Rodrigo Bittencourt Pimenta

SIGME - Sistema de Informação Gerencial do Movimento Espírita

Vitória, ES

2019

Rodrigo Bittencourt Pimenta

SIGME - Sistema de Informação Gerencial do Movimento Espírita

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

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Departamento de Informática

Orientador: Prof. Dr. Vítor E. Silva Souza

Vitória, ES

2019

Rodrigo Bittencourt Pimenta
SIGME - Sistema de Informação Gerencial do Movimento Espírita/ Rodrigo Bittencourt Pimenta. – Vitória, ES, 2019-
85 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, 2019.

1. Desenvolvimento web. 2. FrameWeb. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. SIGME - Sistema de Informação Gerencial do Movimento Espírita

CDU 02:141:005.7

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Centro Tecnológico
Colegiado do Curso de Engenharia de Computação
Coordenação de Projeto de Graduação

ATA DE APRESENTAÇÃO ORAL DO PROJETO DE GRADUAÇÃO

Título: SIGME - Sistema de Informação Gerencial do Movimento Espírita

Aluno: Rodrigo Bittencourt Pimenta

Matrícula: 2012103286

Orientador: Vitor E. Silva Souza

Co-orientador: _____

Data: 08 de Julho de 2019. Local: CT7 - AQUÁRIO

Critérios	Nota Atribuída
Apresentação, monografia, conhecimento sobre o assunto	10,0

Banca examinadora (nomes)	Assinatura
Davidson Cury	
César Henrique Bahmad	
Vítor Estêvão Silva Souza	

Observações:

Presidente da Sessão:

Nome e assinatura: Vítor E. Silva Souza

Assinatura do aluno: _____

☐ O CD com a Monografia (em formato PDF) referente ao projeto de graduação do aluno foi entregue juntamente com o Anexo II da Resolução 02/2013-CCEC.

☐ A Monografia (PDF) foi encaminhada por email para o Colegiado pelo Orientador.

Data, nome, assinatura e carimbo da Secretaria do CCEC

Agradecimentos

Agradeço à minha família que é a base para tudo na minha vida. Sem a educação e os exemplos de vida do meu pai, Marcos Saraiva Pimenta, e da minha mãe, Lídia Dietrich Bittencourt, eu não seria a pessoa que sou hoje. Obrigado pelas palavras de conforto quando pensei em desistir e, acima de tudo, por acreditarem em mim todos os dias. Agradeço ao meu irmão e aos meus amigos, que sempre ficaram ao meu lado e aguentaram a minhas ausências por conta da sobrecarga de atividades. Sem a presença de vocês tenho certeza de que a trajetória seria bem mais difícil.

Agradeço também a todos os professores do Departamento de Informática e Elétrica, que se dedicaram a passar todo o conhecimento. Um agradecimento especial ao Vitor, meu orientador, que é um grande professor, com uma qualidade de aula e uma conduta extraordinária.

E, por fim, agradeço a Deus, pois sem a força que ele me deu durante todos esses anos eu não teria conseguido chegar até aqui.

Resumo

A Federação Espírita do Estado do Espírito Santo (FEEES) é uma organização que gerencia todas as informações do Movimento Espírita no Estado. Muitas vezes essas informações não se encontram em um único local, ou são de difícil acesso para a secretária da FEEES. Viu-se, então, a necessidade de construção de um sistema de gerenciamento dessas informações, por meio do qual os trabalhadores do Movimento Espírita poderiam cadastrar instituições, verificar as gestões de determinada instituição e visualizar e se inscrever em eventos realizados pela FEEES, ou por outras instituições.

Com isso surgiu o Sigme, Sistema de Informação Gerencial do Movimento Espírita, que em sua primeira versão tem somente os cadastros de Instituições e Espíritas. Neste trabalho incluímos dois novos módulos, o de Gerenciamento de Eventos e Gerenciamento de Gestão, utilizando um processo de Engenharia de Software em conjunto com o método FrameWeb. Realizamos as etapas de levantamento e especificação de requisitos, definição da arquitetura do sistema, criação dos modelos do FrameWeb e sua implementação com base nos documentos e modelos criados.

Durante a realização desse trabalho, foram utilizadas práticas aprendidas nas disciplinas de Engenharia de Software, Programação 3, Linguagem de Programação, Banco de Dados e Desenvolvimento Web e Semântica. Foi utilizado o método FrameWeb, em específico, para o projeto da aplicação Web baseada em *frameworks*.

Palavras-chaves: SIGME, Engenharia Web, frameworks, Java EE, FrameWeb.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CDI	Contexts and Dependency Injection
DAO	Data Access Object
FrameWeb	Framework-based Design Method for Web Engineering
HTML	HyperText Markup Language
Java EE	Java Platform, Enterprise Edition
JPA	Java Persistence API
JSF	JavaServer Faces
MVC	Model-View-Controller
ORM	Object-relational Mapping
SQL	Structured Query Language
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	eXtensible Markup Language

Sumário

1 Introdução

A FEEES é uma organização religiosa, apolítica, de caráter doutrinário, filantrópico e cultural, segundo os fundamentos científicos, filosóficos e morais da Doutrina Espírita, resultante da união das sociedades espíritas sediadas no Estado do Espírito Santo, que objetiva unificar, orientar, coordenar e dinamizar o movimento espírita no Estado e participar do movimento espírita nacional, para divulgar o Espiritismo com base na Codificação de Allan Kardec, no seu tríplice aspecto: científico, filosófico e moral.

A FEEES realiza uma série de atividades por meio da ação de seus departamentos, acompanhados pelos vice-presidentes, além da tesouraria e secretaria, todos coordenados pela presidência. Ela possui um site que provê informações sobre estrutura (diretoria, departamentos, instituições, etc.), e sobre outras informações relevantes para o meio espírita. Para agilizar alguns processos referentes às atividades citadas anteriormente foi idealizado um sistema de gerenciamento denominado Sistema de Informação Gerencial do Movimento Espírita (SIGME).

O SIGME foi idealizado para permitir o cadastro de espíritas e/ou instituições e, que os mesmos, pudessem utilizar o sistema para gerenciar conteúdos dos sites das instituições, gerenciar e organizar eventos, ter um controle financeiro com geração de boletos para cobrança, controle de assinantes informativos, agendamento de registros de atividades, entre outros.

Dado que o núcleo do SIGME, incluindo o cadastro das instituições e dos espíritas, foi desenvolvido anteriormente por Regis (2018), este trabalho consiste na inclusão de dois novos módulos, a saber:

1. **Módulo de Gerenciamento de Eventos** Muitas das atividades desenvolvidas pela FEEES são palestras, congressos e/ou eventos destinados para o público que, em muitos casos, precisa se inscrever para participar de tais eventos. A inscrição em determinada atividade é realizada no site da FEEES, tendo o usuário que procurar o evento e enviar determinados dados para realizar a sua inscrição. O novo módulo exibe para o usuário todos os eventos cadastrados no sistema, permitindo que o mesmo escolha qual evento ele gostaria de participar — bastando preencher a ficha de inscrição pré-determinada por quem criou o evento. Para uma maior facilidade do usuário, são exibidas todas as atividades que o mesmo participou ou se inscreveu e, caso for de seu interesse, o mesmo pode cancelar a sua participação no evento. Esse módulo facilita a comunicação dos participantes com os organizadores do evento, além de adicionar facilidades aos organizadores (tais como a exibição da lista dos inscritos, e a geração automática dos crachás do evento).

2. **Módulo de Gerenciamento de Gestão**

   Outras informações que a FEEES exibe para os usuários que acessam o seu site é a sua estrutura atual — composta de um presidente, 4 vice-presidentes que são responsáveis pelas áreas de Administração, Unificação, Educação e Doutrina, além dos Conselhos (Fiscal e Regional Espírita). Essa estrutura se modifica ao longo dos anos, com novas eleições. O usuário cadastrado no sistema tem a possibilidade de visualizar não só a estrutura atual da FEEES, ou de outra Instituição cadastrada no sistema, como suas gestões passadas. Após novas eleições o sistema permite o cadastro dessa nova gestão, permitindo que usuários cadastrados sejam designados para os cargos existentes. Isso facilita o controle das gestões das instituições e a divulgação da mesma.

1.1 Objetivos

O objetivo geral desse projeto consiste em aprimorar um sistema com interface Web, responsivo, com os dois novos módulos citados anteriormente, utilizando conceitos estudados durante o curso de Engenharia de Computação.

Os objetivos específicos do projeto são:

1. Levantar e especificar os requisitos para termos um escopo definido do que deve conter os dois novos módulos, utilizando conceitos aprendidos na disciplina de Engenharia de Software;

2. Definir a arquitetura do sistema utilizando o método FrameWeb (SOUZA, 2007) e detalhar essa arquitetura em um Documento de Projeto, utilizando conceitos das disciplinas: Engenharia de Software e Desenvolvimento Web;

3. Implementar o sistema de acordo com a arquitetura sugerida, buscando *frameworks* e ferramentas já existentes para auxiliar no desenvolvimento e utilizando conceitos das disciplinas: Desenvolvimento Web, Programação III, Estrutura de Dados I e Banco de Dados;

4. Apresentar o trabalho desenvolvido.

1.2 Metodologia

A metodologia desse trabalho foi composta pelas seguintes atividades:

1. *Leituras preparatórias:* estudo das boas práticas de Engenharia de Software e de Requisitos, dos padrões de design e desenvolvimento de uma interface Web, de banco

de dados e do mapeamento objeto/relacional, dos conceitos de UML, da programação orientada a objetos — em específico, a linguagem Java e seus padrões —, entre outros;

2. *Elaboração dos documentos:* inicialmente foi elaborado o Documento de Requisitos do sistema, contendo o mini-mundo, os requisitos funcionais (RF), requisitos não funcionais (RNF), regras de negocio (RN), os modelos de casos de uso para cada um dos subsistemas e o modelo estrutural do projeto. Em seguida, foi construído o Documento de Projeto que contém uma breve descrição dos softwares e *frameworks* utilizados no desenvolvimento, as técnicas para garantir a qualidade do código desenvolvido e a descrição da arquitetura de software do sistema, utilizando aqui os conceitos do método FrameWeb;

3. *Estudo das tecnologias:* estudo das boas práticas das linguagens e softwares utilizados, sendo alguns deles: linguagem Java, Modelo MVC (*Model-View-Controller*), EJB, CDI, JSF, JPA, Primefaces, SQL, entre outros;

4. *Implementação do sistema:* implementação dos dois novos módulos de acordo com as tecnologias estudadas;

5. *Redação da monografia:* escrita da monografia, utilizando a ferramenta LaTeX e o modelo de documento abnTeX2.[1]

1.3 Organização do Texto

Esse trabalho está organizado da seguinte maneira:

- O Capítulo 1 refere-se à introdução do trabalho e apresenta os Objetivos e a Metodologia;
- O Capítulo 2 resume o referencial teórico do trabalho, apresentando e explicando alguns conceitos necessários para o entendimento do mesmo;
- O Capítulo 3 apresenta a especificação dos requisitos do sistema, descrevendo o mini-mundo e exibindo os diagramas e casos de uso;
- O Capítulo 4 descreve a arquitetura do sistema, mostra a implementação do sistema e exibe algumas telas das novas funcionalidades do sistema;
- O Capítulo 5 contém a conclusão do projeto e as considerações finais onde é indicado possíveis melhorias para o projeto de acordo com as novas práticas abordadas nas interfaces Web.

