
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-
63 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 Requirement* ou *AwReq*), que especificam os indicadores de convergência que o sistema deve atingir para que determinado objetivo seja cumprido; - Requisitos de Evolução (*Evolution Requirements* ou *EvoReq*), 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

# Lista de tabelas

# 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

# 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 $i^\star$, usado pelo framework para verificar se os requisitos do modelo de domínio específico estão sendo atendidos satisfatóriamente. 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 sitemas 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™. Após o levantamento das tecnologias que deveriam ser utilizadas para o desenvolvimento de editores gráficos iniciou-se estudo das tecnologias EMF, Sirius™ e do código fonte do *Zanshin*.

3. *Elaboração do novo Metamodelo*: Nessa etapa o novo metamodelo a ser usado foi elaborado gradativamente a partir dos de 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, inicou-

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*[1] utilizando o template *abnTeX*[2] 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* [3].

## 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 capitulo 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 modulos da mesma.

- **Capítulo 4** – Revisão: é feita validação do metamodelo evoluido 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

[1] LaTeX – http://www.latex-project.org/
[2] abnTeX – http://www.abntex.net.br
[3] 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™, o plugin Sirius™, dentre outros.

## 2.1 Engenharia de Requisitos Orientada a Objetivos

A Egenharia 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 Egenharia 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 elicitadas, além de dizer como o sistema deve ser construído para satisfazê-las (**??**). 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 (**??**). De acordo com (**??**), (**??**) e (**??**), 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 refererir-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; 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.

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.

### 2.1.1 Análise e Especificação de Requisitos

O foco está no levantamento, compreensão e especificação dos requisitos que o sistema a ser desenvolvido tem de tratar. Erros nesta fase são mais dificéis de serem corrigidos quando identificados posteriormente, assim é importante entender o que o cliente deseja.

No que concerne às atividades técnicas, tipicamente o processo de software inicia-se com o Levantamento de Requisitos, quando os requisitos do sistema a ser desenvolvido são preliminarmente capturados e organizados. Uma vez capturados, os requisitos devem ser modelados, avaliados e documentados. Uma parte essencial dessa fase é a elaboração de modelos descrevendo o que o software tem de fazer (e não como fazê-lo), dita Modelagem Conceitual. Até este momento, a ênfase está sobre o domínio do problema e não se deve pensar na solução técnica, computacional a ser adotada (FALBO, 2017).

Dada a importância dos requisitos para o sucesso de um projeto, atividades de controle da qualidade devem ser realizadas para verificar, validar e garantir a qualidade dos requisitos, uma vez que os custos serão bem maiores se defeitos em requisitos forem identificados tardiamente (FALBO, 2017).

Neste projeto foram utilizadas as técnicas de levantamento e especificação de requisitos aprendidas ao longo do curso, como descrição de minimundo, levantamento de requisitos funcionais e não-funcionais, modelagem de casos de uso e modelagem conceitual estrutural. Nos parágrafos que se seguem, descrevemos brevemente estas técnicas.

A descrição do minimundo apresenta, em um texto corrido, uma visão geral do domínio do problema a ser resolvido e dos processos de negócio apoiados, bem como as principais ideias do cliente sobre o sistema a ser desenvolvido.

Já os requisitos funcionais, são declarações de serviços que o sistema deve prover, descrevendo o que o sistema deve fazer, podendo descrever, ainda, como o sistema deve reagir a entradas específicas, como o sistema deve se comportar em situações específicas e o que o sistema não deve fazer (**??**).

Assim como os requisitos funcionais precisam ser especificados em detalhes, o mesmo acontece com os requisitos não-funcionais. Para os atributos de qualidade considerados prioritários, o analista deve trabalhar no sentido de especificá-los de modo que eles se tornem mensuráveis e, por conseguinte, testáveis. Eles descrevem restrições sobre os serviços ou funções oferecidos pelo sistema (**??**).

O modelo de casos de uso é um modelo comportamental, mostrando as funções do sistema, mas de maneira estática. Ele é composto de dois tipos principais de artefatos: os diagramas de casos de uso e as descrições de casos de uso. Um diagrama de casos de uso é um diagrama bastante simples, que descreve o sistema, seu ambiente e como sistema e ambiente estão relacionados. As descrições dos casos de uso descrevem o passo a passo para a realização dos casos de uso e são essencialmente textuais (FALBO, 2017).

Tomando por base casos de uso e suas descrições, é possível passar à modelagem conceitual estrutural, quando os conceitos e relacionamentos envolvidos no domínio são capturados em um conjunto de diagramas de classes. Neste momento é importante definir, também, o significado dos conceitos e de suas propriedades, bem como restrições sobre eles. Essas definições são documentadas em um dicionário de dados do projeto.

Um diagrama de classes exibe um conjunto de classes e seus relacionamentos. Diagramas de classes proveem uma visão estática da estrutura de um sistema e, portanto, são usados na modelagem conceitual estrutural. Para tornar os modelos conceituais mais simples, de modo a facilitar a comunicação com clientes e usuários, tipos de dados de atributos podem ser omitidos do diagrama de classes. Restrições de integridade são regras de negócio e poderiam ser lançadas no Documento de Requisitos. Contudo, como elas são importantes para a compreensão e eliminação de ambiguidades do modelo conceitual, é útil descrevê-las no próprio modelo conceitual (FALBO, 2017).

### 2.1.2 Projeto

Com os requisitos pelo menos parcialmente capturados e especificados na forma de modelos, pode-se começar a trabalhar no domínio da solução. Muitas soluções são possíveis para o mesmo conjunto de requisitos e elas são intrinsecamente ligadas a uma dada plataforma de implementação (linguagem de programação, mecanismo de persistência a ser adotado etc.). A fase de projeto tem por objetivo definir e especificar uma solução a ser implementada. É uma fase de tomada de decisão, tendo em vista que muitas soluções são possíveis (FALBO, 2017).

A fase de Projeto é responsável por incorporar requisitos tecnológicos aos requisitos essenciais do sistema e, portanto, requer que a plataforma de implementação seja conhecida. Basicamente, envolve duas grandes etapas: projeto da arquitetura do sistema e o projeto detalhado. O objetivo da primeira etapa é definir a arquitetura geral do software, tendo por

base o modelo construído na fase de análise de requisitos. Essa arquitetura deve descrever a estrutura de nível mais alto da aplicação e identificar seus principais componentes. O propósito do projeto detalhado é detalhar o projeto do software para cada componente identificado na etapa anterior. Os componentes de software devem ser sucessivamente refinados em níveis maiores de detalhamento, até que possam ser codificados e testados (FALBO, 2014).

## 2.2 FrameWeb

FrameWeb (*Framework-based Design Method for Web Engineering*) é um método de projeto para construção de sistemas de informação Web (*Web Information Systems – WISs*) baseados em *frameworks*. FrameWeb é baseado em metodologias e linguagens de modelagem bastante difundidas na área de Engenharia de Software sem, no entanto, impor nenhum processo de desenvolvimento específico (**??**).

A proposta deste método foi motivada por:

- O uso de *frameworks* ou arquiteturas baseadas em *containers* similares a eles se tornou o padrão de fato para o desenvolvimento de aplicações distribuídas, em especial os baseados na plataforma Web;
- O uso de métodos que se adequam diretamente à arquitetura de software adotada promove uma agilidade maior ao processo, característica que é bastante desejada na maioria dos projetos Web (**??**).

Em linhas gerais, FrameWeb assume que determinados tipos de *frameworks* serão utilizados durante a construção da aplicação, define uma arquitetura básica para o WIS e propõe modelos de projeto que se aproximam da implementação do sistema usando esses *frameworks*.

Devido à popularidade dos *frameworks* no desenvolvimento web, o método FrameWeb propõe a utilização de modelos específicos direcionados à arquitetura baseada em *containers*, que durante a fase de projeto auxiliam o modelador na tarefa de lidar com a complexidade por trás da aplicação WIS. Porém, não só novos *frameworks*, tecnologias e plataformas surgiram, como continuarão surgindo e evoluindo. Por isso, é importante que os métodos também evoluam não só no que diz respeito a novas versões, mas também para permitir que as novidades possam ser incorporadas de forma simples e efetiva, como propõe **??**).

Sendo um método para a fase de projeto, não prescreve um processo de software completo. No entanto, sugere o uso de um processo de desenvolvimento que contemple as fases apresentadas na Figura 1. Para uma melhor utilização de FrameWeb, espera-se

que sejam construídos diagramas de casos de uso e de classes de domínio (ou similares) durante as fases de Requisitos e Análise.

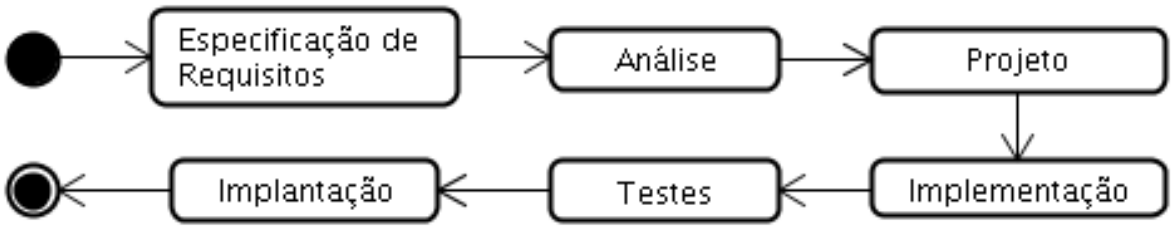


Figura 1 – Processo de Desenvolvimento de Software (**??**).

A fase de Projeto concentra as propostas principais do método: (i) definição de uma arquitetura padrão que divide o sistema em camadas, de modo a se integrar bem com os *frameworks* utilizados; (ii) proposta de um conjunto de modelos de projeto que trazem conceitos utilizados pelos *frameworks* para esta fase do processo por meio da criação de um perfil UML que faz com que os diagramas fiquem mais próximos da implementação.

O FrameWeb define extensões leves (*lightweight extensions*) ao meta-modelo da UML para representar componentes típicos da plataforma Web e dos *frameworks* utilizados, criando um perfil UML que é utilizado para a construção de diagramas de quatro tipos:

- **Modelo de Domínio:** é um diagrama de classes da UML que representa os objetos de domínio do problema e seu mapeamento para a persistência em banco de dados relacional. Os passos para sua construção são:
  - A partir do modelo de classes construído na fase de análise de requisitos, adequar o modelo à plataforma de implementação escolhida, indicando os tipos de dados de cada atributo, promovendo a classes atributos que devam ser promovidos, definindo a navegabilidade das associações etc.;
  - Adicionar os mapeamentos de persistência.
- **Modelo de Aplicação:** é um diagrama de classes da UML que representa as classes de serviço, que são responsáveis pela codificação dos casos de uso, e suas dependências. Os passos para a construção do Modelo de Aplicação são:
  - Analisar os casos de uso modelados durante a Especificação de Requisitos, definir a granularidade das classes de serviço e criá-las. Utilizar, preferencialmente, nomes que as relacionem com os casos de uso ou cenários que representam;
  - Adicionar às classes/interfaces os métodos que implementam a lógica de negócio, com atenção ao nome escolhido (preferencialmente relacionar o método ao cenário que implementa), aos parâmetros de entrada e ao retorno (observar a descrição do caso de uso);

  - Por meio de uma leitura da descrição dos casos de uso, identificar quais DAOs (*Data Access Object*) (**??**) são necessários para cada classe de aplicação e modelar as associações;
  - Voltar ao modelo de navegação (se já foi construído), identificar quais classes controladoras dependem de quais classes de serviço e modelar as associações.

- **Modelo de Navegação:** é um diagrama de classe da UML que representa os diferentes componentes que formam a camada de Lógica de Apresentação, como páginas Web, formulários HTML e classes controladoras. Os passos para a construção dos Modelos de Navegação são:

  - Analisar os casos de uso modelados durante a Especificação de Requisitos, definir a granularidade das classes controladoras e criá-las, definindo seus métodos. Utilizar, preferencialmente, nomes que as relacionem com os casos de uso ou cenários que representam;
  - Identificar como os dados serão submetidos pelos clientes para criar as páginas, modelos e formulários, adicionando atributos à classe controladora;
  - Identificar quais resultados são possíveis a partir dos dados de entrada, para criar as páginas e modelos de resultado, também adicionando atributos à classe controladora.
  - Analisar se o modelo ficou muito complexo e considerar dividi-lo em dois ou mais diagramas.

