do sistema irão variar de acordo com o usuário. A Figura 24 exibe a tela inicial do Egresso no menu à esquerda aparece apenas as funcionalidades que o egresso tem acesso.

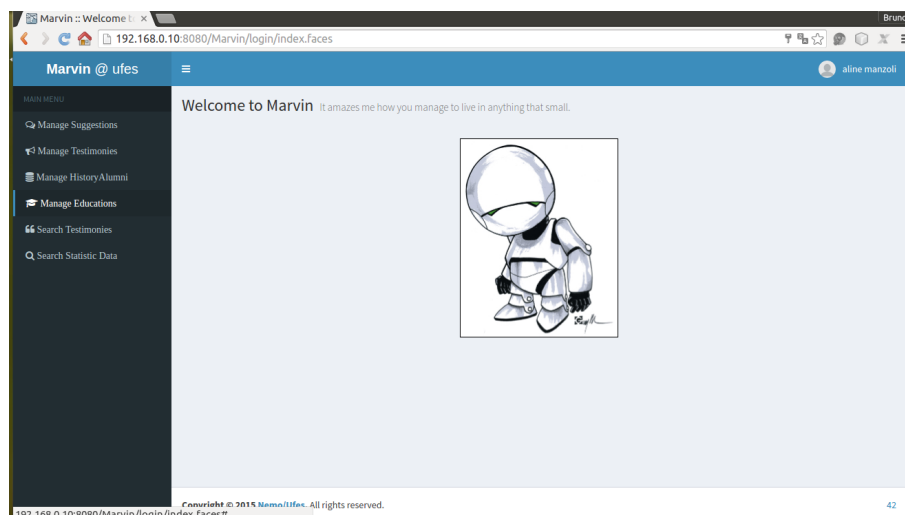


Figura 24 – SAE - Tela inicial Egresso.

A Figura 25 exibe a tela inicial do administrador após realizar o login em um *smartphone*. Como o Marvin utiliza um layout responsivo, haverá diferenças com a visualização em computador. Podemos notar na figura mais à esquerda que a parte de menus

ficar oculta e, para visualizar o menu, como mostra a figura mais à direita, é preciso clicar no ícone ao lado do nome de usuário.

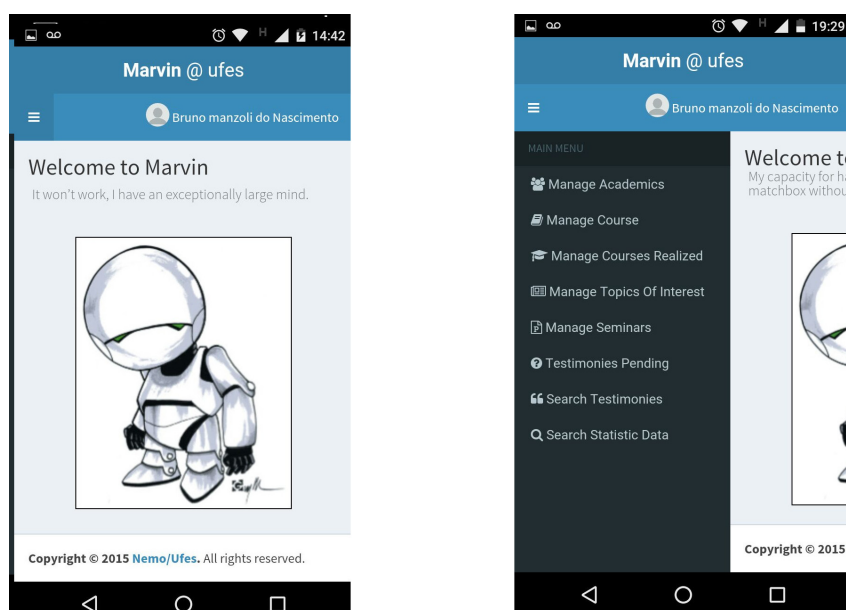


Figura 25 – SAE - Tela Mobile de Administrador.

As funcionalidades que são relacionadas a cadastro, onde se pode criar um novo item, visualizar um item, alterar um item existente ou excluir um item, têm telas que seguem um padrão sendo uma para listar os itens cadastrados e outra para visualizar ou alterar os dados de um item. A Figura 26 exibe a tela que lista os cursos cadastrados no sistema.