[1] <http://www.abntex.net.br>.

2 Referencial Teórico

Esse capítulo introduz os conceitos teóricos utilizados para implementação dos módulos propostos. O mesmo está separado em seções da seguinte forma:

- A Seção 2.1 apresenta conceitos sobre Engenharia de Software que foram utilizados;
- A Seção 2.2 apresenta os conceitos voltados ao desenvolvimento de software, apresentando os *frameworks* utilizados e alguns conceitos sobre o desenvolvimento Web;
- A seção 2.3 dá uma breve descrição do método FrameWeb (SOUZA, 2007).

2.1 Engenharia de Software

De forma a desenvolver um software com qualidade, criou-se um conjunto de práticas que permitiam aumentar a confiabilidade e a qualidade do software que estava sendo desenvolvido, denominada Engenharia de Software. As práticas mais comuns utilizadas são definidas como (i) Levantamento de Requisitos, (ii) Análise/Especificação de Requisitos, (iii) Projeto, (iv) Implementação, (v) Testes e (vi) Implementação. Antes de introduzirmos tais práticas precisamos deixar claro o que é um requisito, portanto dispomos essa seção da seguinte maneira:

- A Seção 2.1.1 explica melhor o que é um requisito e suas definições;
- A Seção 2.1.2 explica os conceitos a prática de Levantamento de Requisitos;
- A Seção 2.1.3 explica a prática de Análise e Especificação de Requisitos;
- A Seção 2.1.4 explica a etapa de Projeto/Design de Software.

2.1.1 Requisitos

Nesta seção descrevemos de forma sucinta o que é um requisito e diferenciamos os tipos existentes.

Podemos dizer, com bases em diversas definições encontradas, que os requisitos de um sistema incluem especificações dos serviços que o sistema deve prover, restrições sob as quais ele deve operar, propriedades gerais do sistema e restrições que devem ser satisfeitas no seu processo de desenvolvimento (BARCELOS, 2018).

Podemos definir os seguintes tipos de requisitos:

- **Requisitos Funcionais:** definem a funcionalidade desejada do software. São, na realidade, as funções que o cliente e os usuários querem que o software faça;
- **Requisitos Não Funcionais:** são as qualidades gerais que um software deve possuir. Alguns exemplos: usabilidade, desempenho, portabilidade, custos, dentre outros. De modo geral, esses requisitos são difíceis de validar;
- **Requisitos de Domínio:** também são conhecidos como **Regras de Negócio**, são provenientes do domínio de aplicação do sistema e refletem características e restrições desse domínio (BARCELOS, 2018). São regras que garantem a implementação correta dos requisitos funcionais citados anteriormente.

Os tipos de requisitos acima apresentados devem ser passíveis de entendimentos por todos os envolvidos no processo de desenvolvimento de um software. Isso nos dá uma nova divisão de requisitos

- **Requisitos de Cliente/Usuário:** são requisitos com um alto nível de abstração, de fácil entendimento para os usuários do sistema que não possuam um conhecimento técnico. Normalmente podem ser exemplificados com o uso de diagramas;
- **Requisitos de Sistema:** são requisitos com um alto nível de detalhamento técnico descrevendo as funções, serviços e restrições do sistema que serão utilizadas durante a implementação do sistema.

Como os requisitos de Sistema e de Cliente possuem propósitos diferentes é comum elaborar dois documentos de requisitos, sendo um deles denominado **Documento de Definição de Requisitos** e outro denominado **Documento de Especificação de Requisitos**. No nosso projeto, unificamos esses dois documentos e o denominamos como Documento de Requisitos. Para a criação desse documento realizamos as práticas de **Levantamento de Requisitos** e **Análise e Especificação de Requisitos**, representadas nas seções 2.1.2 e 2.1.3, respectivamente.

2.1.2 Levantamento de Requisitos

É a fase inicial do processo de Especificação de Requisitos e em sua maioria consiste em conversas com os clientes — também chamados de *stakeholders* —, consulta de documentos, estudos quanto ao negócio da organização para qual a aplicação está sendo desenvolvida, entre outras. Esta etapa é importante para o processo pois facilita os próximos passos e miniminiza eventuais erros de especificação. A partir dela temos uma melhor definição dos requisitos funcionais, não-funcionais e as regras de negócio que o cliente espera da aplicação. Os capítulos do Documento de Requisitos, nos apêndices,

referentes à parte de levantamento de requisitos são a Introdução, Descrição de Propósito dos Módulos em conjunto com o Minimundo e os Requisitos do Usuário. Em tais capítulos foi utilizado a linguagem natural de forma que qualquer pessoa consiga identificar o que se espera do sistema.

2.1.3 Análise e Especificação de Requisitos

Nessa etapa é realizado o refinamento dos requisitos funcionais e, portanto, ocorre em paralelo com a etapa anterior de Levamento de Requisitos. Contudo, diferente da etapa de Levantamento onde temos a descrição dos requisitos com um baixo nível de detalhes, nesta etapa queremos um nível de detalhes mais alto permitindo que sejam criados modelos que auxiliem as próximas etapas de desenvolvimento.

A criação de um ou mais modelos que de acordo com Barcelos (2018) são importantes pois tendem a facilitar comunicação entre membros da equipe, possibilitar a discussão de correções e modificações com o usuário, fomentar a documentação e possibilitar o estudo do comportamento do sistema.

Um modelo enfatiza um conjunto de características, que denominamos perspectivas. As mais importantes são:

- Perspectiva estrutural: normalmente são utilizados os Diagramas de Classes para representar essa perspectiva, possuindo um foco no sobre o quê a aplicação se trata;
- Perspectiva comportamental: podem ser utilizados os Diagramas de Classe, Diagrama de Atividades, Diagrama de Estados e Diagramas de Interação para representar essa perspectiva. Esta perspectiva visa especificar quais funcionalidades/serviços que o sistema deve fornecer.

Podemos criar modelos mais ou menos detalhados, de acordo com as nossas necessidades e, os categorizando do maior para o menor nível de abstração, temos modelos conceituais, lógicos e físicos. Nessa etapa da especificação de requisitos normalmente temos a construção dos modelos conceituais. Nas fases seguintes os modelos lógicos e físicos serão construídos, de acordo com as tecnologias e ferramentas escolhidas para o sistema. Para o projeto em questão na perspectiva comportamental utilizamos os Modelos de Casos de Uso e para a perspectiva estrutural, utilizamos os Diagramas de Classes.

Para auxiliar a criação desses modelos utilizamos uma Linguagem de Modelagem Unificada, também conhecida como UML. Tal linguagem auxilia de forma gráfica a especificação, visualização e documentação dos artefatos do sistema.

Com as etapas de Levantamento e Análise de Requisitos concluídas seguimos para a etapa de Projeto de Software, descrito na seção 2.1.4

2.1.4 Projeto de Software

Esta etapa tem por objetivo vincular os modelos criados nas etapas anteriores a uma tecnologia específica. É nesta etapa que definimos qual linguagem de programação será utilizada, quais os mecanismos de persistências serão utilizados, se será utilizado algum padrão específico de projeto ou não, entre outras características mais específicas para o desenvolvimento dos requisitos do que a descrição dos mesmos. Normalmente os documentos produzidos nesta etapa, conforme Barcelos (2018), são:

- Projeto da Arquitetura de Software: visa a definir os grandes componentes estruturais do software e seus relacionamentos;
- Projeto de Dados: tem por objetivo projetar a estrutura de armazenamento de dados necessária para implementar o software;
- Projeto de Interfaces: descreve como o software deverá se comunicar dentro dele mesmo (interfaces internas), com outros sistemas (interfaces externas) e com pessoas que o utilizam (interface com o usuário);
- Projeto Detalhado: tem por objetivo refinar e detalhar a descrição dos componentes estruturais da arquitetura do software.

No projeto em questão, unificamos todos esses documentos em um só dando o nome de Documento de Projeto de Sistema, que pode ser encontrados nos apêndices.

Os padrões utilizados para este sistema serão mais bem aprofundados nas seções de Desenvolvimento Web (2.2), e para auxiliar a criação de modelos com mais detalhes utilizaremos o Método FrameWeb, que é explicado em maiores detalhes na Seção 2.3.

2.2 Desenvolvimento Web

Nessa seção iremos apresentar algumas das ferramentas e *frameworks* utilizados para o desenvolvimento do trabalho. Durante a etapa de Especificação e Análise de Requisitos definimos qual o Paradigma de Programação, também chamado de Paradigma de Desenvolvimento, que usaremos para a elaboração do software. Na Tabela 1 citamos dois paradigmas, porém devemos ter consciência que outros existem e que dependendo do projeto podem se adequar melhor.

O paradigma escolhido para a aplicação foi o Orientado a Objetos. A partir desse momento, podemos escolher algumas linguagens que seguem esse paradigma, por exemplo, Java, C++, entre outras. A escolha da linguagem de programação também é importante, pois de acordo com a linguagem temos funcionalidades e facilidades diferentes. Por ser a linguagem utilizada na construção inicial do sistema SIGME, continuamos com Java

Tabela 1 – Paradigmas de Desenvolvimento

Paradigma	Descrição
Estruturado	Adota uma visão de desenvolvimento baseada em um modelo entrada-processamento-saída. No paradigma estruturado, os dados são considerados separadamente das funções que os transformam e a decomposição funcional é usada intensamente.
Orientado a Objetos	Esse paradigma parte do pressuposto que o mundo é povoado por objetos, ou seja, a abstração básica para se representar as coisas do mundo são os objetos.

para o desenvolvimento dos módulos propostos. Podemos usar a linguagem Java para criar diferentes tipos de aplicações. Porém, como estamos falando de uma aplicação para a Internet, utilizaremos uma plataforma padrão do Java para desenvolver aplicações voltadas para a Internet, ou para aplicações de grande porte. Essa plataforma é denominada Java EE (Java Plataform Enterprise Edition).

2.2.1 Java Plataform Enterprise Edition - Java EE

A Java EE é uma plataforma de desenvolvimento que possui diversas ferramentas para agilizar e auxiliar no desenvolvimento de aplicações Java. O foco dessa plataforma é prover aos desenvolvedores um conjunto poderoso de APIs para diminuir o tempo de desenvolvimento, reduzir a complexidade das aplicações e aumentar a performance das mesmas. O desenvolvedor utiliza de *anotações* durante o desenvolvimento, diretamente nos arquivos Java para informar aos servidores Java EE as configurações dos componentes no momento execução da aplicação.