- **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. Os passos para a construção desse modelo são:

  - Criar as interfaces e implementações concretas dos DAOs base;
  - Definir quais classes de domínio precisam de lógica de acesso a dados e, portanto, precisam de um DAO;
  - Para cada classe que precisa de um DAO, avaliar a necessidade de consultas específicas ao banco de dados, adicionando-as como operações nos respectivos DAOs.

  O Modelo de Persistência apresenta, para cada classe de domínio que necessita de lógica de acesso a dados, uma interface e uma classe concreta DAO que implementa a interface. A interface, que é única, define os métodos de persistência existentes para aquela classe, a serem implementados por uma ou mais classes concretas, uma para cada tecnologia de persistência diferente (ex.: um DAO para o JPA, outro para o *framework* OJB, etc.).

Esses quatro modelos serão descritos na Seção 4.3 no contexto do SAE, sistema desenvolvido neste trabalho.

## 2.3 Desenvolvimento Web

As aplicações Web de hoje em dia já possuem regras de negócio bastante complicadas. Codificar essas regras já representa um grande trabalho. Além dessas regras, conhecidas como requisitos funcionais de uma aplicação, existem outros requisitos (não-funcionais) que precisam ser atingidos através da infraestrutura do sistema: persistência em banco de dados, transação, acesso remoto, *web services*, gerenciamento de *threads*, gerenciamento de conexões HTTP, *cache* de objetos, gerenciamento da sessão web, balanceamento de carga, entre outros (**??**).

A Java EE (*Java Platform, Enterprise Edition*) é uma plataforma padrão para desenvolver aplicações Java de grande porte e/ou para a Internet, que inclui bibliotecas e funcionalidades para implementar software Java distribuído, baseado em componentes modulares que executam em servidores de aplicações e que suportam escalabilidade, segurança, integridade e outros requisitos de aplicações corporativas ou de grande porte (**??**).

A plataforma Java EE possui uma série de especificações (tecnologias) com objetivos distintos, por isso é considerada uma plataforma guarda-chuva. Entre as especificações da Java EE, as mais conhecidas são:

- **Servlets**: são componentes Java executados no servidor para gerar conteúdo dinâmico para a web, como HTML e XML.
- **JSF (*JavaServer Faces*)**: é um framework web baseado em Java que tem como objetivo simplificar o desenvolvimento de interfaces de sistemas para a web, através de um modelo de componentes reutilizáveis.
- **JPA (*Java Persistence API*)**: é uma API padrão do Java para persistência de dados, baseada no conceito de mapeamento objeto-relacional. Essa tecnologia traz alta produtividade para o desenvolvimento de sistemas que necessitam de integração com banco de dados.
- **EJB (*Enterprise Java Beans*)**: são componentes que executam em servidores de aplicação e possuem como principais objetivos fornecer facilidade e produtividade no desenvolvimento de componentes distribuídos, transacionados, seguros e portáveis.
- **JAAS (*Java Autenthication and Authorization Service*)**: API padrão do Java para segurança.
- **CDI (*Contexts and Dependency Injection*)**: é a especificação da Java EE que trabalha com injeção de dependências.

### 2.3.1 *Servlets*

O nome *servlet* vem da ideia de um pequeno servidor (servidorzinho, em inglês) cujo objetivo é receber chamadas HTTP, processá-las e devolver uma resposta ao cliente(**??**).

Uma primeira ideia da *servlet* seria que cada uma delas é responsável por uma página, sendo que ela lê dados da requisição do cliente e responde com outros dados (uma página HTML, uma imagem GIF etc).

No mundo Java, os servidores web são chamados de *Servlet Container*, pois implementam a especificação de *Servlet*. O servidor converte a requisição em um objeto do tipo *HttpServletRequest* . Este objeto é então passado aos componentes web, que podem executar qualquer código Java para que possa ser gerado um conteúdo dinâmico. Em seguida, o componente web devolve um objeto *HttpServletResponse* , que representa a resposta ao cliente. Este objeto é utilizado para que o conteúdo gerado seja enviado ao navegador do usuário (**??**).

### 2.3.2 EJB

EJBs simplificam o desenvolvimento de aplicações grandes e distribuídas. Primeiro, porque o *container* EJB fornece serviços de nível de sistema a elas. Assim sendo o desenvolvedor pode se concentrar em resolver problemas do negócio. O *container* EJB é responsável por serviços como gestão de transações e autorizações de segurança. Segundo, porque são os EJBs que contêm a lógica de negócios, não os clientes. Assim sendo, o desenvolvedor da aplicação cliente pode se concentrar na apresentação, não tendo que implementar regras de negócio ou bancos de dados de acesso. Como resultado, clientes tornam-se mais leves, executáveis em máquinas menos poderosas. Terceiro, porque EJBs são componentes portáteis, podendo ser reutilizados em outros aplicativos (**??**).

***Session Beans*** encapsulam lógica de negócio que pode ser invocada programaticamente por um cliente de maneira local, remota ou via web service. Para acessar uma aplicação armazenada em um servidor, o cliente invoca os métodos do *Session Bean*. Sobre este tipo de EJB, trazemos as seguintes informações:

*Stateful Session Beans:* em um EJB *stateful*, suas variáveis de instância representam o estado de uma sessão única aberta entre o cliente e o EJB. Devido ao fato do cliente interagir com o EJB, esse estado é frequentemente denominado estado conversacional. O estado é mantido enquanto durar a sessão cliente/EJB.

*Stateless Session Beans:* esses EJBs não mantêm um estado conversacional com o cliente. Quando o cliente invoca métodos de um EJB *stateless*, as variáveis de instância mantêm um estado específico apenas enquanto durar a execução do método. Quando esta termina, o estado não é mantido. Exceto durante a invocação de métodos, todas as instâncias de EJBs *stateless* são equivalentes, permitindo alocar uma instância para

qualquer cliente.

*Singleton Session Beans:* são instanciados apenas uma vez por aplicação e existem durante o ciclo de vida da mesma. São projetados para circunstâncias nas quais uma única instância do EJB é compartilhada e concorrentemente acessada por clientes. Oferecem a mesma funcionalidade dos EJBs *stateful*, a menos da instância única. O estado é mantido entre invocações de clientes, mas não quando ocorrem quedas do servidor.

*Acesso local:* o cliente deve estar na mesma aplicação do EJB. Pode ser um componente web ou outro EJB. `@Local` deve ser colocado no cabeçalho da interface do EJB local.

*Acesso remoto:* pode rodar em uma máquina e JVM diferentes. Pode ser um componente web, uma aplicação cliente ou outro EJB. `@Remote` deve ser colocado no cabeçalho da interface do EJB remote.

***Message-Driven Beans*** são EJBs que permitem a aplicações Java EE processar mensagens assincronamente. Agem de maneira semelhante a um *listener* (monitor) de eventos, mas ao invés de eventos, recebe mensagens provenientes de aplicações, outro EJB ou componentes web. Eles são acessados através de um serviço de mensagens (JMS), enviando mensagens ao destinatário (*MessageListener*). São executados mediante a recepção de uma mensagem vinda do cliente, assincronamente e *stateless*. Quando a mensagem chega, o *container* chama o método `onmessage` do *Message-Driven Bean*, que por sua vez chama métodos auxiliares para processá-la. Tudo isso ocorre em um contexto transacional. Eles não são acessados via interfaces, como *Session Beans*, ou seja, possuem apenas uma classe *bean* (**??**).

### 2.3.3 JSF

*JavaServer Faces*, também conhecido como JSF, é uma tecnologia para desenvolvimento web que utiliza um modelo de interfaces gráficas baseado em eventos. Esta tecnologia foi definida pelo JCP (*Java Community Process*)[1], o que a torna um padrão de desenvolvimento e facilita o trabalho dos fornecedores de ferramentas, ao criarem produtos que valorizem a produtividade no desenvolvimento de interfaces visuais (**??**).

JSF é baseado no padrão de projeto MVC (*Model-View-Controller*)[2], o que torna o desenvolvimento de sistemas menos complicado. O padrão MVC separa o sistema em três responsabilidades (modelo, visão e controle), onde o modelo é responsável por representar os objetos de negócio, manter o estado da aplicação e fornecer ao controlador o acesso aos dados. A visualização é responsável pela interface do usuário: ela que define a forma como os dados são apresentados e encaminha as ações do usuário para o controlador. O

[1] JCP – https://www.jcp.org/en/home/index
[2] MVC – https://pt.wikipedia.org/wiki/MVC

controlador é responsável por ligar o modelo e a visualização, interpretando as solicitações do usuário, traduzindo para uma operação no modelo e retornando a visualização adequada à solicitação (**??**).

Em JSF, o controle é feito através de uma *servlet* chamada *Faces Servlet*, opcionalmente configurada por arquivos XML e delegando o controle de requisições específicas aos vários manipuladores de ações e observadores de eventos implementados no sistema. Em resumo, a *Faces Servlet* recebe as requisições dos usuários na web, redireciona para o modelo e envia uma resposta.

O verdadeiro poder de *JavaServer Faces* está em seu modelo de componentes de interface do usuário, que gera alta produtividade aos desenvolvedores, permitindo a construção de interfaces para web usando um conjunto de componentes pré-construídos, ao invés de criar interfaces inteiramente do zero.

Existem vários componentes JSF, desde os mais simples, como um *Output Label*, que apresenta simplesmente um texto, ou um *Data Table*, que representa dados tabulares de uma coleção que pode vir do banco de dados. A API de JSF suporta a extensão e criação de novos componentes, que podem fornecer funcionalidades adicionais. Atualmente, existem diversas organizações que trabalham na criação de componentes personalizados, como exemplo, podemos citar a Oracle (*ADF Faces Rich Client*)[3], IceSoft (*IceFaces*)[4], Red Hat (*RichFaces*)[5], Prime Technology (*PrimeFaces*)[6], etc.

A biblioteca de componentes JSF utilizada neste trabalho foi a PrimeFaces. Ela inclui diversos campos de entrada, botões, tabelas de dados, árvores, gráficos, diálogos, etc (**??**).

### 2.3.4 JPA

Mapeamento objeto relacional (*object-relational mapping, ORM, O/RM ou O/R mapping*) é uma técnica de programação para conversão de dados entre banco de dados relacionais e linguagens de programação orientada a objetos (**??**). Em banco de dados, entidades são representadas por tabelas, que possuem colunas que armazenam propriedades de diversos tipos. Uma tabela pode se associar com outras e criar relacionamentos diversos. Em uma linguagem orientada a objetos, como Java, entidades são classes, e objetos dessas classes representam elementos que existem no mundo real (**??**).

A *Java Persistence API* (JPA) é um *framework* para persistência em Java, que oferece uma API de mapeamento objeto-relacional e soluções para integrar persistência com sistemas corporativos escaláveis (**??**). Entre as vantagens podemos citar:

[3] ADF – http://www.oracle.com/technetwork/developer-tools/adf/overview/index-092391.html
[4] IceFaces – http://www.icesoft.org/java/projects/ICEfaces/overview.jsf
[5] RichFaces – http://richfaces.jboss.org/
[6] PrimeFaces – http://www.primefaces.org/

- Códigos de acesso a banco de dados com *queries SQL* são custosos de se desenvolver. JPA elimina muito do trabalho e deixa você se concentrar na lógica de negócio. JPA trará uma produtividade imensa para você.

- A manutenabilidade de sistemas desenvolvidos com ORM é excelente, pois o mecanismo faz com que menos linhas de código sejam necessárias. Além de facilitar o entendimento, menos linhas de código deixam o sistema mais fácil de ser alterado.

- ORM abstrai sua aplicação do banco de dados e do dialeto SQL. Com JPA, você pode desenvolver um sistema usando um banco de dados e colocá-lo em produção usando diversos outros banco de dados, sem precisar alterar códigos-fontes para adequar sintaxe de queries que só funcionam em SGBDs de determinados fornecedores.