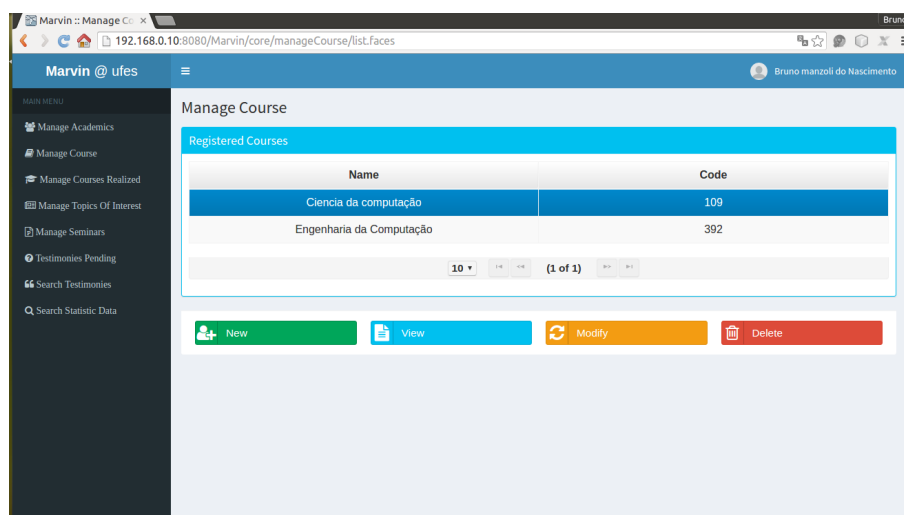


Figura 26 – SAE - Lista de Curso.

Quando o usuário clicar nos botões **New** (novo) ou **View** (visualizar) ou **Modify** (alterar), será redirecionado para a pagina de cadastro de um item conforme a Figura 27.

Marvin @ ufes

Course's data

Course Registration Form

Name\*: Ciencia da computação

Code\*: 109

Coordinator: Bruno manzoli do Nascimento

Save Back

Figura 27 – SAE - Tela cadastro de Curso.

Para exclusão de um item, após clicar no botão “delete” será exibido um novo painel para confirmação da exclusão como podemos ver na Figura 28, na parte inferior da tela informando o item que será excluída e os botões de confirmar a exclusão e de cancelar a exclusão.

Marvin @ ufes

Manage Course

Registered Courses

Name	Code
Ciencia da computação	109
Engenharia da Computação	392

10 (1 of 1)

New View Modify Delete

Table with the courses to be excluded

Items to delete:

1. Ciencia da computação

Cancel Confirm deletion

Figura 28 – SAE - Tela exclusão de Curso.

A Figura 29 apresenta a telas de listagem de itens e de dados de um item para quem está utilizando um dispositivo móvel, podemos notar algumas diferenças em relação a visualização para *desktop* como o menu que não aparece e principalmente a disposição dos botões onde eles passam de uma formação horizontal na versão *desktop* para uma vertical na versão *mobile*. Para facilitar e orientar o usuário foram utilizados cores para identificar as funções dos botões como, por exemplo, o botão de criar um novo item tem a cor verde, o de visualizar os dados de um item tem a cor azul.