Para auxiliar o desenvolvimento, essa plataforma disponibiliza algumas tecnologias. Listamos as que são utilizadas no desenvolvimento do projeto:

- **JavaServer Faces (JSF)[1]:** JSF é uma tecnologia que nos permitir criar aplicações Java para Web utilizando componentes visuais pré-prontos, de forma que o desenvolvedor não se preocupe com JavaScript e HTML. Basta adicionarmos os componentes (calendários, tabelas, formulários) e eles serão renderizados e exibidos em formato HTML;
- **Java Persistence API (JPA)[2]:** é uma API padrão do Java para acessar, persistir e gerenciar dados entre os objetos e classes do Java com o modelo relacional do banco de dados. Foi definido como parte da especificação do EJB 3.0;

[1] https://docs.oracle.com/javaee/7/tutorial/jsf-intro.htm
[2] https://docs.oracle.com/javaee/7/tutorial/persistence-intro.htm

- **Enterprise Java Bean (EJB)**[3]: é um componente da plataforma Java EE que roda em um servidor de aplicação, tal como Glassfish, Wildfly, Tomcat, etc. Seu principal objetivo consiste em fornecer um desenvolvimento rápido e simplificado de aplicações Java, com base em componentes distribuídos, transacionais, seguros e portáveis;

- **CDI**[4]: é um framework de injeção de dependências que foi incluído no Java EE 6. Com o CDI habilitado no projeto, todas as classes que atendam a determinadas regras podem ser chamadas de *Managed Beans*.[5] Classes que recebem tal característica são disponibilizadas para outras classes automaticamente pelo CDI, não sendo necessário criar instâncias de tais classes ou recebê-las como parâmetro.

2.2.2 Model View Controller (MVC)

Dentre as várias arquiteturas para se montar um sistema Web a mais utilizada é a denominada *Model View Controller*, conhecida como MVC. Ela tem como benefício principal de isolar as regras de negócio da lógica da apresentação, ou lógica do usuário. Dessa forma, caso ocorra uma mudança na regra de negócio da aplicação a lógica da apresentação não será impactada.

Ela divide a aplicação em uma estrutura ternária, interconectada, de forma a separar cada funcionalidade, aumentando o reúso do código e permitindo o desenvolvimento paralelo da aplicação. Tais estruturas são conhecidas como:

1. **Model**: é a camada que contém as entidades de domínio, de persistência do projeto bem como as regra de negócio da aplicação. Utilizamos na camada de modelo os *frameworks* JPA, EJB e CDI já apresentados;

2. **View**: é a parte visual para o usuário, ou seja, é a representação dos modelos definidos anteriormente. Nesse projeto, a visão é composta pelas páginas Web e recursos associados (imagens, folhas de estilo, scripts, etc.), gerenciados pelo JSF;

3. **Controller**: é o link entre o usuário e o sistema, em outras palavras é quem faz a união entre a interface do usuário e as regras de negócio. Para o nosso projeto, o que faz o papel de controlador é o Faces Servlet, componente oferecido pelo JSF para este fim.

[3] https://docs.oracle.com/javaee/7/tutorial/ejb-intro.htm
[4] https://docs.oracle.com/javaee/7/tutorial/cdi-basic004.htm
[5] https://docs.oracle.com/javaee/7/tutorial/cdi-basic004.htm

2.3 FrameWeb

O FrameWeb (*Framework-based Design Method for Web Engineering*) é um método de projeto para construção de sistemas de informação baseado em *frameworks*. Tal método assume que determinados tipos de *frameworks* serão utilizados durante a construção da aplicação, definindo uma arquitetura básica e propondo modelos de projeto que se aproximam da implementação do sistema usando esses *frameworks* (SOUZA, 2007).

A arquitetura proposta é baseada no padrão *Service Layer* (Camada de Serviço) (FOWLER, 2002), exibida na Figura 1.

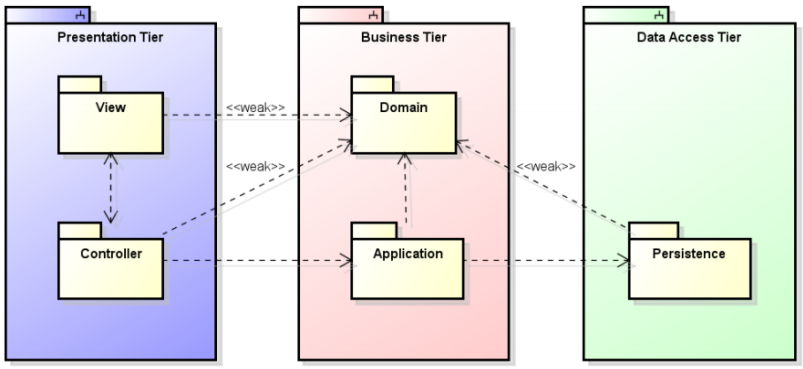

Figura 1 – Arquitetura Proposta Frame Web

Em tal arquitetura temos três camadas importantes.

- **Camada de Apresentação (*Presentation Tier*):** que apresenta as funcionalidades de interface com o usuário;
- **Camada de Negócio (*Business Tier*):** contém as funcionalidades relativas às regras de negócio;
- **Camada de Persistência (*Data Acces Tier*):** contém as funcionalidades de acesso ao banco de dados.

A linguagem **FrameWeb**, originalmente proposta por Souza (2007), propunha o uso de extensões leves da linguagem UML. Atualmente, a linguagem foi aprimorada para uma extensão completa do meta-modelo da UML, conforme proposto por Martins (2016). Tal aprimoramento foi necessário para permitir que a linguagem FrameWeb pudesse ser independente de plataforma, permitindo o uso de diversas combinações de *frameworks*, a escolha do modelador.

Os modelos definidos pelo método, utilizando a nomenclatura revisada por Martins (2016), são apresentados a seguir:

1. **Modelo de Entidades**

   É um diagrama de classes UML que representa os objetos de domínio do problema e seu mapeamento para a persistência em banco de dados relacional. Os passos para a sua construção são:

   a) 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.;

   b) Adicionar os mapeamentos de persistência.

2. **Modelo de Persistência**

   Antes de definirmos o modelo de persistência definido pelo FrameWeb, temos que entender o que é a camada de persistência. A camada de persistência de uma aplicação provê as funcionalidades básicas para o armazenamento e a recuperação dos objetos do sistema. Para isolar a lógica de negócio e o banco de dados da aplicação utilizaremos no projeto o padrão de Objeto de Acesso a Dados, também conhecido como DAO (*Data Access Object*) (BAUER; KING, 2007).

   O padrão DAO define uma interface de operações de persistência, incluindo métodos para criar, recuperar, alterar, excluir e diversos tipos de consulta, relativa a uma particular entidade persistente, agrupando o código relacionado à persistência daquela entidade (BAUER; KING, 2007).

   Com as definições acima podemos dizer que o Modelo de Persistência é um diagrama de classes UML que representa as classes DAO existentes, que pertencem ao pacote de persistência da aplicação. Esse diagrama guia a construção das classes DAO apresentando para cada classe de domínio que necessita de acesso a dados, uma interface e uma classe concreta DAO que implementa a interface.

   Os passos para a construção desse modelo são:

   a) Criar as interfaces e implementações concretas dos DAOs base;

   b) Definir quais classes de domínio precisam de lógica de acesso a dados e, portanto, precisam de um DAO;

   c) 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.

3. **Modelo de Navegação**

   É um diagrama de classe UML que representa os diferentes componentes que formam a camada de Lógica de Apresentação como páginas Web, formulários HTML e classes de ação do *framework Front Controller*.

Os passos para a construção dos Modelos de Navegação são:

a) Analisar os casos de uso modelados durante a Especificação de Requisitos, definir a granularidade das classes de ação e criá-las, definindo seus métodos. Utilizar, preferencialmente, nomes que as relacionem com os casos de uso ou cenários que representam;

b) Identificar como os dados serão submetidos pelos clientes para criar as páginas, modelos e formulários, adicionando atributos à classe de ação;

c) Identificar quais resultados são possíveis a partir dos dados de entrada, para criar as páginas e modelos de resultado, também adicionando atributos à classe de ação.

d) Analisar se o modelo ficou muito complexo e considerar dividi-lo em dois ou mais diagramas.

4. **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. 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.

Os passos para a construção do Modelo de Aplicação são:

a) 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;

b) 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);

c) Por meio de uma leitura da descrição dos casos de uso, identificar quais DAOs são necessários para cada classe de aplicação e modelar as associações;

d) Voltar ao modelo de navegação (se já foi construído), identificar quais classes de ação dependem de quais classes de serviço e modelar as associações.

3 Especificação de Requisitos

Nessa seção abordamos os conceitos de Engenharia de Requisitos para a construção do SIGME. Ne Seção 3.1 temos uma descrição do escopo do projeto; na Seção 3.2 são apresentados os diagramas de casos de uso e na Seção 3.3 são apresentados os diagramas de classe.

Os requisitos funcionais, requisitos não funcionais, regras de negócio e o dicionário do projeto podem ser encontrados no **Documento de Especificação de Requisitos** nos Apêndices.

3.1 Descrição do Escopo

O sistema SIGME — Sistema de Informação Gerencial do Movimento Espírita — foi desenvolvido inicialmente com o intuito de facilitar algumas operações existentes na instituição Federação Espírita do Estado do Espírito Santo (FEEES). As primeiras funcionalidades implementadas foram as de cadastro básicos, sendo elas as instituições e os usuários do sistema (REGIS, 2018). Com novas análises, vimos a necessidade de incluir outros módulos ao sistema, sendo eles o de Gerenciamento de Evento e de Gestão. A Seção 3.1.1 descreve o Módulo de Gerenciamento de Eventos e a Seção 3.1.2 descreve o módulo de Gerenciamento de Gestão.

3.1.1 Módulo de Gerenciamento de Eventos

A maioria das instituições realizam eventos para a divulgação da Doutrina Espírita. Logo, nosso sistema permite que um usuário cadastrado crie um evento, escolha a instituição à qual o evento está associado, determine o endereço onde será realizado, informe a data e hora inicial do evento, estipule o preço de participação e quaisquer outras informações pertinentes. Após o evento cadastrado, os demais usuários do sistema podem escolher quais eventos eles querem se inscrever, preenchendo um formulário com informações básicas, que podem ser editadas antes da realização do mesmo. O sistema permite que os usuários vejam quais eventos eles já estão cadastrados, ou quais eles participaram. Qualquer usuário pode se inscrever num evento, porém apenas aqueles que fazem parte da gestão atual de uma instituição, já cadastrada, podem criar e/ou gerenciar eventos. Não é permitido a um usuário, com exceção do administrador, cadastrar um evento para uma instituição da qual ele não faz parte.