Dentro do JPA, a API de Critérios (*Criteria API*) é baseada no esquema abstrato de entidades persistentes (metamodelos), as suas relações e objetos incorporados. Esta API opera neste esquema abstrato para permitir que desenvolvedores de encontrar, modificar e excluir entidades persistentes invocando operações da JPA. Os metamodelos funcionam em conjunto com a API para modelar classes persistentes da entidade para consultas (**??**).

O padrão DAO (**??**) adiciona uma camada de abstração a mais, separando a lógica de acesso a dados da tecnologia de persistência de maneira que a camada de aplicação não conheça qual *framework* ORM está sendo utilizado, permitindo que o mesmo seja trocado, se necessário. O uso deste padrão também facilita a execução de testes unitários na camada de aplicação. Numa aplicação que utilize a arquitetura MVC, todas as funcionalidades de bancos de dados, tais como obter as conexões, mapear objetos Java para tipos de dados SQL ou executar comandos SQL, devem ser feitas por classes DAO.

A vantagem de usar objetos de acesso a dados é a separação simples e rigorosa entre duas partes importantes de uma aplicação que não devem e não podem conhecer quase que nada uma da outra, e que podem evoluir frequentemente e independentemente. Alterar a lógica de negócio pode esperar apenas a implementação de uma interface, enquanto que modificações na lógica de persistência não alteram a lógica de negócio, desde que a interface entre elas não seja modificada.

### 2.3.5 JAAS

Aplicações Java EE consistem em componentes que podem conter ambos os recursos protegidos e desprotegidos. Muitas vezes, é necessário proteger os recursos para garantir que somente usuários autorizados tenham acesso. Autorização fornece acesso controlado a recursos protegidos (**??**).

*Java Authentication and Authorization Service* (JAAS) é um conjunto de APIs que permitem serviços para autenticar e aplicar controles de acesso sobre os usuários. JAAS é

parte do núcleo da Java SE API e é uma tecnologia de base para mecanismos de segurança Java EE. Os conceitos principais desta API são:

- Autenticação: os meios pelos quais entidades comunicantes, como cliente e servidor, provam um ao outro que estão agindo em nome de identidades específicas que são autorizados para o acesso. Isso garante que os usuários são quem eles dizem que são (**??**).
- Autorização ou controle de acesso: os meios pelos quais as interações com os recursos são limitados a grupos de usuários ou programas com o objetivo de impor a integridade, confidencialidade, disponibilidade ou restrições. Isso garante que os usuários têm permissão para executar operações ou dados de acesso (**??**).

Um *Role* (Papel) é um nome abstrato para a permissão de acesso a um determinado conjunto de recursos em um aplicação. Em uma aplicação que utiliza JAAS, um usuário só terá acesso à um recurso protegido se ele possuir um *role* que permita este acesso, assim um *role* pode ser comparado a uma chave que pode abrir uma fechadura.

Para definir os recursos que serão protegidos e o mecanismo de autenticação na aplicação pode utilizar-se de um descritor de implementação. Em tempo de execução, o servidor Java EE lê este descritor de implementação e age sobre o correspondente aplicativo, em conformidade com as informações estruturais para cada módulo ou componente.

Restrição de segurança é usada para definir os privilégios de acesso a uma coleção de recursos utilizando o seu mapeamento de URL (**??**). Este mapeamento é feito através de uma lista de padrões de URL (a parte de uma URL após o nome do *host* e a porta que deseja restringir) e de operações HTTP que descrevem um conjunto de recursos a serem protegidos. Também é preciso definir quais os usuários terão permissão de executar essas requisições protegidas e como os dados serão protegidos quando transportados entre um cliente e um servidor.

Quando um mecanismo de autenticação for especificado no descritor de implementação, o usuário deve ser autenticado antes que o acesso seja concedido a qualquer recurso limitado por uma restrição de segurança. Pode haver várias restrições de segurança que se aplicam a vários recursos, mas o mesmo método de autenticação será aplicado a todos os recursos limitados em um aplicativo (**??**).

A plataforma Java EE suporta os seguintes mecanismos de autenticação: autenticação básica, autenticação baseada em formulário, autenticação *digest*, autenticação do cliente e autenticação mútua.

A **autenticação baseada em formulário** permite ao desenvolvedor controlar a aparência das telas de autenticação de login e de erro que um navegador HTTP apresenta

para o usuário final. Quando a autenticação baseada em formulário é especificada, ocorrem as seguintes ações conforme a Figura 2.

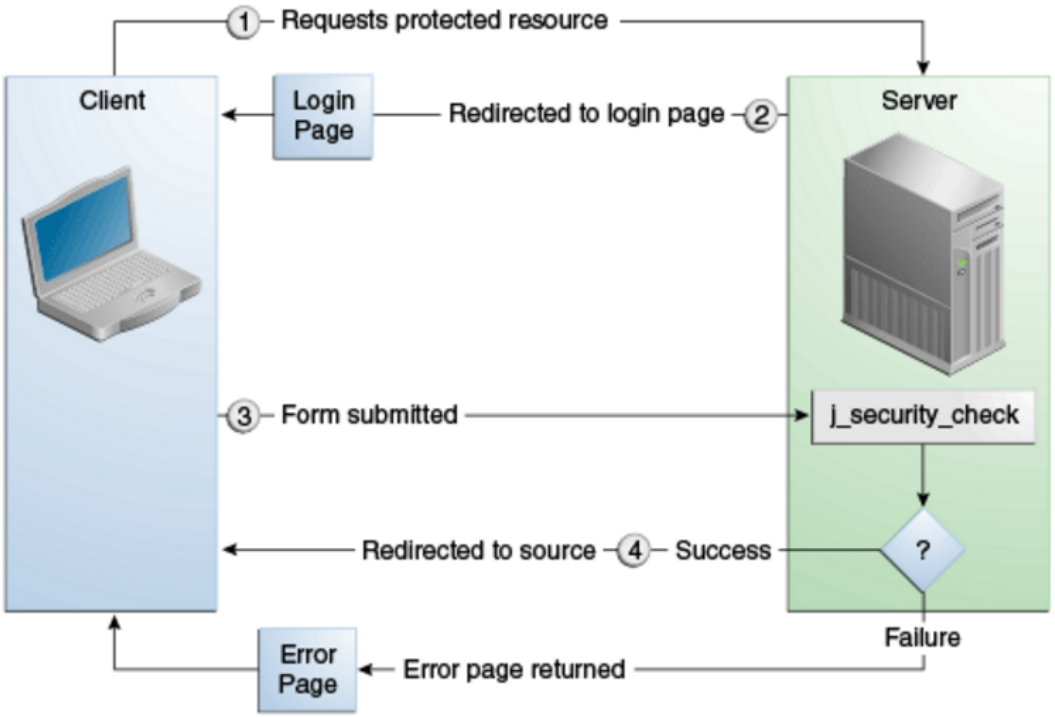


Figura 2 – JAAS - Autenticação baseada em formulário (**??**)

Segurança declarativa permite o desenvolvedor de aplicações especificar quais usuários estão autorizados a acessar determinados métodos dos Enterprise Java Beans (EJBs). Um desenvolvedor que usa segurança declarativa para definir as permissões de métodos e mecanismos de autenticação passa junto ao implementador uma visão de segurança dos EJBs contidos em sua aplicação. Se você não definir uma visão da segurança, o implementador terá que determinar o que cada método de negócio faz para determinar quais usuários estão autorizados a chamar cada método.

A visão da segurança consiste de um conjunto de papéis de segurança, um agrupamento semântico de permissões que um determinado tipo de usuários de uma aplicação deve ter para acessá-lá com êxito.

Permissões podem ser especificadas na classe, nos métodos da classe ou ambos. Permissões podem ser especificadas em um método da classe para substituir o valor das permissões especificado em toda a classe. As seguintes anotações são usadas para especificar permissões:

- `@DeclareRoles`: especifica todas as roles que a aplicação irá utilizar, incluindo roles não designadas especificamente na uma anotação de `@RolesAllowed`.

- `@RolesAllowed`: especifica as roles de segurança autorizadas a métodos de acesso em um aplicação. Esta anotação pode ser especificada numa classe ou em um ou mais métodos. Quando especificada no nível de classe, a anotação aplica-se a todos os métodos na classe. Quando especificada em um método, a anotação aplica-se apenas ao método e substitui quaisquer valores especificados no nível de classe.

- `@DenyAll`: especifica que não há roles autorizadas a métodos de acesso em uma aplicação.

- `@PermitAll`: especifica que um usuário em qualquer papel está autorizado a métodos de acesso em uma aplicação.

### 2.3.6 CDI

Injeção de dependências (*dependency injection* ou DI) é um padrão de desenvolvimento de software usado para manter o baixo acoplamento entre classes do sistema. As dependências de um objeto não são instanciadas programaticamente, mas sim injetadas de alguma forma (**??**).

CDI (*Contexts and Dependency Injection*) é a especificação da Java EE que trabalha com injeção de dependências. Podemos usar CDI para instanciar e injetar objetos de nossa aplicação (**??**). Os serviços mais fundamentais fornecidos pelo CDI são os seguintes (**??**).

- Contextos: este serviço permite ligar o ciclo de vida e interações de componentes *stateful* a contextos de ciclo de vida bem definidos, mas extensível.

- Injeção de dependência: este serviço permite que você injetar componentes em uam aplicação de uma maneira *typesafe* e escolher no momento da implantação, qual implementação de uma interface específica injetar.

Além disso, CDI oferece os seguintes serviços:

- Integração com o *Expression Language* (EL), que permite que qualquer componente a ser usado diretamente dentro de uma página *JavaServer Faces* ou *JavaServer Pages*;

- A capacidade de decorar componentes injetados;

- A capacidade de associar interceptores com componentes usando ligações *typesafe* de interceptadores;

- Um modelo de notificação de eventos;

- Um escopo de conversação web, para além dos três escopos padrão (solicitação, sessão e aplicativo) definido pela especificação *Java Servlet*;
- Uma Interface de provedor de serviços (SPI) completa que permite estruturas de terceiros integrar de forma limpa no ambiente Java EE 7.

O CDI permite que declaremos uma dependência de uma classe do sistema (chamada de bean) a um EJB utilizando a anotação `@EJB` ou a uma classe não-EJB utilizando `@Inject`, ambos sobre o atributo que representa a associação entre o dependente e sua dependência. Provê acesso via *Expression Language* (EL) a beans que utilizarem a anotação `@Named` na definição da classe. Tal anotação permite definir um nome para o componente ou utilizar o nome default: o mesmo nome da classe, trocando a primeira letra para minúscula.

Possibilita, ainda, que o desenvolvedor crie seus próprios estereótipos. As classes gerenciadas pelo CDI são associadas a determinados contextos para gerenciamento automático do seu ciclo de vida. O CDI oferece, além disso, uma série de funcionalidades como qualificadores, alternativas, decoradores, interceptadores e eventos que permitem uma grande flexibilidade no desenvolvimento da aplicação (**??**).

# 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 divido 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.

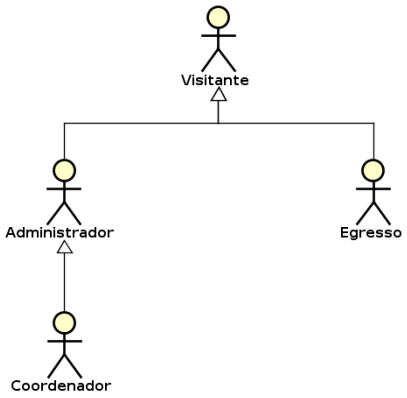


Figura 3 – SAE - CORE - Diagrama de Casos de Uso.