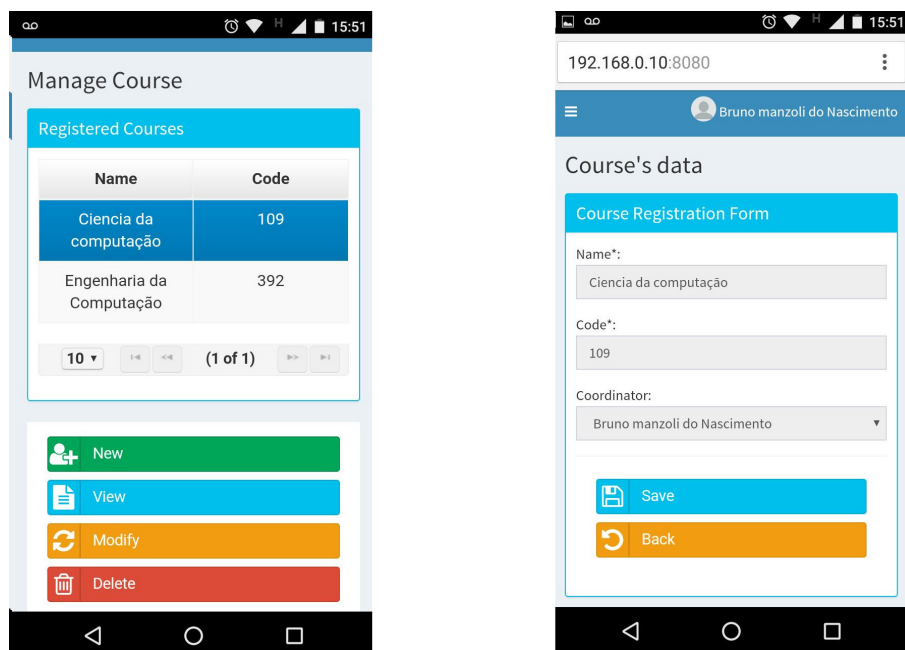


Figura 29 – SAE - Telas mobile.

As figuras 30 e 31 mostram as telas relacionadas ao caso de uso consultar dados estatísticos, onde se tem a tela com as opções de gráficos como por exemplo o de nível salarial, área de atuação, área de formação, entre outros. As figuras também mostram as telas com os gráficos em forma de pizza com as porcentagem que representa cada item da sua legenda.

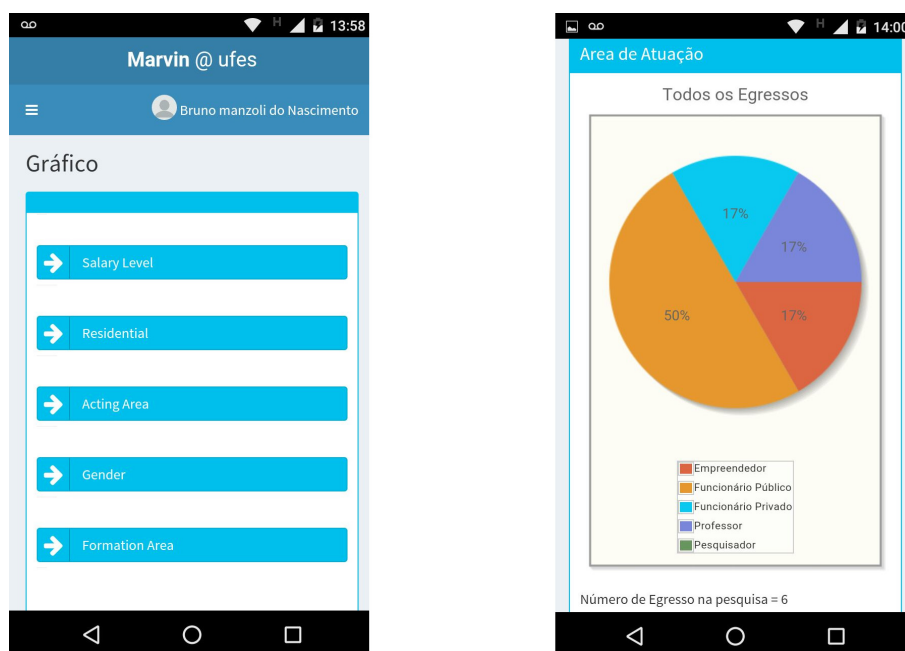


Figura 30 – SAE - Telas mobile Gráfico.

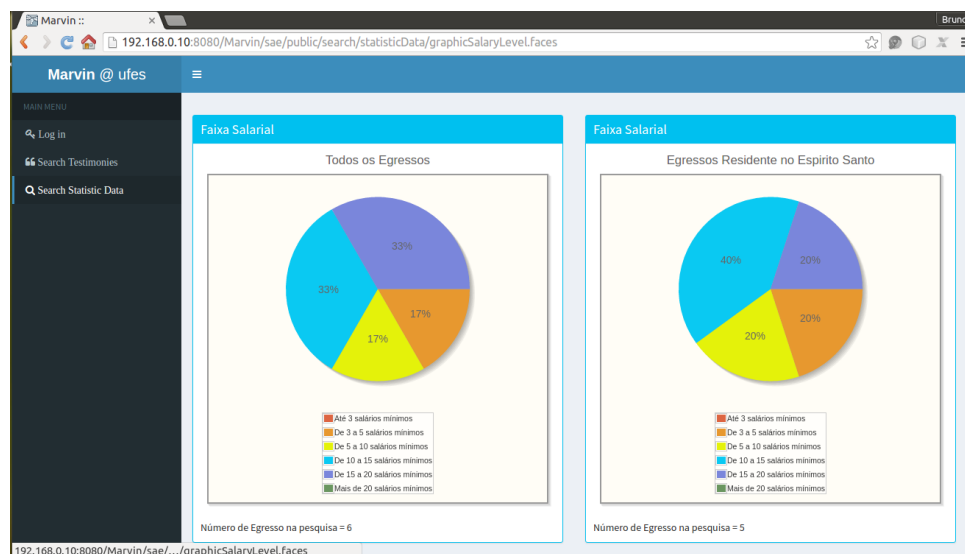


Figura 31 – SAE - Tela Consulta Faixa Salarial.