3.1.2 Módulo de Gerenciamento de Gestão

As instituições, atualmente, realizam eleições para determinar quem será o responsável por cada uma de suas atividades. O comum é que se tenha o presidente da instituição, o(s) vice-presidente(s), um secretário e demais diretores responsáveis por áreas. Tendo isso como base, o sistema permite que as instituições cadastrem suas gestões. Na inclusão de uma nova gestão de uma instituição já cadastrada, informações da anterior são salvas, garantindo um histórico das gestões anteriores. Como temos diversas formas de gestão/gerenciamento em cada instituição, é permitido o cadastro de diferentes tipos de gestões baseadas nos cargos, ou papéis, que os espíritas participantes daquela instituição podem possuir. O sistema possui uma gestão e cargos padrões, utilizando como exemplo a utilizada na FEEES. No entanto, novos tipos de cargos e gestões podem ser incluídos, de acordo com a necessidade de cada usuário. Uma vez cadastrada a nova gestão, quem faz parte dela pode editar suas informações, gerenciar os espíritas responsáveis por cada cargo e adicionar uma nova gestão baseada na atual.

3.2 Diagramas de Caso de Uso

O SIGME é dividido nos subsistemas apresentados na Tabela 2.

Tabela 2 – Subsistemas que compõem o SIGME.

Subsistema	Descrição
Core	Envolve todas as funcionalidades básicas do sistema, abrangendo as Instituições, Espíritas, Gestão, Tipo de Gestão, Papéis da Gestão, Região e as configurações globais que o sistema necessita para operar.
People	Envolve as funcionalidades bases para a definição das classe Espírita e Instituição. Abrange os endereços, cidades, tipos de telefone, telefone e estado.
Event	Envolve todas as funcionalidades referente a eventos do sistema, abrange os Eventos e as Inscrições feitas pelos usuários.

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. Os atores identificados no contexto deste projeto estão descritos na Tabela 3.

3.2.2 Subsistema *Core*

A Figura 2 apresenta o diagrama de casos de uso do subsistema *Core*, que envolve o módulo de gestão.

Tabela 3 – Atores que interagem com o SIGME.

Ator	Descrição
Administrador	É a pessoa responsável por administrar o sistema, tendo acesso a todas as funcionalidades do mesmo.
Espírita	Qualquer pessoa que realizar o cadastro no sistema.
Gestor	Espírita que faz parte da gestão atual de uma Instituição cadastrada no sistema.

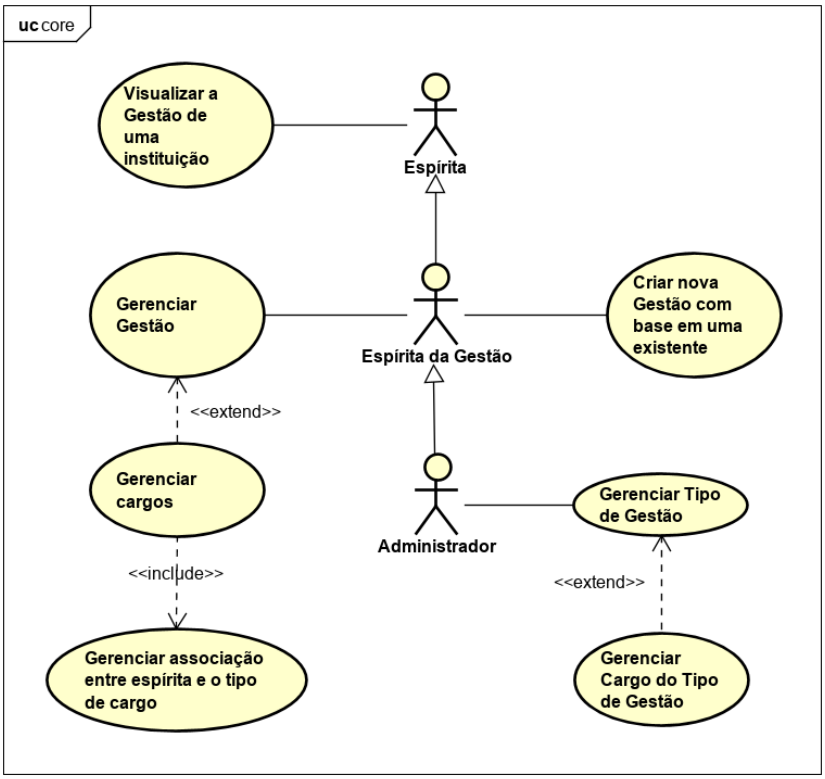

Figura 2 – Diagramas de casos de uso do subsistema *Core*.

Os casos de uso **Gerenciar Gestão**, **Gerenciar Tipo de Gestão** e **Gerenciar Cargo do Tipo de Gestão** são casos de uso de baixa complexidade, envolvendo inclusão, alteração, consulta e exclusão.

Todos os usuários do sistema podem visualizar a gestão de uma determinada instituição, com isso foi criado o caso de uso **Visualizar a Gestão de uma Instituição**. O usuário deverá selecionar a instituição desejada sendo exibida uma lista contendo todas as gestões cadastradas para a instituição, ordenadas da mais recente para a mais antiga.

É permitido que o administrador do sistema, ou o gestor da instituição, possa criar uma nova gestão com base em uma anterior, de forma a agilizar o processo. Para isso, foi criado o caso de uso **Criar uma nova gestão com base em alguma existente**. Dessa forma deve-se escolher qual a gestão anterior a ser copiada e o sistema irá criar uma gestão nova com a mesma estrutura, cargos e espíritas escolhidos.

O administrador, ou o gestor, podem modificar os espíritas ou cargos da uma gestão já inserida. Com isso, criou-se o caso de uso **Gerenciar Cargos** onde, dada uma gestão já criada, o usuário poderá criar um novo cargo, associar um cargo já existente, escolher o espírita para tal cargo, modificar o espírita de determinado cargo.

3.2.3 Subsistema *Event*

O diagrama de caso de uso para o subsistema *Event* é exibido na Figura 3.

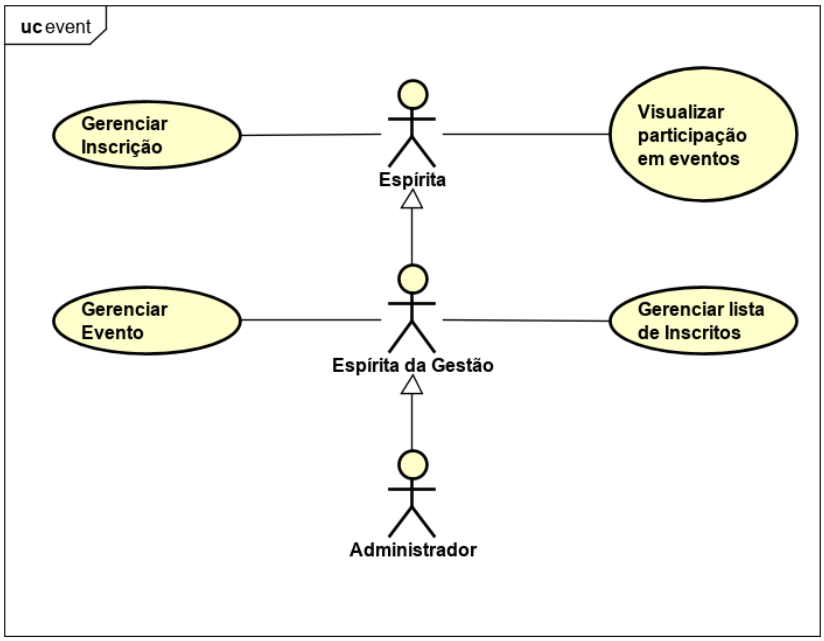

Figura 3 – Diagramas de classe do subsistema *Event*.

O caso de uso **Gerenciar Evento**, é o único caso de uso de baixa complexidade (cadastro) para esse subsistema. Abaixo descreveremos os casos de uso mais complexos.

Um espírita pode se inscrever, ou gerenciar uma inscrição, em um evento criado. Para isso criou-se o caso de uso **Gerenciar Inscrição num Evento**, possibilitando que o usuário realize a sua inscrição (informando alguns dados para o mesmo), edite uma inscrição ou cancele a sua inscrição.

Um espírita deve ter acesso a todos os eventos que ele já se inscreveu. Para isso foi criado o caso de uso **Visualizar Participação em Eventos** de forma a exibir uma lista com todos os eventos que o usuário já participou, ou está inscrito.

O administrador do sistema, ou o espírita da gestão atual, à qual o evento faz parte, deve ter a lista de espíritas que estão inscritos em determinado evento, podendo imprimir os crachás para os mesmos. O caso de uso **Gerenciar Lista de Inscritos**, abrange tal funcionalidade.

3.3 Diagrama de Classes

As seções 3.3.1 e 3.3.2 a seguir exibem os diagramas de classes criados para cada novo módulo do sistema. Como estamos aprimorando um sistema já existente, só detalharemos as classes que foram criadas especificamente para atender a esses novos módulos. Classes já criadas anteriormente são referenciadas com o padrão NomePacote::NomeClasse, por exemplo a classe Espírita - que faz parte do pacote Core - é referenciada como Core::Espírita.

3.3.1 Subsistema *Core*

Cada Gestão deve ter uma instituição à qual faz parte e um determinado tipo de gestão, representado pela classe TipoGestão. Os tipos de cargos existentes, representado pela classe TipoCargo, são definidos de acordo com o tipo de gestão. A gestão pode ter algum cargo, ou não. Por fim, os cargos de uma gestão, representado pela classe CargoGestao, tem uma relação com a gestão ao qual está sendo inserido, ao tipo de cargo e ao espírita que terá o cargo específico. Isso é representado na Figura 4.