### 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.

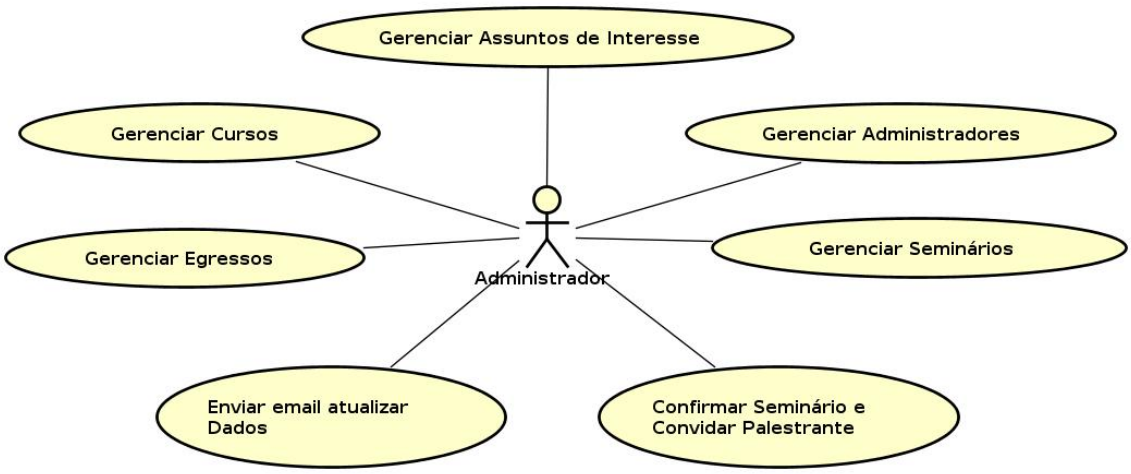


Figura 4 – SAE - CORE - Diagrama de Casos de Uso.

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 matricula.

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

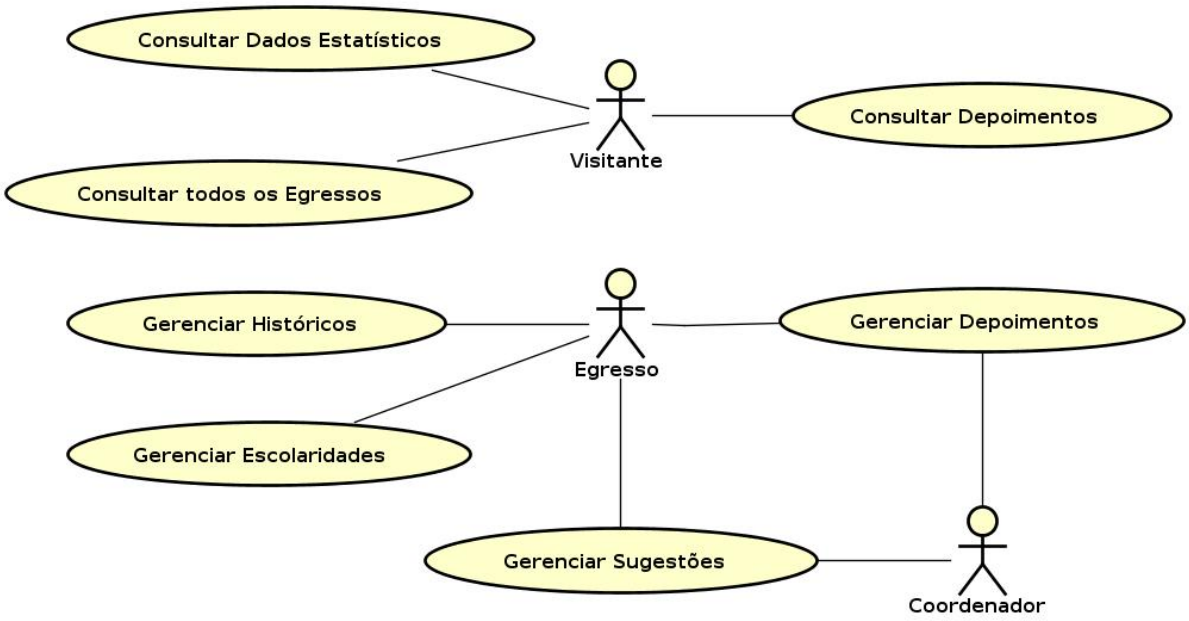


Figura 5 – SAE - PUBLIC - Diagrama de Casos de Uso.

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 criticas 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 Espirito 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 pertecentes 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.

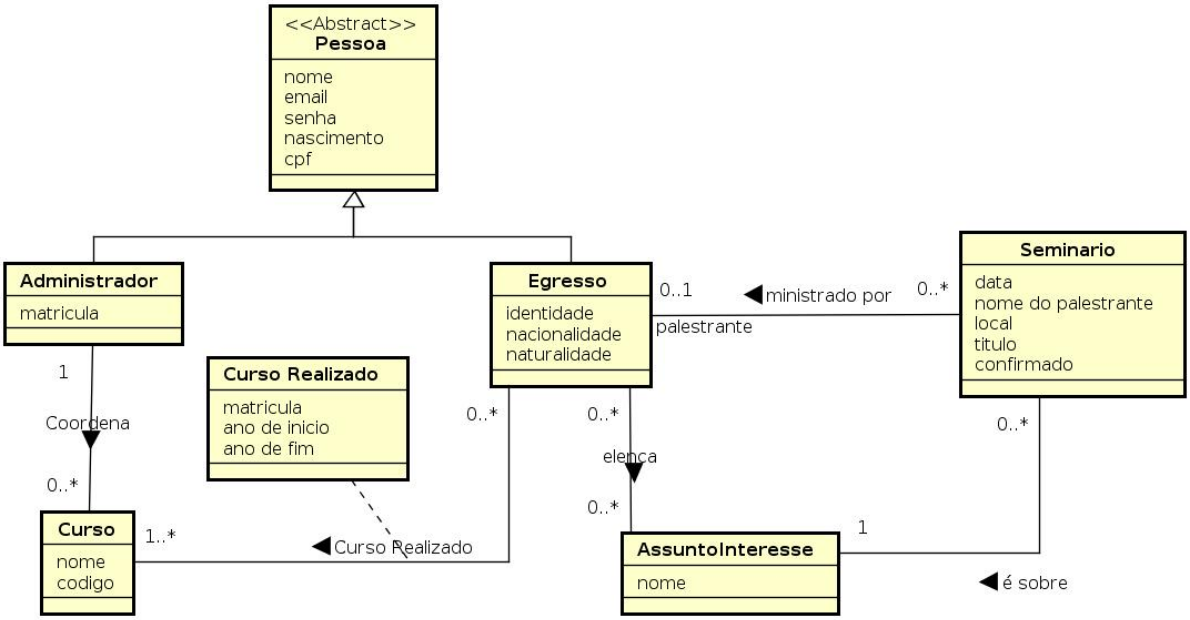


Figura 6 – SAE - CORE - Diagrama de Classes.

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.

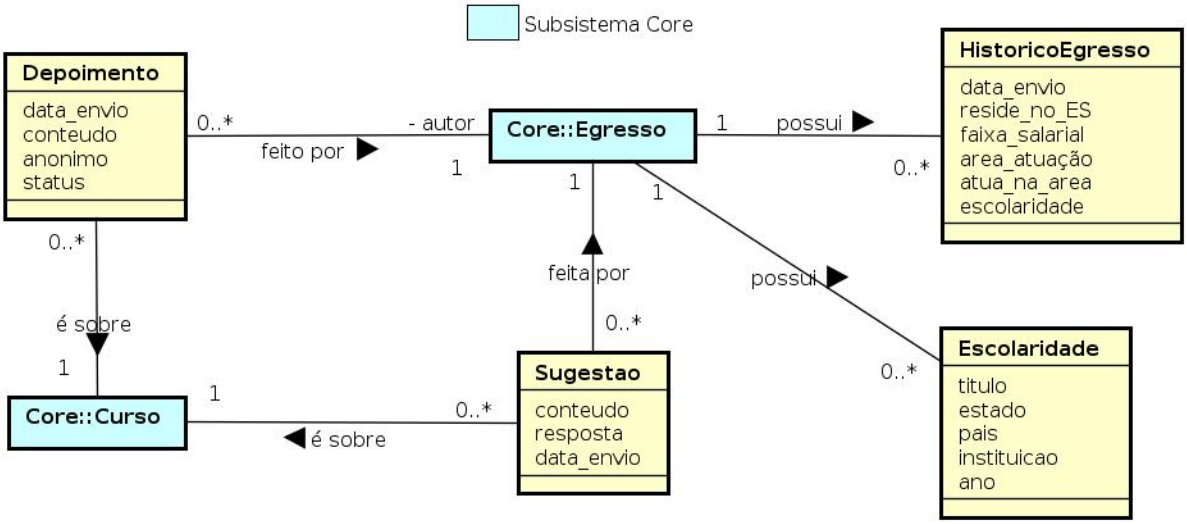


Figura 7 – SAE - PUBLIC - Diagrama de Classes.

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 especifição e análise de requisistos 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 .

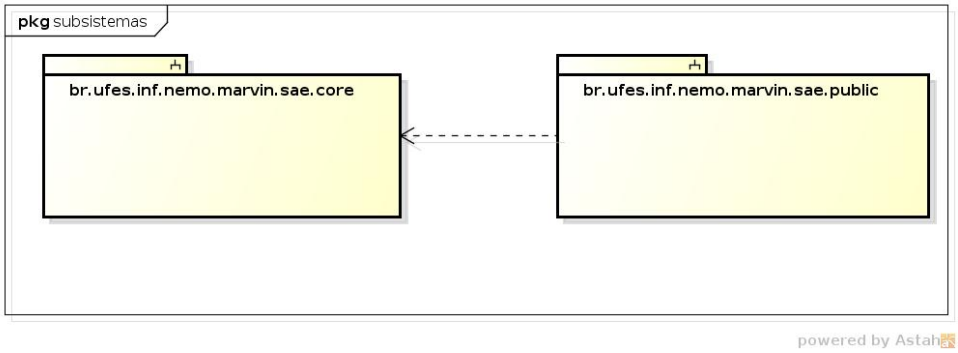


Figura 8 – Pacotes que formam a arquitetura do SAE.

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 divido 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 2.3.

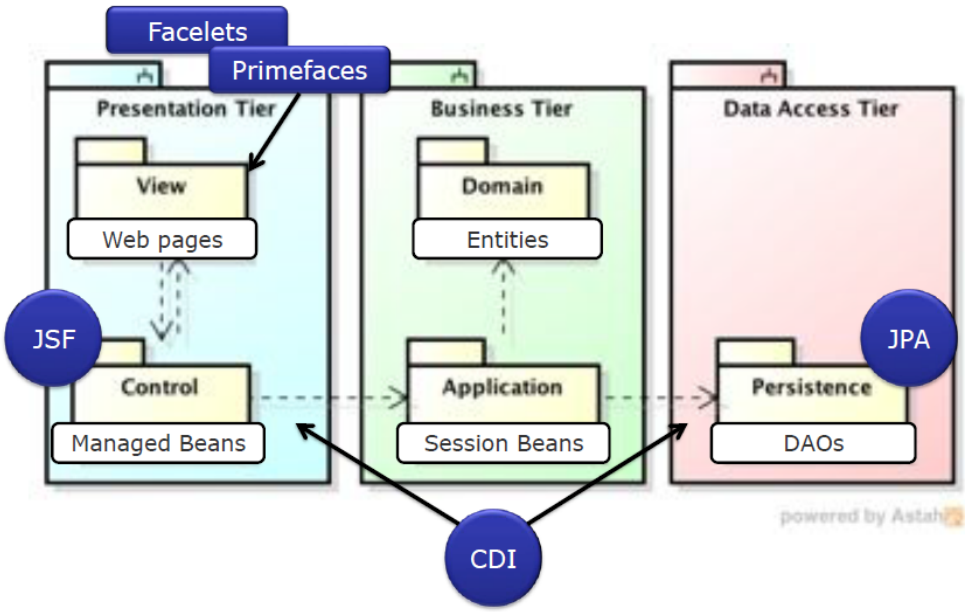


Figura 9 – SAE - Arquitetura - Sistema (**??**).

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.

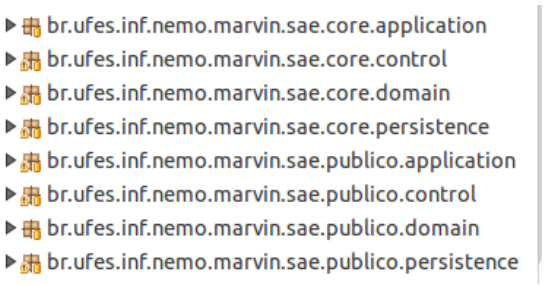


Figura 10 – SAE - Implementação - Pacotes.