Para se ter um depoimento divulgado no site antes ele tem que passar por uma avaliação de um administrador, a Figura 32 mostra os depoimento que estão esperando por uma avaliação, para avaliar o administrador seleciona o depoimento e clica no botão **View** (visualizar).

**Testimonies Pending**

Table with the Testimony registered

Course	Send Date	Status
Ciencia da computação	23/05/2016	Pendente
Ciencia da computação	23/05/2016	Pendente

[View](#)

Figura 32 – SAE - Tela Depoimentos á serem analisados.

Depois de clicar no botão *view* o administrador será redirecionado para a pagina de avaliação do depoimento como mostra a Figura 33, onde se tem os dados do depoimento no modo de leitura apenas. Assim, o administrador vai apenas clicar no botão **Approve** (aprovar) ou no botão **Disapprove** (desaprovar) para aprovar ou desaprovar o depoimento respectivamente.



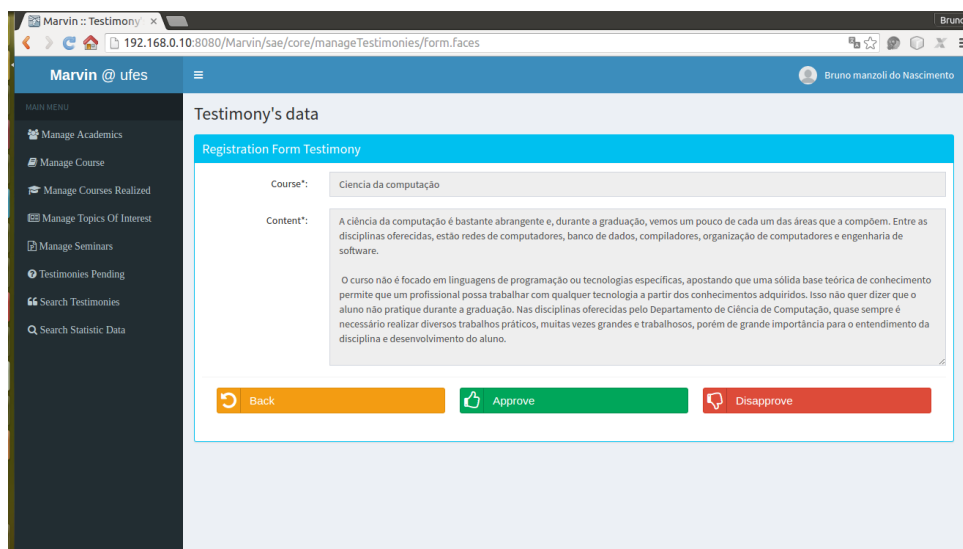


Figura 33 – SAE - Tela Análise de Depoimentos.

A Figura 34 mostra como serão divulgados os depoimentos aprovados pelos administradores. Podemos observar nesta figura que são mostrados o nome do egresso, o seu depoimento e a data que ele o enviou. Para os casos onde o egresso não queira se identificado como o autor do depoimento, no lugar do nome aparecerá que o depoimento é anônimo.