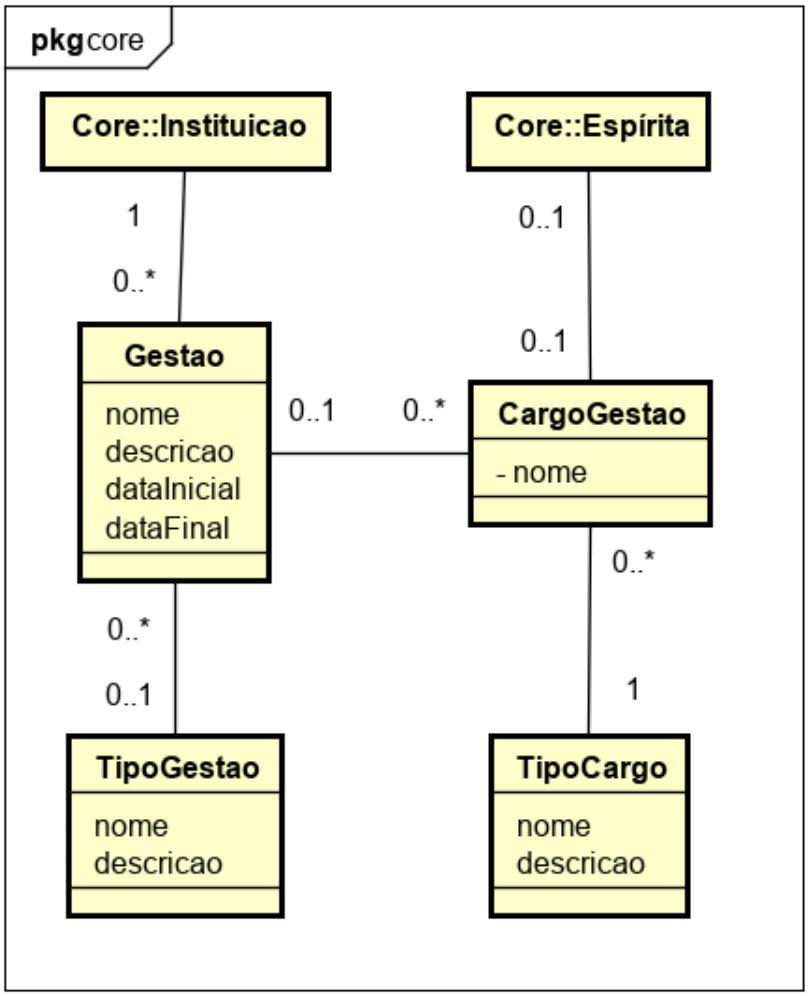

Figura 4 – Diagramas de classe do subsistema *Core*.

3.3.2 Subsistema *Event*

Um evento, representado pela classe Evento, deve possuir os dados básicos, tai como: nome, descrição, data de início, data de término, data de início e término do período de inscrição, entre outras. O evento deve estar vinculado a uma regional (classe Regional), a uma instituição (classe Instituição) e deve possui o espírita responsável pelo mesmo (classe Espírita). Os demais espíritas que se inscreverem no evento, serão associados ao mesmo pela classe Inscricao, que conterá todos os dados necessários para a confecção do crachá e demais dados. Isso é representado na Figura 5.

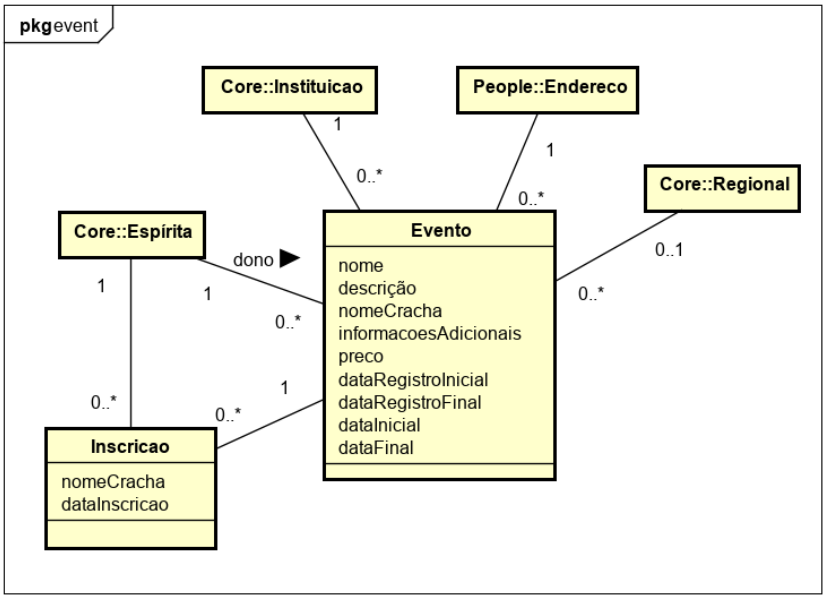

Figura 5 – Diagramas de classe do subsistema *Event*.

Maiores informações sobre o diagrama de classes poderão ser consultados do **Documento de Especificação de Requisitos**, disponível nos Apêndices ao final dessa monografia.

4 Projeto Arquitetural e Implementação

Essa etapa do projeto consiste na especificação detalhada das tecnologias a serem utilizadas para a solução proposta. Este capítulo está dividido nas seguintes seções:

- Na Seção 4.1 apresentamos a arquitetura do sistema;
- Na Seção 4.2 apresentamos o *framework nemo-utils*;
- Na Seção 4.3 apresentamos o modelos FrameWeb identificados;
- Na Seção 4.4 apresentamos o sistema.

4.1 Arquitetura

O projeto possui 2 módulos principais, a saber:

- **br.org.feees.sigme.core:** módulo que descreve os dados centrais do sistema. Neste módulo acrescentamos as classes de gestão e tipo de gestão, além de modificações nas classes de especialista e instituição, as quais já faziam parte da aplicação;
- **br.org.feees.sigme.event:** módulo novo, criado para agregar as classes referentes ao cadastro e gestão dos eventos.

A divisão do projeto, dentro dos módulos descritos acima, constitui nas 3 camadas apresentadas na Seção 2.3 (Figura 1) quando apresentamos o FrameWeb. A primeira camada, denominada Camada de Apresentação, contém os pacotes de Visão (**view**) e Controle (**controller**). A segunda camada, denominada Camada de Negócios, contém os pacotes de Domínio (**domain**) e o de Aplicação (**application**). E a terceira camada, denominada Camada de Negócios contém o pacote de persistência (**persistence**). Existe ainda um outro pacote, referente às exceções lançadas. Tal pacote não tem correspondência com o modelo FrameWeb descrito e por conta disso não está nas camadas mencionadas. A seguir detalharemos um pouco mais cada camada.

4.1.1 Camada de Apresentação

Tal camada possui os pacotes de Visão e Controle (*controller*). Ela é responsável pela exibição e navegação do sistema, de acordo com as ações do usuário. As páginas Web, que fazem parte do pacote de visão da aplicação, estão organizadas dentro de uma pasta denominada *WebContent* que possui uma subpasta de acordo com os módulos principais do

sistema e tais pastas possuem uma subpasta de acordo com o caso de uso correspondente. No caso de casos de uso de cadastro, as pastas possuem o nome da entidade de domínio manipulada naquele cadastro. A Figura 6 exibe em mais detalhes tal configuração.

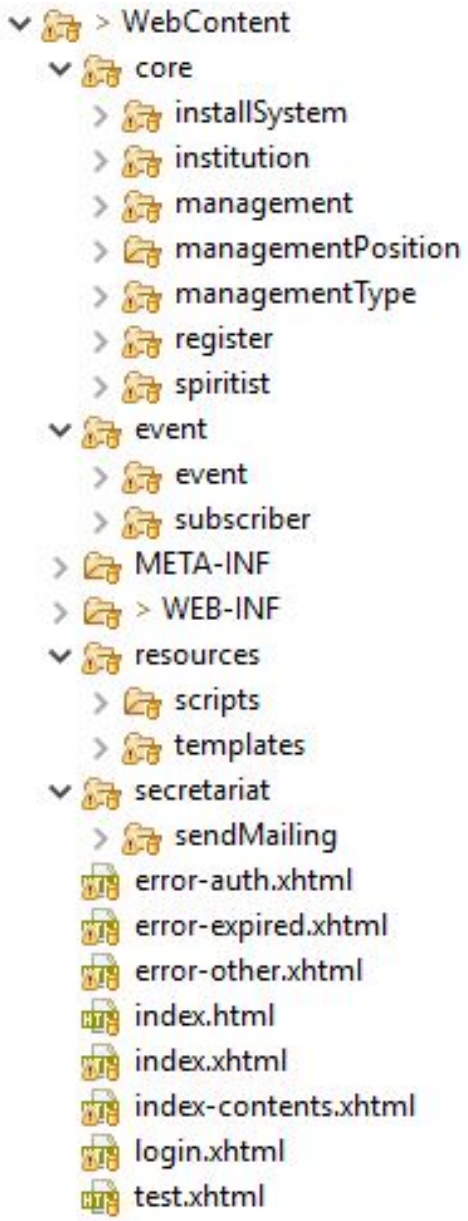

Figura 6 – Arquivos da pasta WebComponent

Cada pasta referente a um cadastro de entidade de domínio possui no mínimo dois arquivos, denominados *form.xhtml* e *list.xhtml*. Estes dois arquivos utilizam um decorador padrão do projeto denominado *decorator.xhtml*. A utilização de um decorador garante o reuso de um código e uma padronização, o que melhora a manutenção do sistema.

O arquivo *form.xhtml* consiste num formulário de inserção, edição ou exclusão do registro. Além de utilizar o decorador padrão, ele utiliza um decorador específico para os formulários, também denominado *form.xhtml*. O arquivo *list.xhtml*, consiste numa listagem

de dados. Nele temos os botões de Novo Registro, Atualização, Exclusão e as outras ações mais específicas de acordo com cada página.

Além dos arquivos mencionados, temos os arquivos que tratam outras situações específicas, sendo eles:

- **error-auth.xhtml:** trata erros de autenticação;
- **error-expired.xhtml:** trata erros de expiração da sessão de login;
- **error-other.xhtml:** qualquer outro tipo de erro apresentado pela aplicação;
- **login.xhtml:** tela de login da aplicação;
- **index.xhtml:** trata da tela inicial da aplicação;
- **index-contents.xhtml:** constitui numa subtela, reutilizada no arquivo *index.xhtml*.

No pacote de Controle as classes são nomeadas de acordo com o nome da entidade equivalente com o sufixo ***Controller.java**. As classes que são de cadastro (CRUD — *Create*, *Read*, *Update* e *Delete*), estendem a classe **CrudController**, que é uma classe genérica esperando um tipo T de dado que estende a classe **PersistenceObject** (classe que todas as entidades de domínio estendem por padrão). A classe **CrudController** será mais explicada na Seção 4.2 quando falarmos do *framework* nemo-utils. Tais classes são responsáveis pelo correto redirecionamento da aplicação a partir de uma ação específica do usuário nas páginas Web. A Figura 7 apresenta as classes de controle do módulo de eventos.

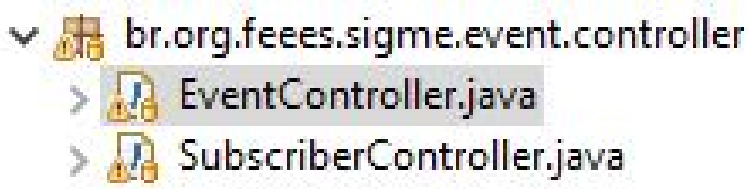

Figura 7 – Classes de controle do módulo de eventos.

4.1.2 Camada de Negócio

Conforme dito anteriormente, a camada de negócios compreende os pacotes de domínio (**domain**) e de aplicação (**application**).