### 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 divida em `search`, para os casos de uso de consulta e `alumni` para os casos de usos relacionado aos egressos.

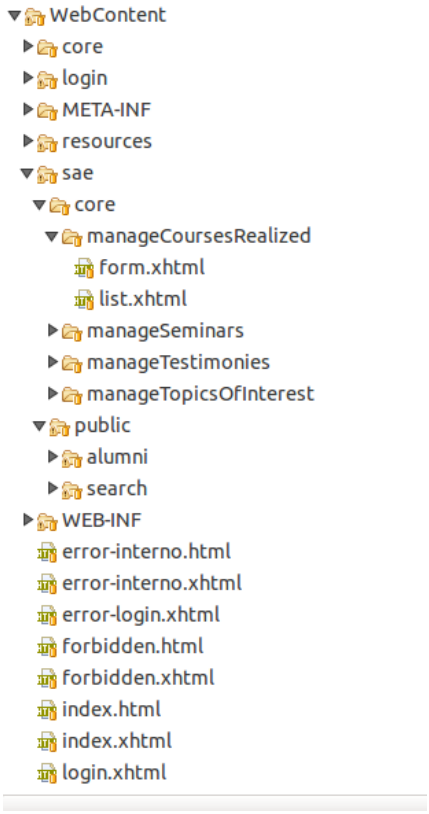


Figura 11 – SAE - Implementação - Páginas Web

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 utilizas 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 2.3.4) 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* [1], 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 reúso quanto ao desenvolvimento para reúso*". 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 despender 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-

[1] 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 `BaseJPADAO` 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 2.2. 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.

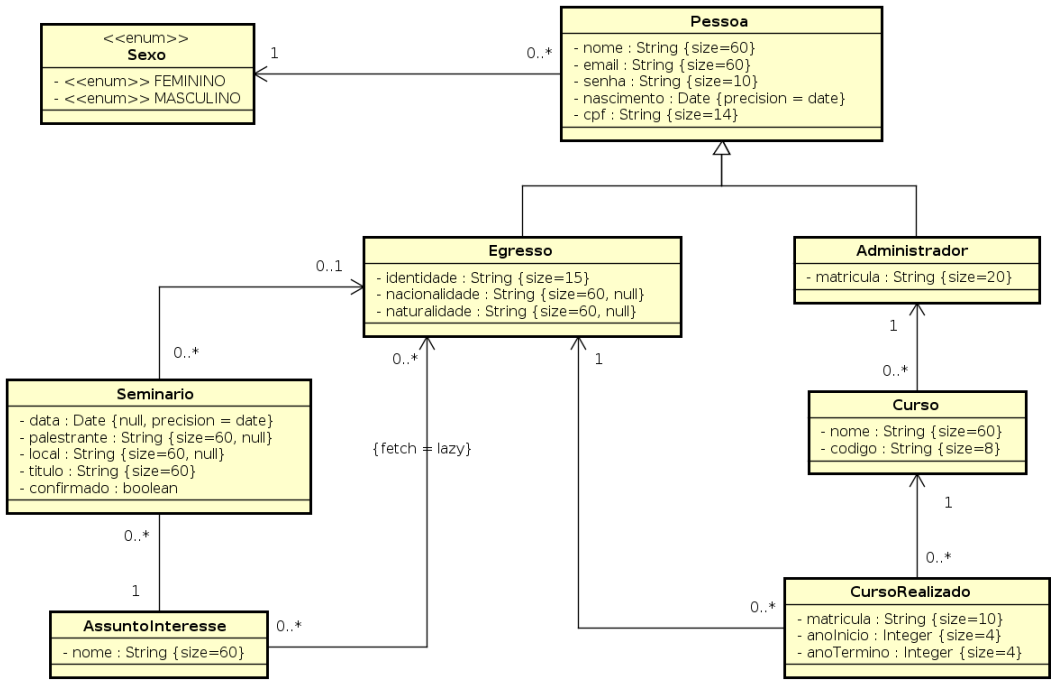


Figura 12 – FrameWeb - sae.core - Modelo Domínio.

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.

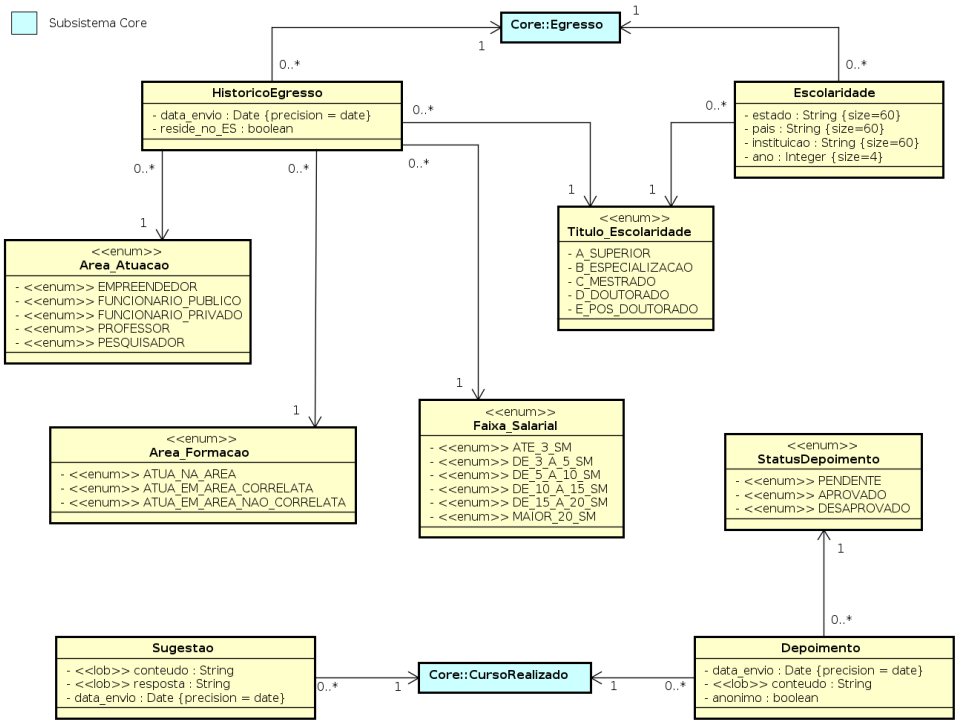


Figura 13 – FrameWeb - Public - Modelo Domínio.

### 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 pagina *index.xhtml* possui um formulário marcado com estereótipo `«form»` que

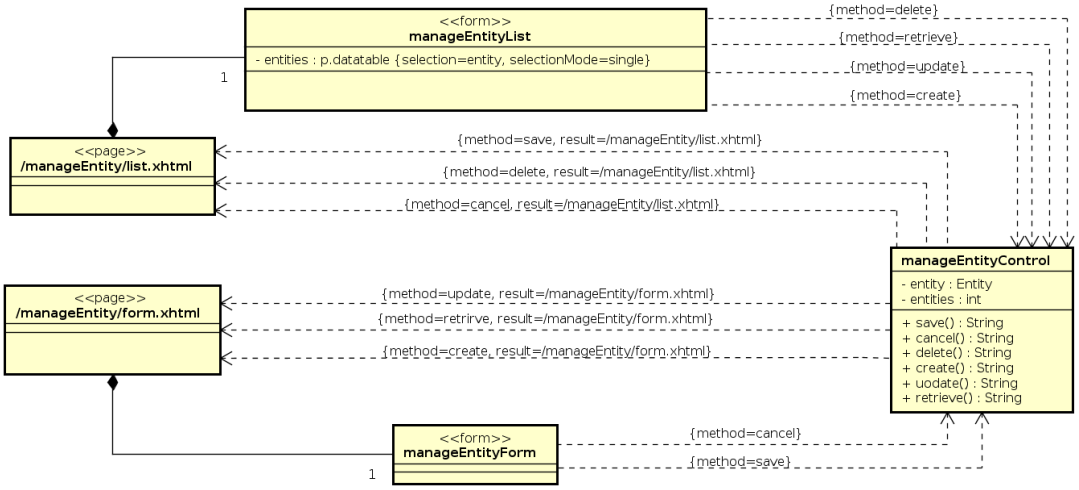


Figura 14 – FrameWeb - *nemo-utils* - Modelo Navegação (**??**).

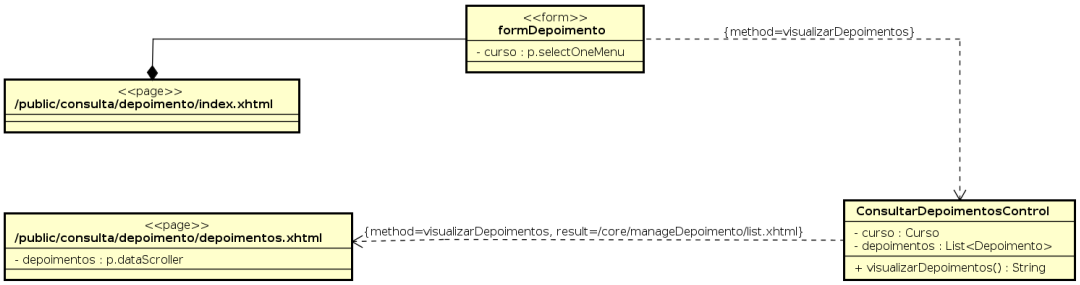


Figura 15 – FrameWeb - Consultar Depoimentos - Modelo Navegação.

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.

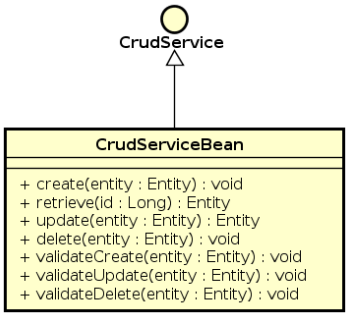


Figura 16 – FrameWeb - *nemo-utils* - Modelo Aplicação (**??**).

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` .

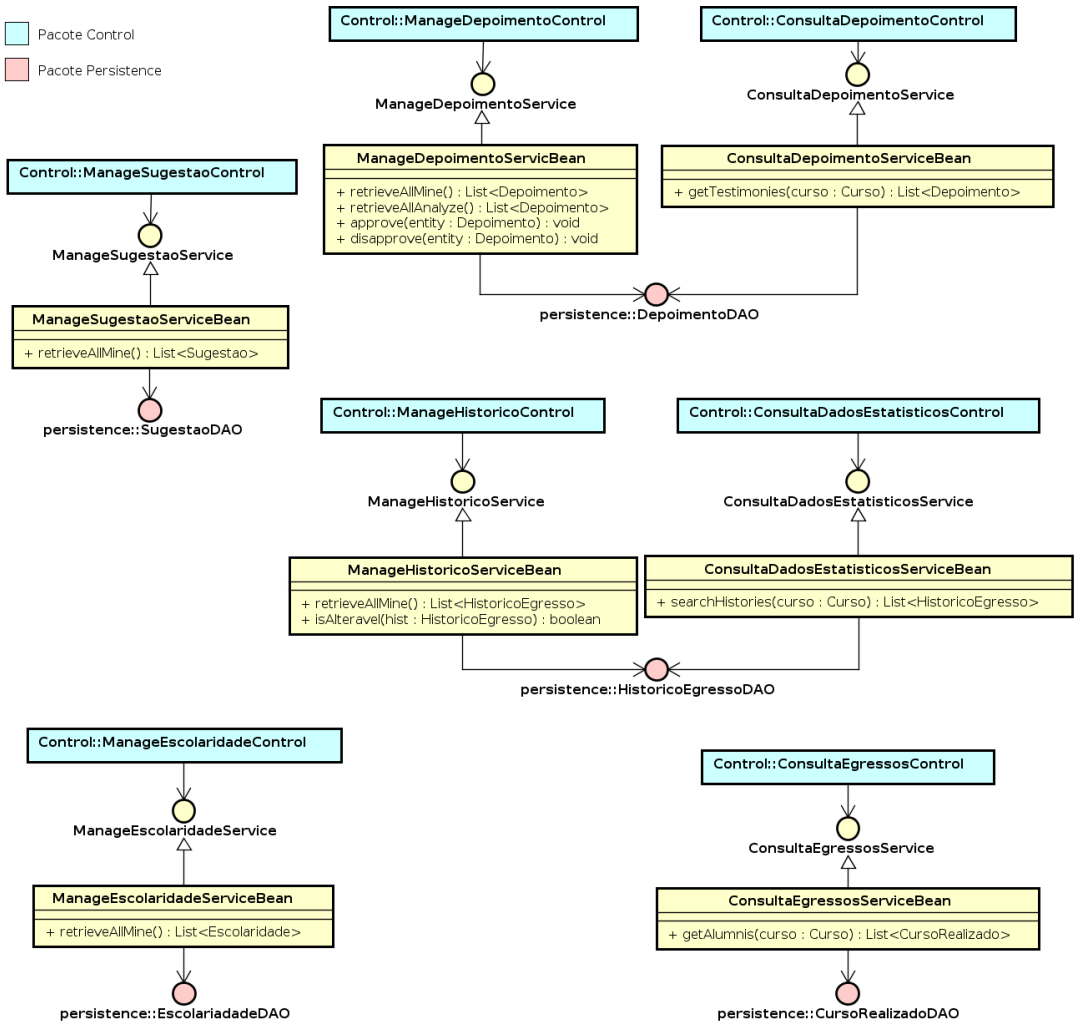


Figura 17 – FrameWeb - Public - Modelo Aplicação.