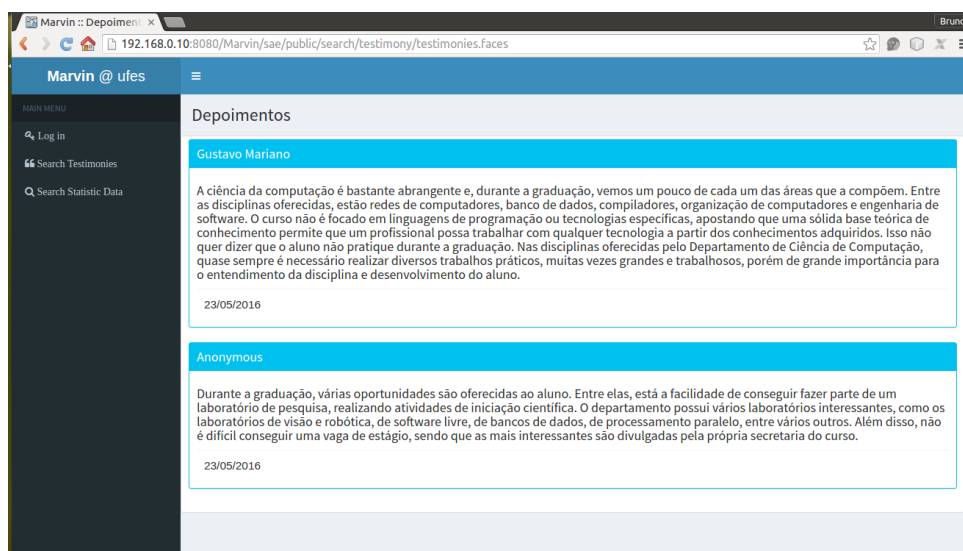


Figura 34 – SAE - Tela Consulta Depoimentos.

## 5 Considerações Finais

Este capítulo apresenta as conclusões do trabalho realizado, mostrando suas contribuições. Por fim, são apresentadas suas limitações e perspectivas de trabalhos futuros.

### 5.1 Conclusões

Com a necessidade de acompanhamento dos alunos egressos do DI/Ufes, objetivando estimular alunos do ensino médio pela área da informática, viu-se a oportunidade de desenvolver um sistema web para atender esta necessidade. Além disso, como muitos estudantes do DI/Ufes desenvolvem ferramentas como parte de seu projeto final de graduação, viu-se a necessidade de integrar futuras ferramentas de forma a serem realmente utilizadas.

Os objetivos elencados no Capítulo 1 foram alcançados, de forma que toda a documentação indicada pela Engenharia de Software foi feita. Primeiramente os requisitos foram levantados e analisados, gerando os Documento de Especificação de Requisitos contendo os requisitos funcionais e não funcionais, a descrição do propósito do sistema e do minimundo, a definição dos atores, casos de uso, diagrama de estado e diagrama de classe. Após esta fase deu início ao desenvolvimento do Documento de Projeto contendo os Atributos de Qualidade e Táticas, os modelos propostos pelo FrameWeb e a arquitetura de software para o SAE.

Dentre as dificuldades encontradas para o desenvolvimento desse trabalho podemos destacar o estudo e entendimento das tecnologias Java EE tais como JAAS, CDI, JPA. Assim, observou-se a necessidade de realizar pesquisas de exemplos e tutoriais e a leitura da documentação destes frameworks. Outras dificuldades encontradas foram: assimilar os conceitos do FrameWeb tendo em vista o curto período de tempo para o desenvolvimento do projeto e escrita da monografia; Implementar o SAE como um módulo de um sistema que visa a integração de outros sistemas a serem desenvolvidos nos projetos de graduação, visto que a base desse sistema integrador (Marvin) estava ainda em construção.

Durante a fase de desenvolvimento do projeto e da implementação do mesmo, foi possível praticar e avaliar o método FrameWeb, verificando que ele auxiliou no desenvolvimento com os modelos de projeto e do perfil UML propostos pelo método por eles aproximarem o modelo de projeto arquitetural da implementação do sistema, reduzindo assim o tempo gasto com o desenvolvimento. Por outro lado, sentiu-se a falta de uma forma de especificar um modelo de segurança, mostrando quais classes seriam protegidas e quais usuários teriam acesso a elas. Uma sugestão para especificar esse modelo de segurança

encontra-se na Figura 35, que utiliza o modelo de aplicação do FrameWeb visto que as classes desse modelo que serão protegidas, foi utilizado cores para especificar quais usuários terão acesso as classes.

Por fim, cabe destacar o grande desafio que foi integrar as diferentes disciplinas realizadas durante o curso de Ciência da Computação, pois elas foram vistas muitas vezes de forma teórica e separadamente uma da outra, mas a experiência adquirida com o desenvolvimento desse trabalho foi enorme e proveitosa, pois foi possível colocar na prática os conceitos aprendidos em sala de aula superando as dificuldades encontradas e, além disso, foi possível adquirir conhecimentos de novas tecnologias que servem para resolver os problemas que podemos encontrar no dia-a-dia.