Nos pacotes de domínio temos as entidades de negócio do sistema. Tais classes estendem a classe **PersistenceObject** do *framework* nemo-utils e será explicada na Seção 4.2. Tais classes possuem os atributos necessários para representar a entidade de negócio bem como o mapeamento ao banco de dados, através das anotações fornecidas pelo

pacote **javax.persistence**. Para maior manutenção do código e aderindo boas práticas de programação, tais classes possuem os *setters* e *getters* de cada atributo, garantindo o encapsulamento dos dados. A Figura 8 apresenta as classes de domínio do módulo de eventos.

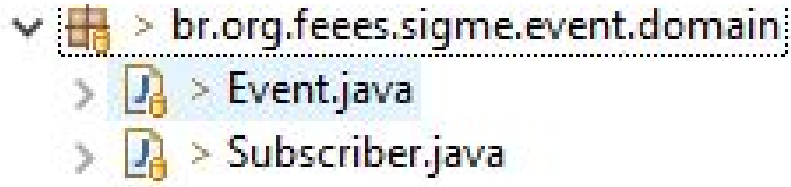

Figura 8 – Classes de domínio do módulo de eventos.

Nos pacotes de aplicação temos as entidades as classes e interfaces que realizam as ações e validações do sistema. As classes desse pacote são, normalmente, injetadas nas classes da camada de controle e utilizam as classes da camada de acesso a dados para realizar a persistência ou leitura dos dados. As interfaces possuem seu nome definido inicialmente com o nome da classe de domínio a qual ela se referencia juntamente com o sufixo **Service** e estendem a classe **CrudService** do *framework* nemo-utils. As classes possuem o mesmo nome que a interface acrescentadas do sufixo **Bean**. Elas estendem a classe **CrudServiceBean** do *framework* nemo-utils e implementam a interface correspondente. A Figura 9 apresenta as classes de aplicação do módulo de eventos.

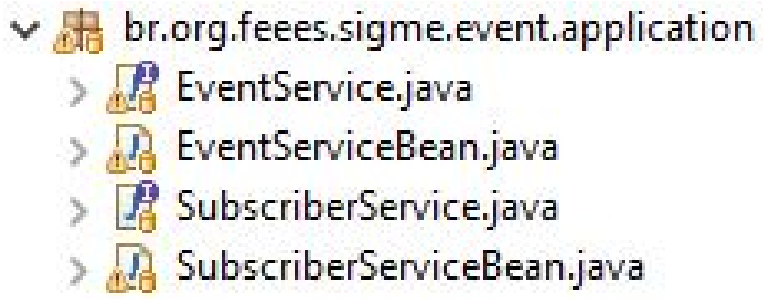

Figura 9 – Classes de aplicação do módulo de eventos.

4.1.3 Camada de Acesso a Dados

A camada de acesso a dados possui somente o pacote de persistência de dados (**persistence**). Tal pacote possui as classes e interfaces de persistência de dados, denominadas DAOs (ver item Modelo de Persistêcia da Seção 2.3). O nome das classes e interfaces têm ligação com o nome da entidade de negócio à qual se referenciam, tendo como sufixo ***JPADAO** e ***DAO**, respectivamente. As interfaces estendem a interface **BaseDAO** do *framework* nemo-utils, e as classes implementam tais interfaces e estendem a classe **BaseJPADAO** do *framework* nemo-utils.

Para realizar a comunicação com o banco é utilizada a API de Critérios (*Cri-*

teria API) do JPA, que necessita de metamodelos estáticos das classes mapeadas na camada de domínio. Por isso foi integrado ao arquivo pom do projeto a biblioteca **org.hibernate.hibernate-jpamodelgen**, que gera tais arquivos ao compilar o projeto. A nomenclatura de tais metamodelos segue o padrão de nome da entidade seguido do sufixo _.

4.2 *Framework* nemo-utils

O *framework* nemo-utils é um repositório contendo classes utilitárias para auxiliar no desenvolvimento de projetos em Java, em especial Java EE. O nemo-utils foi utilizado para diminuir a repetição de código entre as classes implementadas. Atualmente é composto pelos seguintes pacotes:

- **nemo-utils-java**: para projetos Java genéricos.
- **nemo-utils-jee-wp**: para projetos utilizando Java EE Web Profile.
- **nemo-utils-jee-full**: para projetos utilizando Java EE Full Profile.

Por se tratar de um projeto com ênfase na Web, incluímos nas dependências do projeto os pacotes **nemo-utils-java** e **nemo-utils-jee-wp**. A Figura 10 exibe os pacotes e as classes que foram reutilizadas no desenvolvimento do sistema.

Nas próximas seções iremos explicar as classes e interfaces que foram utilizadas no desenvolvimento do projeto, definições de alguns métodos ou classes que não são relevantes para o projeto serão deixados de fora visto que não é o foco do trabalho apresentar todo o conteúdo do *nemo-utils*.

4.2.1 Pacote de Domínio

O pacote de domínio do *nemo-utils* consiste na classe **DomainObjectSupport** e na interface **DomainObject**. Tal classe tem a responsabilidade de definir um identificador universal único (UUID) para cada entidade do sistema a ser utilizadas no método **compareTo()**. A sobrescrita do método **compareTo()**, bem como dos métodos auxiliares **hashCode()** e **equals()** é a solução padrão para verificar se uma entidade é igual à outra do sistema, de forma que a persistência consiga identificar se estamos realizando uma inserção ou atualização na tabela. Essas classes não são estendidas diretamente pelas classes de domínio do sistema e sim pela classe **PersistentObjectSupport** definida na camada de persistência do *framework*.

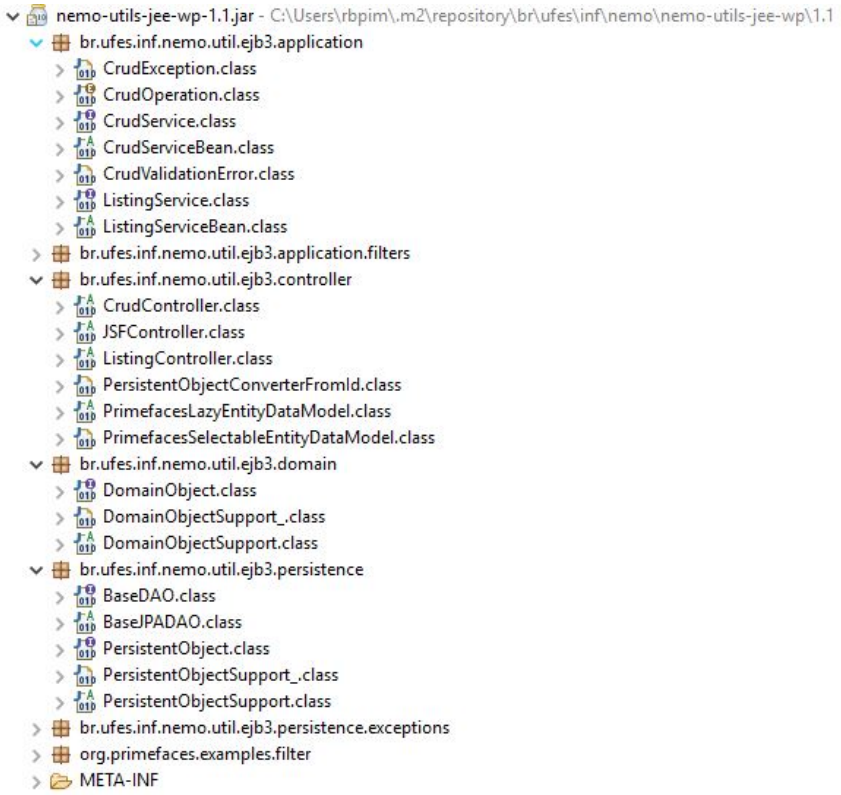

Figura 10 – Pacotes e classes reutilizados do nemo-utils.

4.2.2 Pacote de Persistência

O pacote de persistência possui a classe **PersistentObjectSupport** que é a classe pai de todas as entidades de domínio do sistema, além da classe **BaseJPADAO** que é a classe que define uma implementação padrão para as ações de acesso ao banco, tais como inserção, atualização, remoção e obtenção de dados e, portanto, é a classe pai de todos as classes de persistência.

A classe **PersistentObjectSupport** estende a classe **DomainObjectSupport**, de forma a ter as vantagens de comparação descritas na Seção 4.2.1, e implementa a interface **PersistentObject**, que expõe métodos para obtenção dos atributos **id**, **version** e a verificação pelo método **isPersistent()** (para verificar se a entidade já foi persistida).

O atributo **id** é marcado pela anotação **@Id**, relacionando esse atributo com a coluna do banco que representa a chave primária da tabela. O atributo **version** é marcado com a anotação **@Version**, informando ao *framework* de mapeamento objeto/relacional que use esse atributo para o tratamento de concorrência de transação. O método **isPersistent()** é utilizado pela classe **BaseJPADAO** no método de persistência de dado: se o retorno desse método for verdadeiro, a classe já foi persistida e portanto possui um **id** determinado, a aplicação realiza uma atualização de dados do registro do banco, caso contrário realizamos uma inserção de dados.

A classe **BaseJPADAO** é uma classe parametrizada, também chamada de genérica,

que espera a definição sobre qual o tipo de dado que será utilizado em suas operações. Tal definição está exibida na Figura 11, onde vemos que um tipo de dado T (que é do tipo **PersistentObject**) é esperado.

```
public abstract class BaseJPADAO<T extends PersistentObject> implements BaseDAO<T> {
    /** Serialization id. */
    private static final long serialVersionUID = 1L;
```

Figura 11 – Definição da classe BaseJPADAO

Ao realizar tal definição, não é preciso se procupar em passar o tipo de dado para os métodos definidos na classe, basta que na utilização da classe **BaseJPADAO** seja informado qual o tipo de parâmetro T que estamos nos referindo. Por exemplo, na definição da classe **EventJPADAO**, que é a classe onde temos a persistência dos dados do tipo **Evento**, informamos que o seu pai seria uma classe do tipo **BaseJPADAO<Event>** e portanto não foi mais necessário nos preocupar em passar a classe que estava sendo persistida para o método **save()**, uma vez que o parâmetro da classe **BaseJPADAO** já tinha sido definido.

A classe **BaseJPADAO** implementa a interface **BaseDAO**, que é uma interface genérica, com a mesma definição do tipo de dado **T extends PersistentObject**. Ela expõe para as classes que a injetarem os métodos básicos implementados na classe que a implementa. Alguns métodos que foram reutilizados:

- **retrieveAll()**: retorna uma lista de todos os registros do banco de dados;
- **getOrderList()**: estabelece uma ordem específica para as listagem retornadas;
- **retrieveById()**: obtém um determinado registro a partir do id informado;
- **save()**: insere dados no banco de dados;
- **delete()**: exclui dados do banco de dados;
- **merge()**: atualiza dados no banco de dados.

4.2.3 Camada de Aplicação

Em tal pacote temos as classes **CrudServiceBean**, que implementa a classe **CrudService** e estende a classe **ListingServiceBean**, além da classe de exceções básicas, nomeada como **CrudException**, que pode ser lançada em qualquer operação de CRUD, promovendo a implementação base de como um erro de validação irá aparecer na tela.