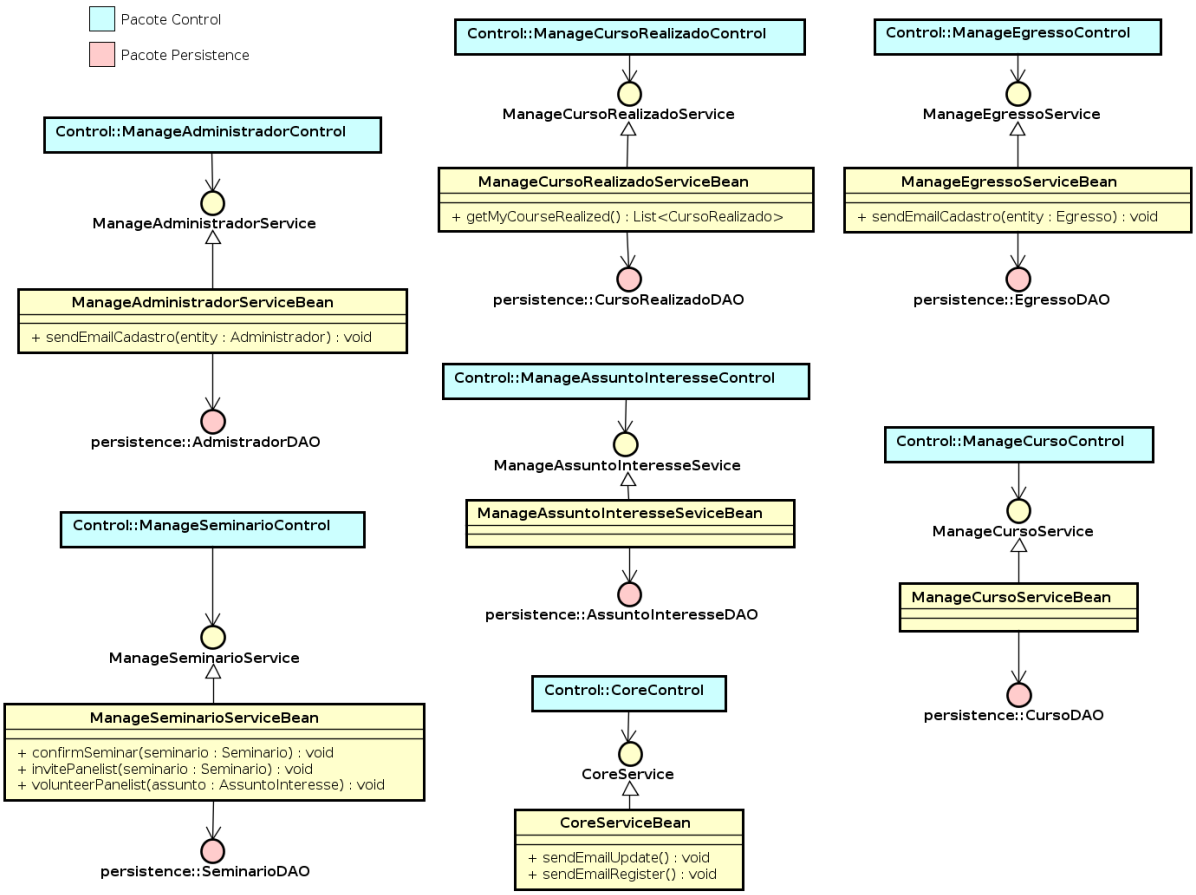


Figura 18 – FrameWeb - Core - Modelo Aplicação.

### 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 **BaseJPADAO** 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

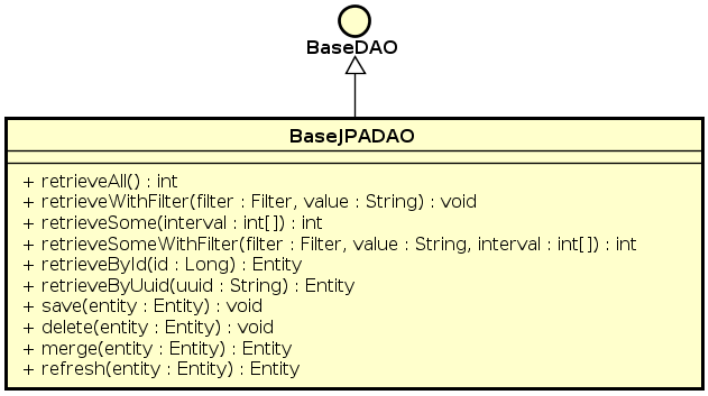


Figura 19 – FrameWeb - *nemo-utils* - Modelo Persistência (**??**).

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.

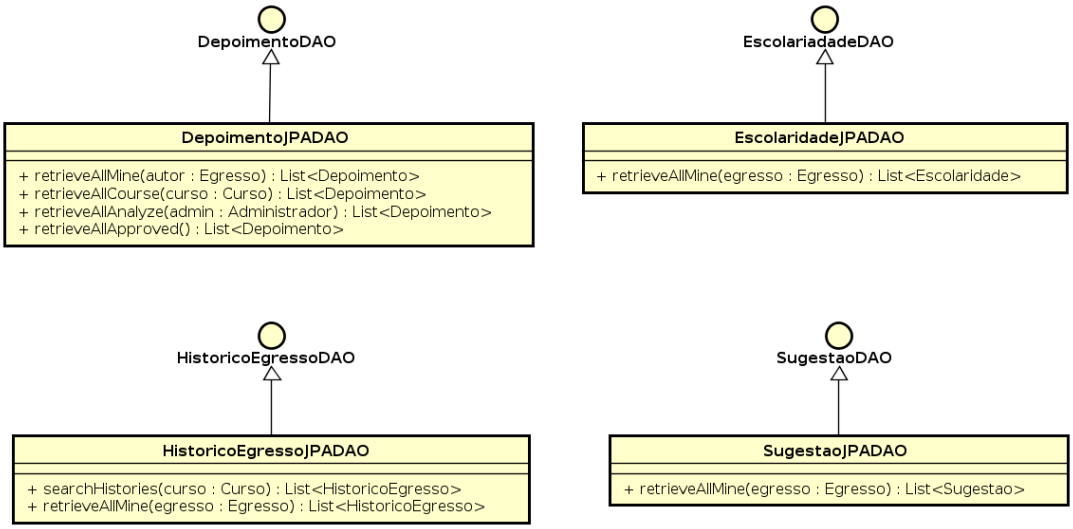


Figura 20 – FrameWeb - Public - Modelo Persistência.

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

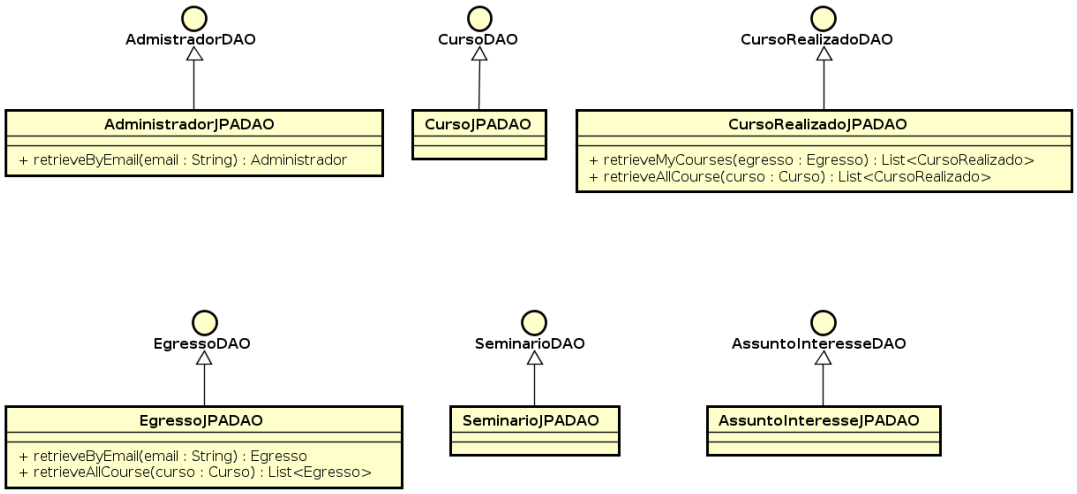


Figura 21 – FrameWeb - Core - Modelo Persistência.

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.

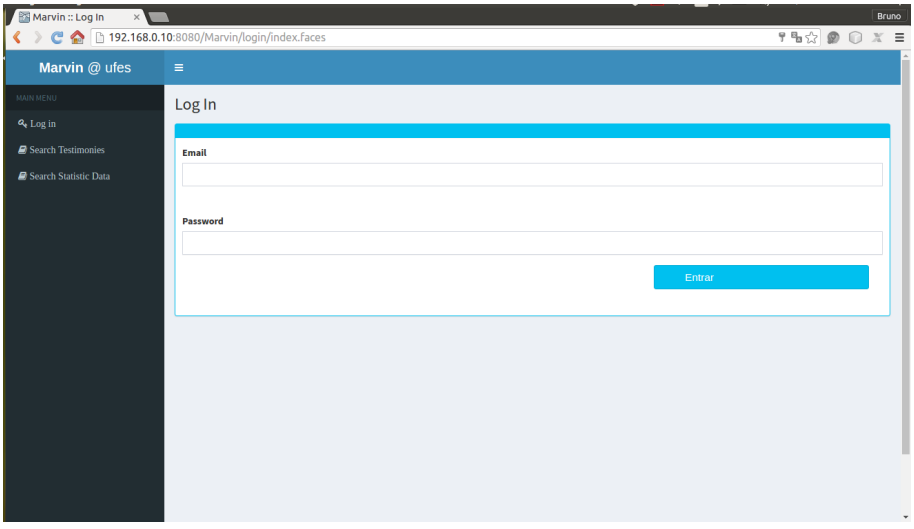


Figura 22 – SAE - Tela Login.

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 á 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.

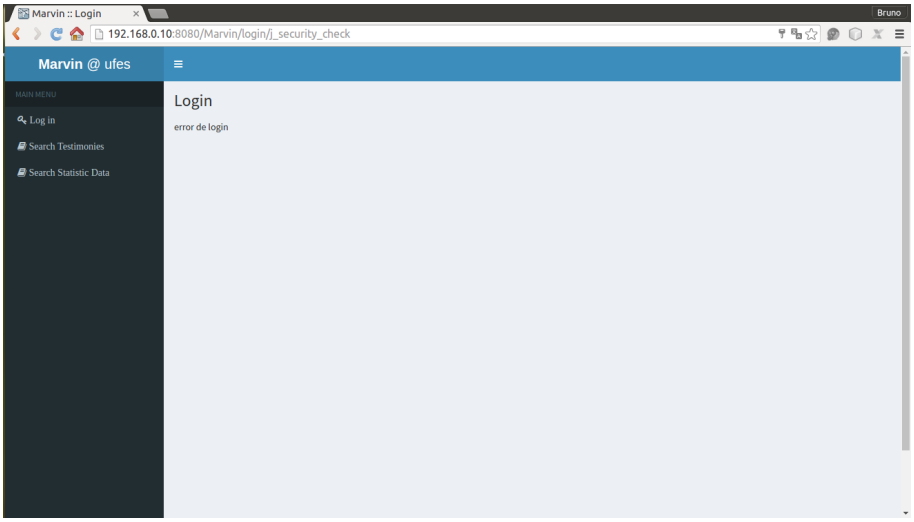


Figura 23 – SAE - Erro Login.