## 5.2 Limitações e Perspectivas Futuras

No final do desenvolvimento de um software, tipicamente novas necessidades são identificadas. A manutenção e a evolução de software devem ser um trabalho constante, de forma que o ciclo de vida não finalize na homologação, mas permaneça ao longo de toda a vida do software.

A partir dos resultados alcançados, algumas limitações podem ser observadas, o que dá margem para a realização de trabalhos futuros, sendo assim alguns trabalhos surgirão a partir deste. Essas limitações são apresentadas nos itens abaixo.

- Adicionar ao método FrameWeb um modelo onde seja possível modelar os controle de segurança do sistema.
- Ampliar o escopo do sistema do DI/Ufes para todos os departamentos da Ufes, assim todos os cursos poderiam ser incluídos.

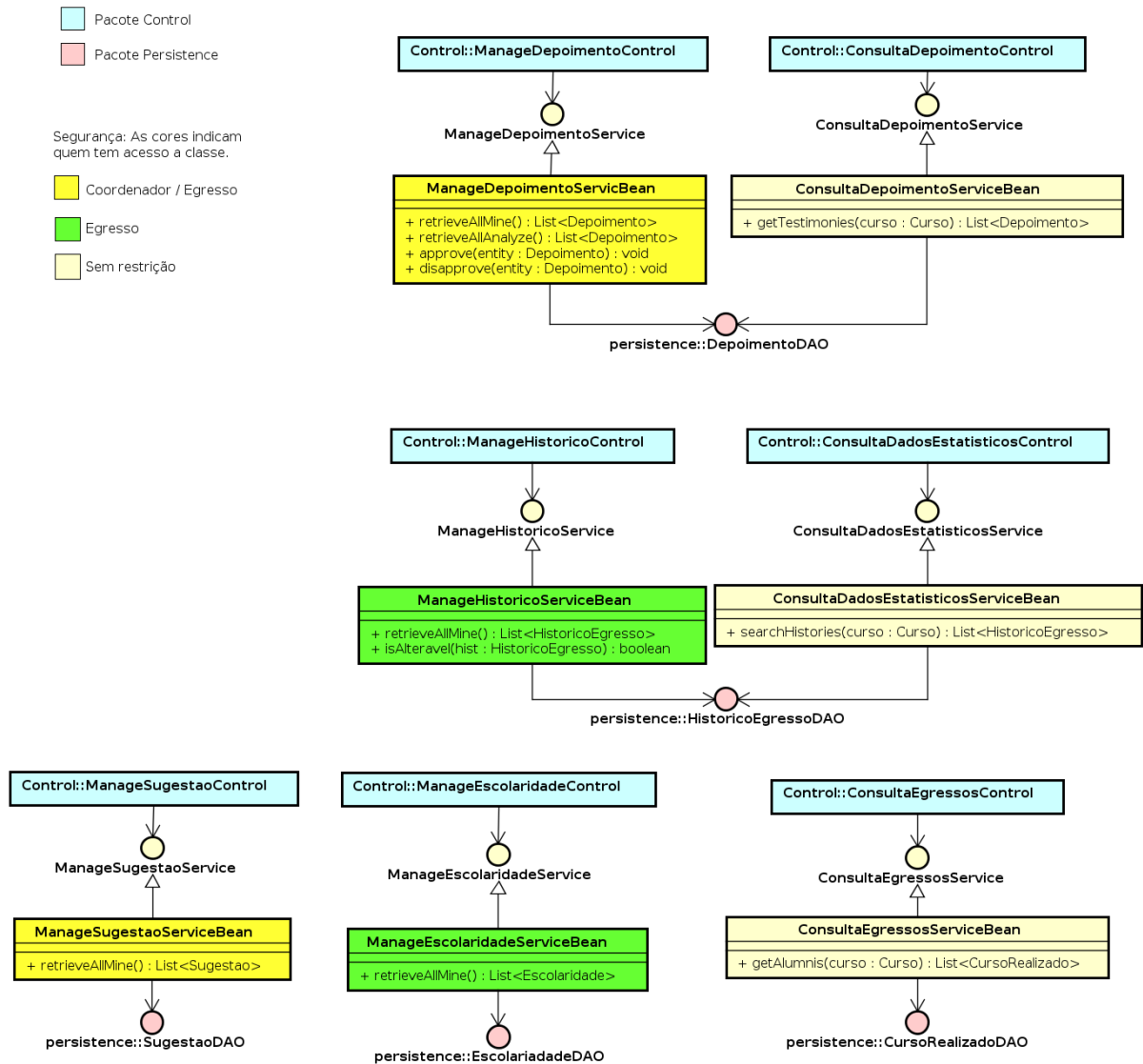


Figura 35 – FrameWeb Modelo de Aplicação Sugerido

# Referências

- ANDERSSON, J. et al. Modeling dimensions of self-adaptive software systems. In: *Software engineering for self-adaptive systems*. [S.l.]: Springer, 2009. p. 27–47. Citado na página 12.
- BRUN, Y. et al. Engineering self-adaptive systems through feedback loops. In: *Software engineering for self-adaptive systems*. [S.l.]: Springer, 2009. p. 48–70. Citado 2 vezes nas páginas 12 e 19.
- DALPIAZ, F. et al. Runtime goal models: Keynote. In: IEEE. *Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on*. [S.l.], 2013. p. 1–11. Citado na página 19.
- DARDENNE, A.; FICKAS, S.; LAMSWEERDE, A. van. Goal-directed concept acquisition in requirements elicitation. In: IEEE COMPUTER SOCIETY PRESS. *Proceedings of the 6th international workshop on Software specification and design*. [S.l.], 1991. p. 14–21. Citado na página 18.
- DARDENNE, A.; LAMSWEERDE, A. V.; FICKAS, S. Goal-directed requirements acquisition. *Science of computer programming*, Elsevier, v. 20, n. 1-2, p. 3–50, 1993. Citado 2 vezes nas páginas 17 e 18.
- FALBO, R. A. *Engenharia de Software*. [s.n.], 2014. 141 p. Disponível em: [http://www.inf.ufes.br/~falbo/files/ES/Notas\\_Aula\\_Engenharia\\_Software.pdf](http://www.inf.ufes.br/~falbo/files/ES/Notas_Aula_Engenharia_Software.pdf). Citado na página 16.