Todas as classes do pacote de aplicação do sistema estenderam a classe **CrudServiceBean** definindo qual a entidade de domínio que tinham responsabilidade, visto que tal classe de cadastro (CRUD) é uma classe genérica. Os métodos para validar se

ocorreu um erro durante uma operação de salvar, atualizar ou excluir são chamados **validadeCreate()**, **validateUpdate()** e **validateDelete()**, respectivamente. As outras operações dessa classe são específicas para as ações de salvar, atualizar e/ou excluir.

As funcionalidades de listagem estão implementadas na classe **ListingServiceBean**. As operações de obter o DAO correspondente para as chamadas do banco de dados, as operações de obter a listagem de registros sem filtro e com filtro e ainda a funcionalidade de listar de uma forma *lazy* — a ser utilizado na listagem de objetos de domínio —, são encontradas nos métodos **getDAO()**, **list()**, **filter()** e **fetchLazy()**, respectivamente.

4.2.4 Camada de Controle

O pacote **controller** tem a classe **CrudController** como a classe pai de todos as classes de controle do sistema, sendo também uma classe genérica. Os métodos dessa classe definem as funcionalidades de uma tela de cadastro (CRUD) básico, sendo os métodos **update()**, **save()** e **delete()** os responsáveis por chamar as ações de atualizar, salvar e excluir uma entidade, respectivamente. É nessa classe que os métodos de validação, definidos nas classes de serviço, são chamados. Caso a validação da entidade operação falhe por qualquer motivo uma mensagem de erro é exibida na página Web correspondente. Tal classe possui um método abstrato **createNewEntity()** que todas as classes não-abstratas que a estendem devem obrigatoriamente implementar, sendo responsável por inicializar a entidade e definir as propriedades auxiliares das telas de listagem e/ou criação.

Essa classe estende de outra classe existente desse pacote, chamada **ListingController**, tal classe é uma classe genérica também e é responsável por prover as funcionalidades de listagem básicas. Esta classe, por sua vez, estende a classe **JSFController**, responsável por prover funcionalidades básicas de internacionalização. Alguns métodos da classe **ListingController** que merecem atenção são:

- **getViewPath()**: se tal método não for sobrescrito, ele tentará "adivinhar" onde está a *view* da classe de controle de uma certa entidade (abaixo da pasta *WebContent*). É por conta dessa implementação que as pastas descritas na Seção 4.1.1 estão definidas dessa maneira. Basicamente, se a classe de controle está dentro de um pacote que segue a ordem **br.com.package.controller.EntidadeController**, tal método define a view como */package/entidade/*;
- **retrieveEntities()**: obtém, a partir do serviço de listagem definido, uma coleção de registros, respeitando a paginação. Em outras palavras, esse é o método que disponibiliza a lista para a *view*;
- **getLazyEntities()**: é o método responsável por carregar uma coleção de registros de forma *lazy*, a ser utilizada pelo componente tabela de dados (*data table*). Tal

método reutiliza o método *retrieveEntities*;

- **initFilters()**: adiciona os filtros possíveis da tela de listagem.

Existem outras classes definidas nesse pacote, mas os apresentados acima são os mais importantes e os que mais foram reutilizados no desenvolvimento da aplicação.

4.3 Modelos FrameWeb

Nesta seção iremos apresentar os modelos criados a partir da linguagem de modelagem FrameWeb, apresentada na Seção 2.3. Nos anexos temos a modelagem para os dois módulos implementados do sistema, o de gerenciamento de eventos e o de gerenciamento de gestão. Por questões de visibilidade iremos exibir nessa seção somente os modelos de Domínio (4.3.1), Persistência, Navegação e Aplicação para o módulo de gerenciamento de eventos, os demais modelos podem ser visualizados nos anexos.

4.3.1 Modelos de Entidades

Na Figura 12 temos o modelo de entidades para o pacote **sigme.event.domain**, exibindo as classes **Event** e **Subscriber** e suas relações com as classes que já existiam na aplicação, a saber **Institution**, **Address**, **Spiritist** e **Regional**. As classes e os métodos do nemo-utils não foram incluídas nos modelos.

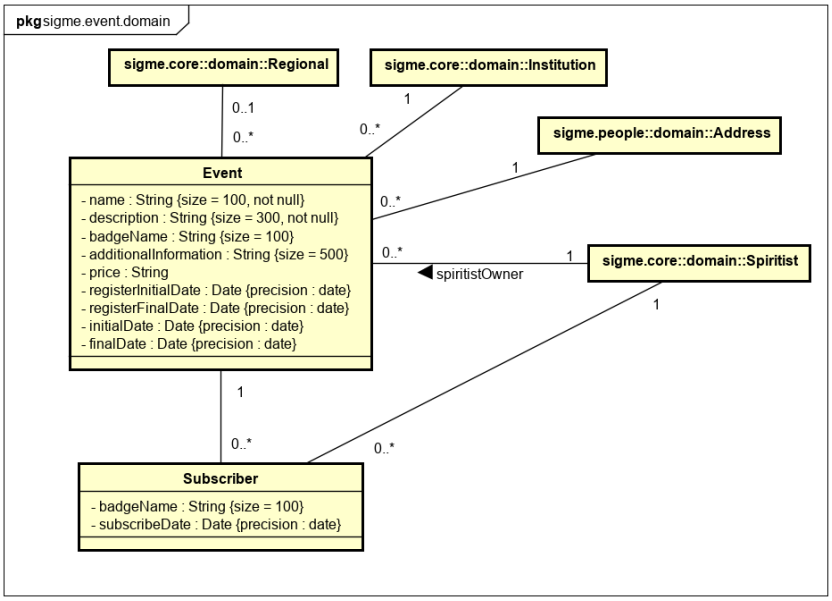

Figura 12 – Modelo de entidades do módulo de eventos.

4.3.2 Modelos de Persistência

Na Figura 13 temos o modelo de persistência para o pacote **sigme.event.persistence**, exibindo as classes **EventJPADAO** e **SubscriberJPADAO**, juntamente com as suas interfaces **EventDAO**, **SubscriberDAO** e os métodos criados para ações específicas dessas classes. As classes e os métodos do nemo-utils não foram incluídas nos modelos.

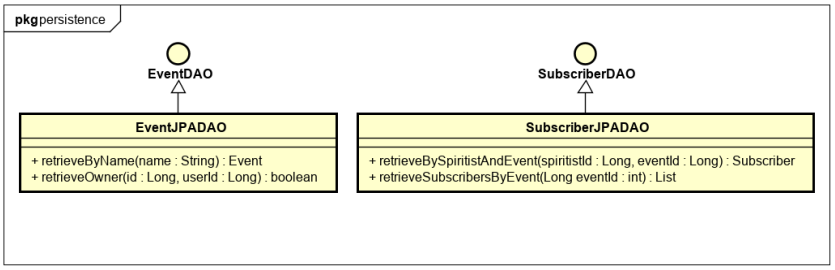

Figura 13 – Modelo de persistência do módulo de eventos.

4.3.3 Modelos de Aplicação

A Figura 14 mostra o modelo de aplicação para o pacote **sigme.event.application**. A classe **EventServiceBean** é utilizada pela classe de controle **EventController**, disponibilizada a partir da injeção da interface **EventService**, e não possui métodos específicos além dos de validação de dados já apresentados nos anexos. A classe **SubscriberServiceBean** possui os métodos de obter uma lista de todos os inscritos em um determinado evento, e outro método para obter uma inscrição dado o identificador do usuário e de um evento.

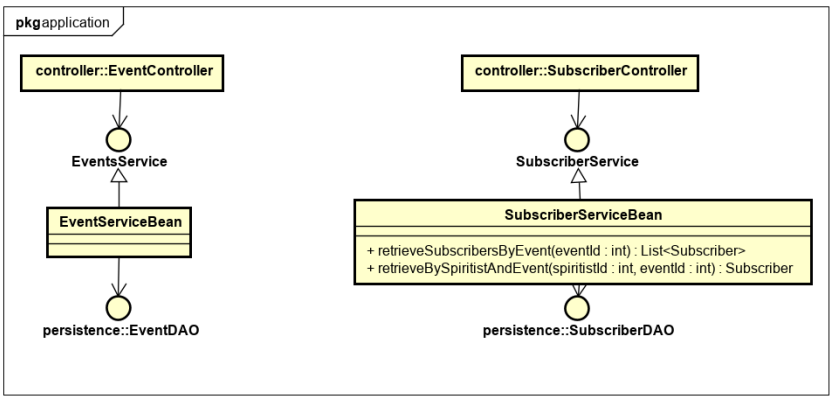

Figura 14 – Modelo de Aplicação do módulo de eventos.

4.3.4 Modelo de Navegação

Conforme pode ser visto nos anexos, a aplicação como um todo possui um modelo de navegação para os CRUDs básicos, onde todas as telas de listagem possuem as ações de

incluir, alterar, excluir e visualizar. Para simplificar o modelo vamos exibir somente ações que diferem do básico. A tela de listagem de eventos lista todos os eventos cadastrados no sistema, dando ao usuário identificado as ações de se inscrever no evento selecionado ou de gerenciar o mesmo. Esta segunda funcionalidade é acessível somente se o usuário estiver cadastrado na gestão atual da instituição à qual o evento faz parte. Ao selecionar a ação de se inscrever num evento, o sistema preenche automaticamente os dados necessários para realizar a inscrição (que fica visível na tela de inscrições do usuário). Ao selecionar a ação de gerenciar um evento, o sistema é levado para uma tela de gerenciamento de inscrições, onde é exibida a listagem de todos os espíritas inscritos, informando se a inscrição está confirmada ou não. A Figura 15 exibe o modelo de navegação para as ações descritas acima.

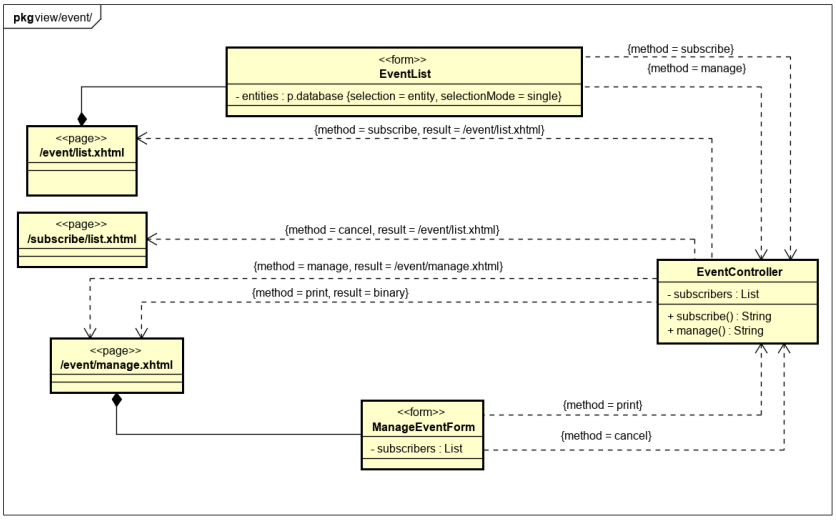

Figura 15 – Modelo de Navegação do módulo de eventos

As funcionalidade de inscrições não possui nenhuma ação diferente e pode ser descrita pelo modelo de navegação de CRUD, que pode ser visualizada nos anexos.