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.

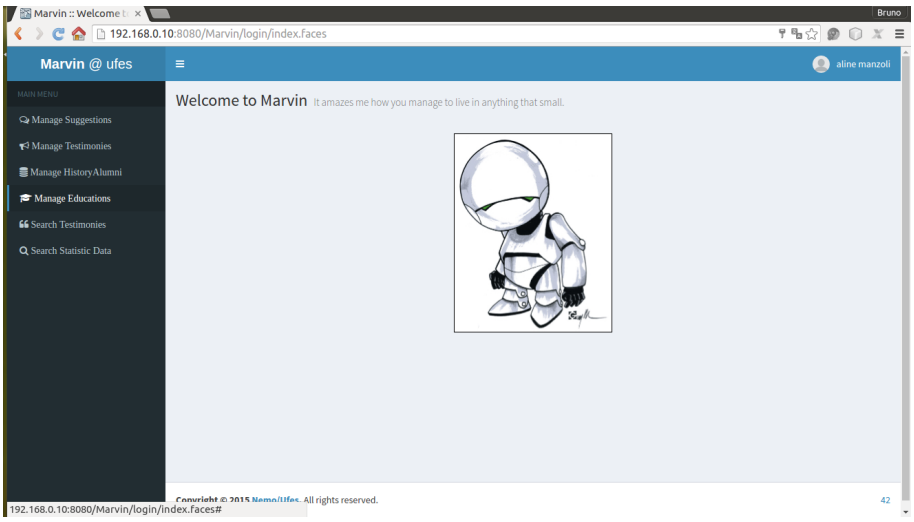


Figura 24 – SAE - Tela inicial Egresso.

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.

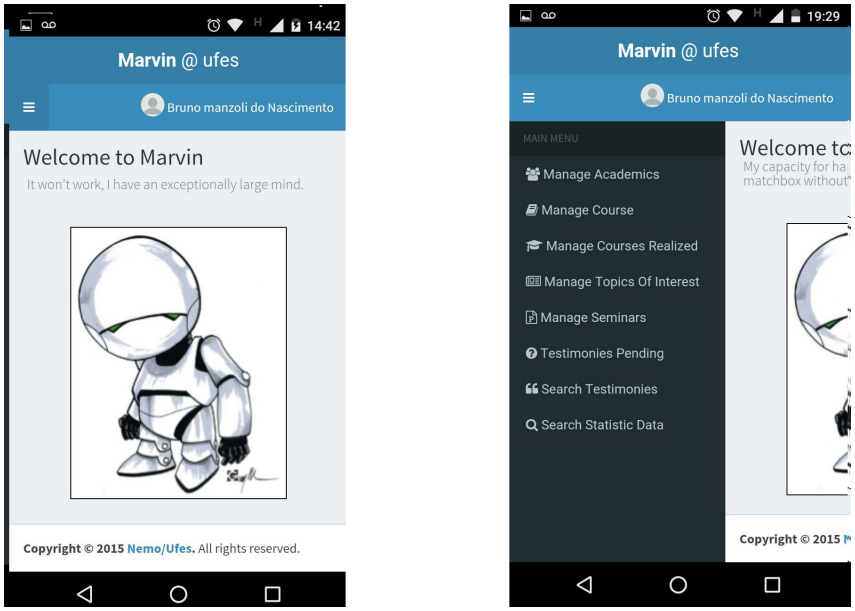


Figura 25 – SAE - Tela Mobile de Administrador.

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.

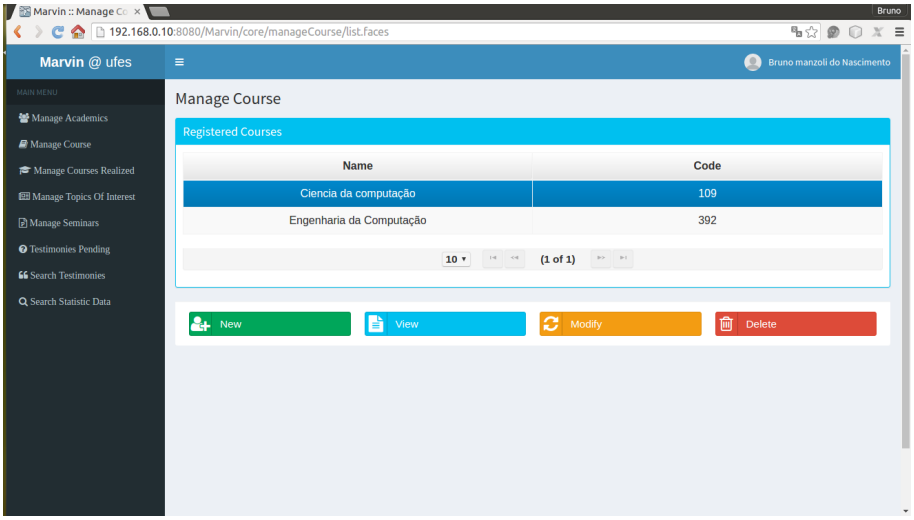


Figura 26 – SAE - Lista de Curso.

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

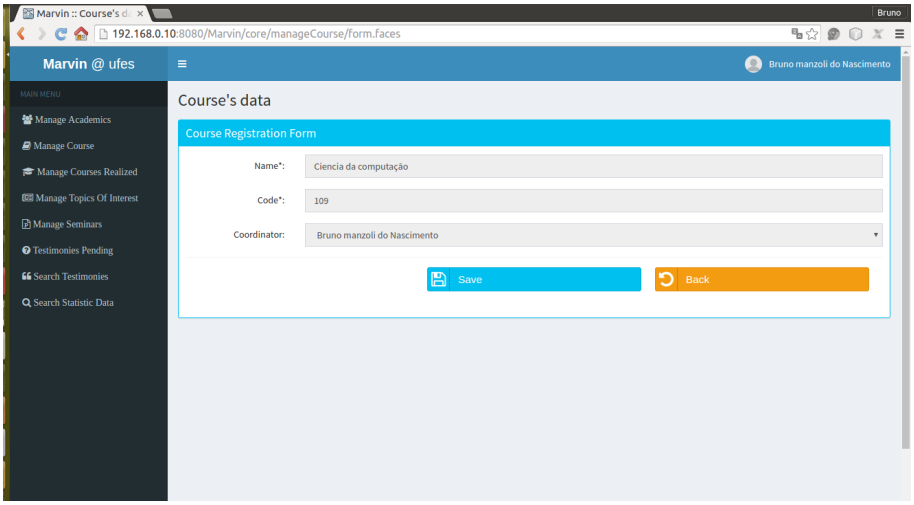


Figura 27 – SAE - Tela cadastro de Curso.

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.

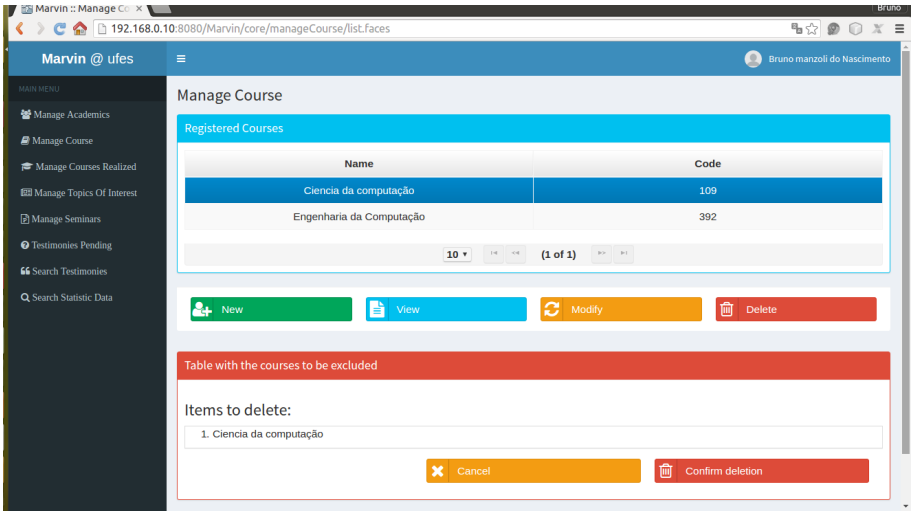


Figura 28 – SAE - Tela exclusão de Curso.

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.

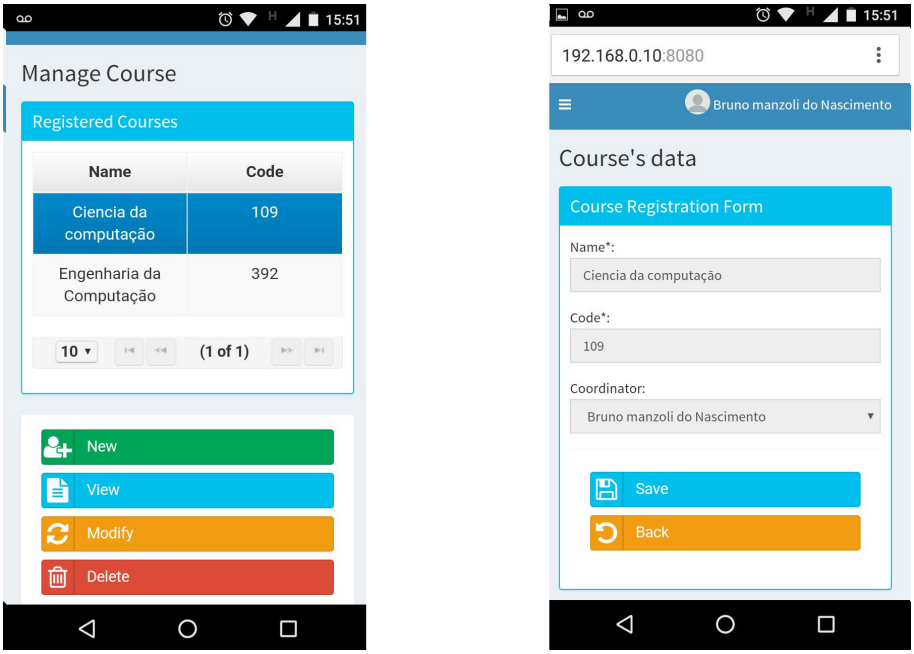


Figura 29 – SAE - Telas mobile.

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.

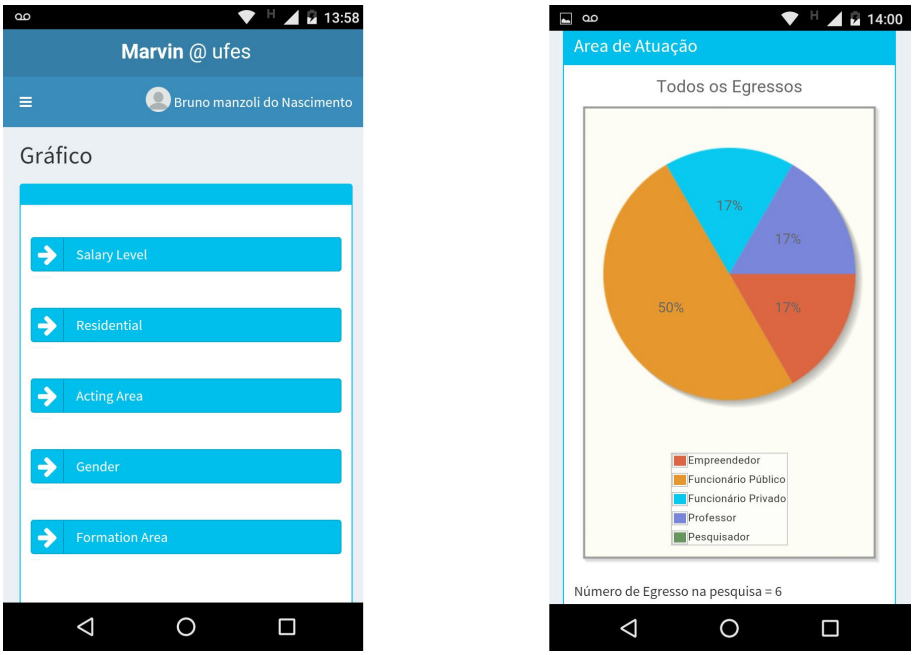


Figura 30 – SAE - Telas mobile Gráfico.