- FALBO, R. A. *Engenharia de Software*. [s.n.], 2017. 71 p. Disponível em: [http://www.inf.ufes.br/~falbo/files/ER/Notas\\_Aula\\_Engenharia\\_Requisitos.pdf](http://www.inf.ufes.br/~falbo/files/ER/Notas_Aula_Engenharia_Requisitos.pdf). Citado na página 16.
- JURETA, I.; MYLOPOULOS, J.; FAULKNER, S. Revisiting the core ontology and problem in requirements engineering. In: IEEE. *International Requirements Engineering, 2008. RE'08. 16th IEEE*. [S.l.], 2008. p. 71–80. Citado na página 17.
- KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. *Computer*, IEEE, v. 36, n. 1, p. 41–50, 2003. Citado na página 12.
- LAMSWEERDE, A. V. Goal-oriented requirements engineering: A guided tour. In: IEEE. *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. [S.l.], 2001. p. 249–262. Citado na página 17.
- LAMSWEERDE, A. V.; DARIMONT, R.; LETIER, E. Managing conflicts in goal-driven requirements engineering. *IEEE transactions on Software engineering*, IEEE, v. 24, n. 11, p. 908–926, 1998. Citado na página 17.
- PFLEEGER, S. L. *Engenharia de software: teoria e prática*. [S.l.]: Prentice Hall, 2004. Citado na página 16.
- ROSS, D. T.; SCHOMAN, K. E. Structured analysis for requirements definition. *IEEE transactions on Software Engineering*, IEEE, n. 1, p. 6–15, 1977. Citado na página 16.

- SOUZA, V. E. S. *Requirements-based software system adaptation*. Tese (Doutorado) — University of Trento, 2012. Citado 4 vezes nas páginas 7, 12, 18 e 21.
- SOUZA, V. E. S.; LAPOUCHNIAN, A.; MYLOPOULOS, J. (requirement) evolution requirements for adaptive systems. In: IEEE PRESS. *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. [S.l.], 2012. p. 155–164. Citado 4 vezes nas páginas 12, 18, 19 e 20.
- SOUZA, V. E. S. et al. Awareness requirements. In: *Software Engineering for Self-Adaptive Systems II*. [S.l.]: Springer, 2013. p. 133–161. Citado 3 vezes nas páginas 18, 19 e 20.
- YU, E. et al. 1 social modeling for requirements engineering: An introduction. *Social Modeling for Requirements Engineering*, MIT Press, p. 3–10, 2011. Citado na página 18.

## Apêndices