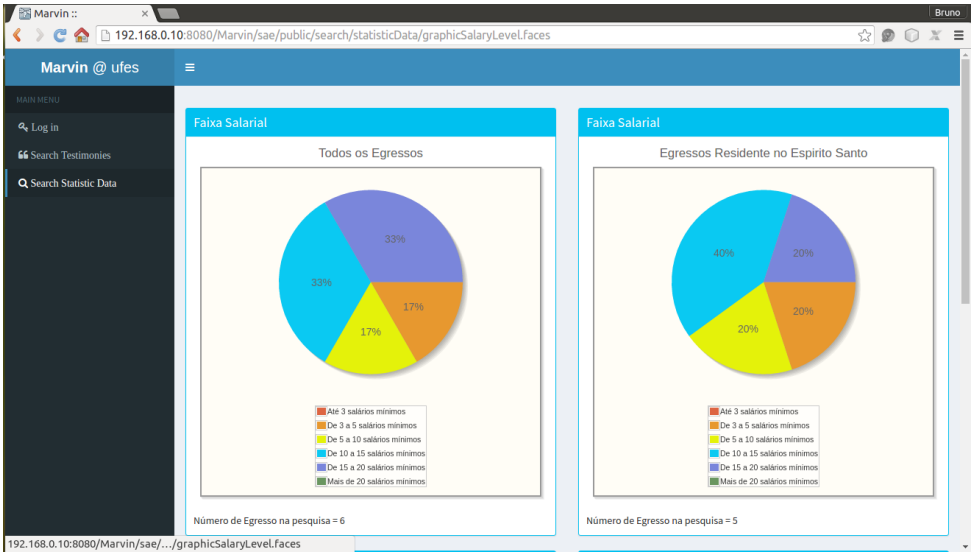


Figura 31 – SAE - Tela Consulta Faixa Salarial.

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 avalização, para avaliar o administrador seleciona o depoimento e clica no botão **View** (visualizar).

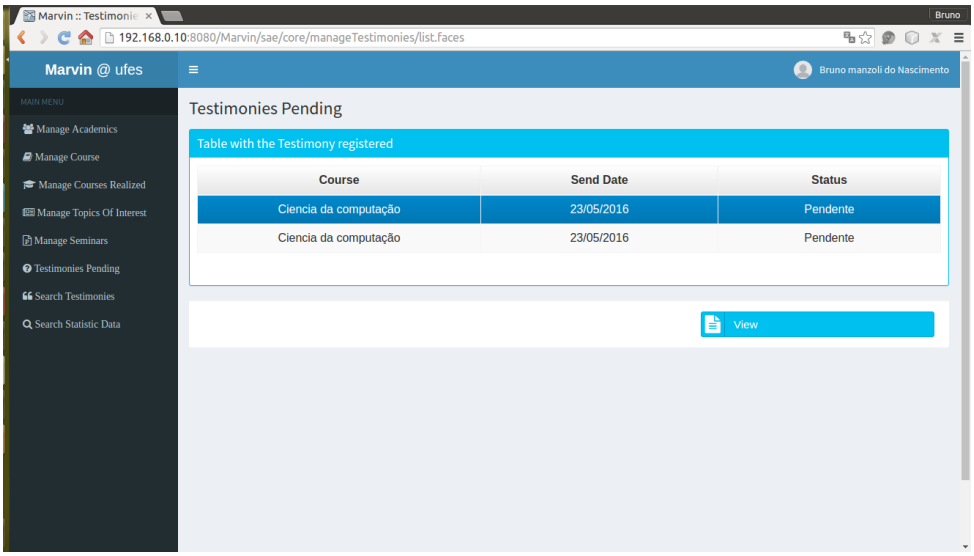


Figura 32 – SAE - Tela Depoimentos á serem analisados.

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.

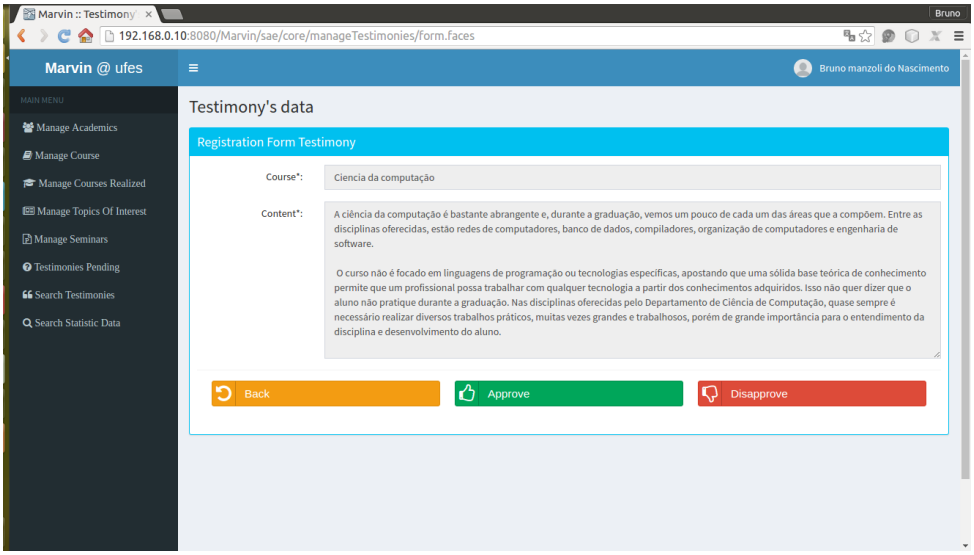


Figura 33 – SAE - Tela Análise de Depoimentos.

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.

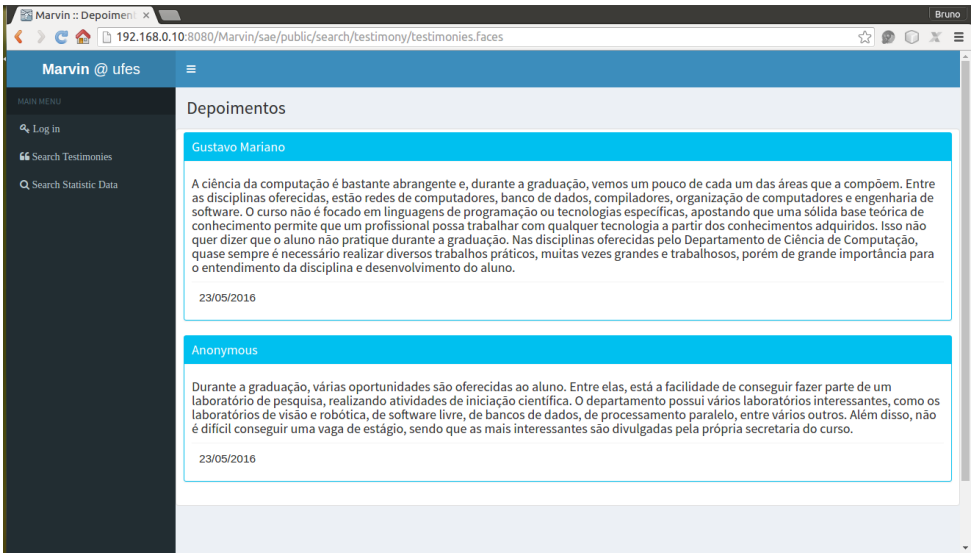


Figura 34 – SAE - Tela Consulta Depoimentos.

# 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. Outra 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 um 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.

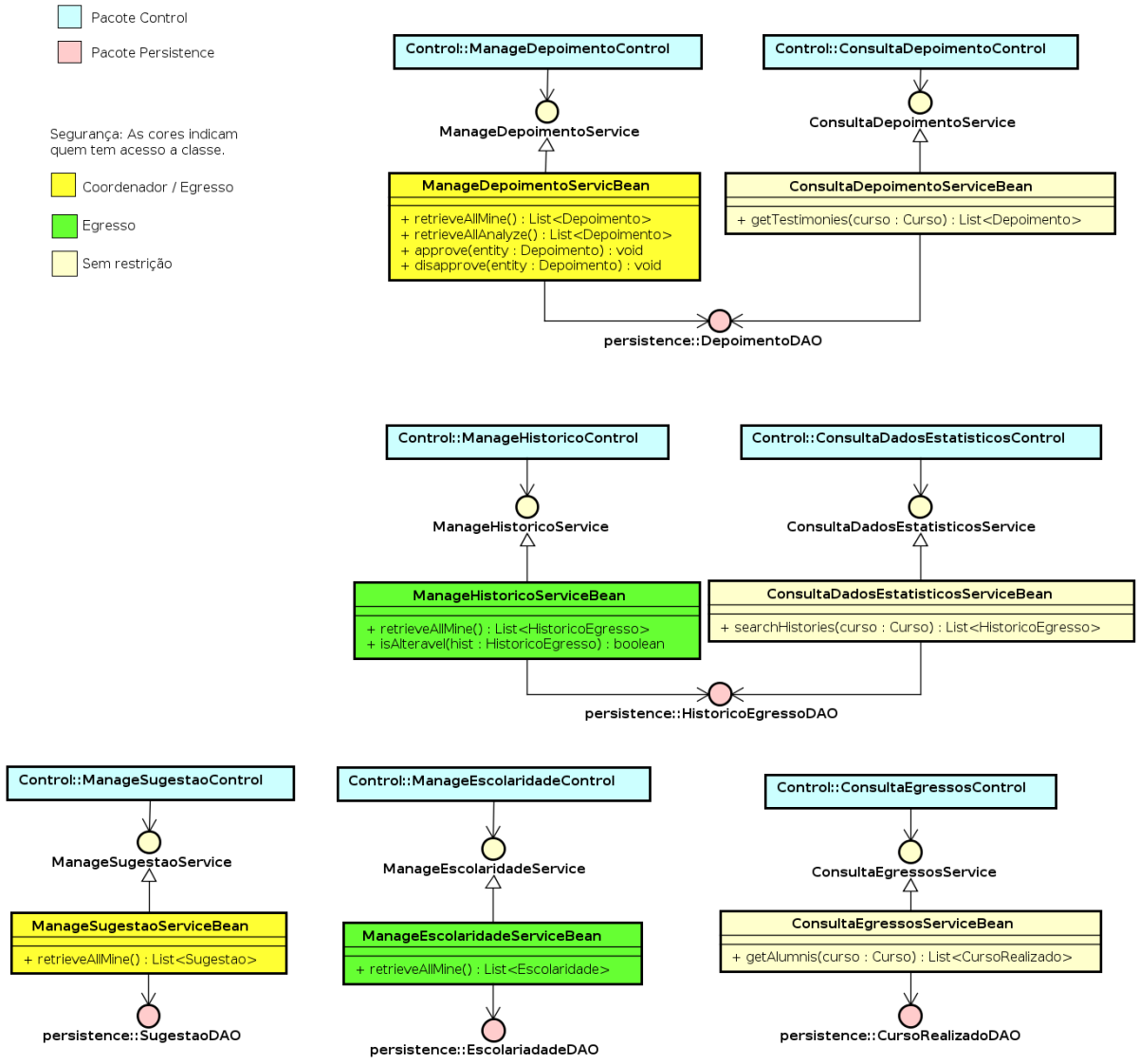


Figura 35 – FrameWeb Modelo de Aplicação Sugerido

# 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 na página 12.

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>. Citado 2 vezes nas páginas 16 e 20.

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>. Citado 3 vezes nas páginas 16, 18 e 19.

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.; 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.

SOUZA, V. E. S. *Requirements-based software system adaptation*. Tese (Doutorado) — University of Trento, 2012. Citado na página 12.

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 na página 12.

# Apêndices