4.4 Apresentação do Sistema

Nesta seção iremos apresentar as telas do módulo de Gerenciamento de Eventos desenvolvidas no projeto. Ao acessar o sistema nos é apresentada a tela inicial, dando uma breve descrição do sistema. Esta tela já estava desenvolvida e a apresentamos só para conhecimento. A tela inicial pode ser visualizada na Figura 16.

Após realizar login na aplicação o menu lateral do usuário é modificado, apresentando todas as entradas possíveis para o usuário. Tal menu não é alterado quando o usuário muda para outra tela de forma proporcionar uma boa navegabilidade pela aplicação. A Figura 17 exibe tal menu.

Figura 16 – Tela inicial do SIGME.

Figura 17 – Menu lateral do usuário logado no SIGME.

Para acessar a tela de eventos cadastrados, o usuário deve selecionar a opção **Listar Eventos**. A listagem de todos os eventos cadastrados é exibida informando dados relevantes tais como o nome do evento, o período válido de inscrição e o período no qual o evento irá ocorrer. A Figura 18 exibe a listagem de evento.

O único botão habilitado, sem a seleção de um evento, é o botão de criar um **Novo Evento** no sistema. Os demais botões de **Consultar**, **Gerenciar**, **Alterar**, **Increver** e **Excluir** só serão ativados após o usuário selecionar um evento na listagem. Os botões de **Novo**, **Consultar** e **Alterar** levam o sistema para a mesma tela de formulário, exibida nas figuras 19 e 20, com a diferencia que o botão **Consultar** apresenta os dados de forma informativa, não permitindo sua alteração.

Na tela de listagem, caso o usuário queira se inscrever em um evento, basta selecionar o evento desejado e clicar no botão de **Inscrever**. O SIGME automaticamente preenche o campo **Nome do Crachá** com o nome do usuário, preenchido no cadastro de Espíritas. Ao realizar tal ação o sistema direciona o usuário para a listagem dos eventos em que o usuário se cadastrou. A tela de inscrições do usuário é exibida na Figura 21.

Figura 18 – Tela de listagem dos Eventos do Sigme

Figura 19 – Parte 1 do Formulário de Cadastro de Evento no SIGME.

Tal tela foi considerada como um CRUD, podendo o usuário visualizar, alterar ou excluir os dados de um evento cadastro. Excluir equivale a cancelar sua inscrição no evento. Caso o usuário queira se inscrever num evento que já sabe que está ativo, o mesmo pode selecionar o botão **Novo**, na tela de **Minhas Inscrições**. Após tal ação o formulário da Figura 22 é apresentado, exibindo os campos de **Nome para Crachá** e uma lista com os eventos ativos no momento. Caso o usuário tente se inscrever em um evento no qual ele já está inscrito, o sistema irá bloquear a inscrição informando que tal usuário já está cadastrado no evento, conforme Figura 23.

Por último, caso o usuário tenha permissão para gerenciar um evento, ao selecionar a ação de **Gerenciar** na tela de listagem de eventos, o mesmo é redirecionado para uma tela de gerenciamento no qual ele pode ver os espíritas que foram inscritos e pode solicitar uma exportação para PDF de todas as inscrições. Tal tela é exibida na Figura 24.

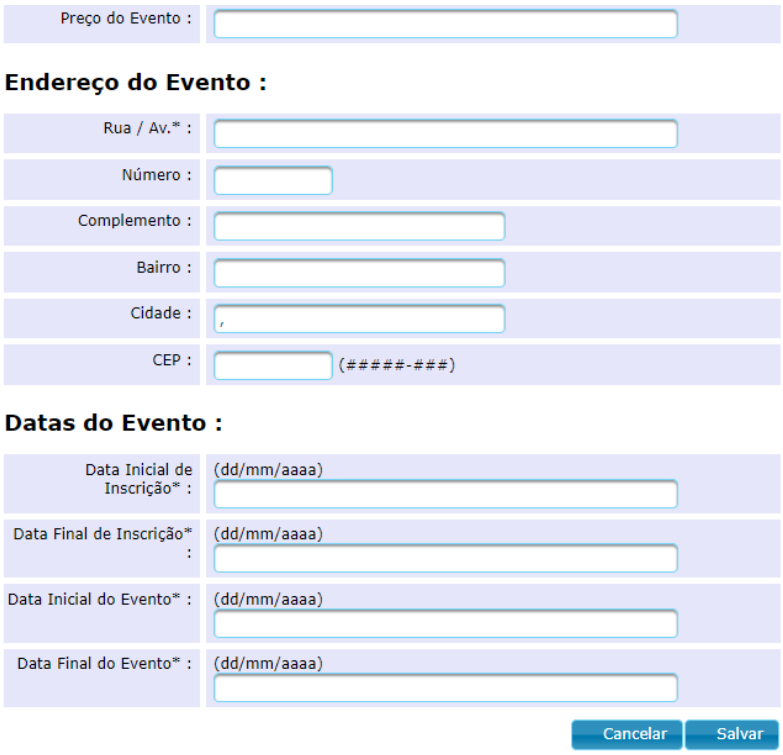

Figura 20 – Parte 2 do Formulário de Cadastro de Evento no SIGME.

Figura 21 – Listagem das Inscrições do usuário no SIGME.

Figura 22 – Formulário de Inscrição em evento no SIGME.

Figura 23 – Inscrição com falha de Espírita já cadastrado no evento.

Figura 24 – Tela de gerenciamento de evento do SIGME.

5 Conclusão e Considerações Finais

A inclusão dos dois módulos no SIGME, o de Gerenciamento de Eventos e de Gestão, acrescenta funcionalidades benéficas para os usuários. Com o módulo de eventos, o usuário tem acesso às funcionalidades de criação e/ou inscrição de um evento, facilidade de impressão da lista de inscritos, pode visualizar os eventos espíritas que estão acontecendo no momento, entre outros.

O módulo de gestão traz funcionalidades benéficas tanto para quem administra as instituições quanto para os demais usuários do sistema. Para quem administra, temos as novas funcionalidades de associar um espírita a um determinado cargo e a possibilidade de criar novos cargos para um tipo de gestão. Os demais usuários, que podem frequentar ou não a instituição, ganham mais clareza sobre quem é o responsável por determinada área.

Ainda existem aspectos a serem aprimorados nesses dois módulos como, por exemplo:

- Comunicação com um sistema de agenda, tais como o disponibilizado pelo Google, para que seja salvo automaticamente um evento na agenda do usuário;
- Aprimorar a visualização dos eventos para a forma de calendário, dando uma maior clareza sobre quando um evento vai ocorrer;
- Incluir um sistema de pagamento para o processo de inscrição;
- Permitir a customização do formulário de inscrição, incluindo campos e modificando o que está escrito no mesmo;
- Habilitar o bloqueio de determinadas operações de acordo com o perfil do usuário numa gestão;
- Criar módulos específicos para cada cargo da gestão, permitindo que o sistema unifique as informações da instituição, tais como financeira, gestão pessoal, etc.

Quanto aos objetivos deste trabalho, os mesmos foram cumpridos em sua totalidade. Foi realizado um processo de levantamento e especificação de requisitos seguindo os padrões de Engenharia de Software estudados no decorrer da graduação. Isso pode ser visualizado no Documento de Requisitos, presente nos apêndices, onde estão documentados os requisitos funcionais, não funcionais, as regras de negócios e os digramas de caso de uso da aplicação. Após a criação de tal documento, seguimos com a criação do Documento de Projeto do sistema, criando os modelos de Entidades, Persistência, Navegação e Aplicação, conforme

orientado pela linguagem FrameWeb. O sistema foi implementado utilizando a plataforma Java EE, estudada especificamente na disciplina de Desenvolvimento Web, e as demais implementações foram baseadas nos conhecimentos e nas boas práticas apresentadas nas disciplinas de Programação II, Estrutura de Dados I e Banco de Dados.

Visando uma facilidade de configuração do sistema, poderia ser estudado a atualização do sistema atual para a utilização de uma arquitetura baseada no Spring Boot e no Angular, ou React. Como tais *frameworks* não são tão engessados como a arquitetura Java EE, alguns acreditam que tal arquitetura seria melhor nesse aspecto. Ressaltamos aqui que teria a necessidade de atualização de todo o *core* do sistema, visto que o *nemo-utils* disponibiliza classes específicas para as configurações do Java EE.

Quanto ao método FrameWeb em si, pontuo que a utilização dele no projeto facilitou o desenvolvimento. A criação dos modelos numa fase de projeto garantiu que o desenvolvedor tivesse mais consciência do que tinha que ser implementado e qual a melhor forma de relacionar as camadas. Os pontos negativos a serem considerados são: (i) a necessidade de atualização desses modelos quando ocorre alguma mudança no entendimento da funcionalidade, ou mudanças na forma da exibição dos dados, (ii) necessidade de um conhecimento técnico nos *frameworks* sendo utilizados para quem cria os modelos do método. O conhecimento técnico não é obrigatório, mas garante uma melhor precisão na modelagem. De modo geral, a utilização do método é proveitosa.

Referências

BARCELOS, M. P. Engenharia de Software. *Engenharia de Software*, p. 218, 2018. Disponível em: <https://pt.wikipedia.org/wiki/Engenharia{_}de{_}so>. Citado 4 vezes nas páginas 14, 15, 16 e 17.

BAUER, C.; KING, G. *Java Persistence with Hibernate*. [S.l.]: Manning, 2007. Citado na página 21.

FOWLER, M. *Patterns of Enterprise Application Architecture*. [S.l.]: Inc. Boston, 2002. Citado na página 20.

MARTINS, B. F. *Evolução do Método FrameWeb para o Projeto de Sistemas de Informação Web Utilizando uma Abordagem Dirigida a Modelos*. Vitória, ES, Brasil, 2016. Citado na página 20.

REGIS, R. V. H. *Desenvolvimento do Núcleo do Sistema de Informação Gerencial do Movimento Espírita*. Vitória, ES, Brasil, 2018. Citado 2 vezes nas páginas 11 e 23.

SOUZA, V. E. S. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. [S.l.], 2007. Citado 3 vezes nas páginas 12, 14 e 20.

Apêndices

Documento de Requisitos

Sigme - Sistema de Informação Gerencial do Movimento Espírita

Registro de Alterações:

Versão	Resposável	Data	Alterações
1.0	Rodrigo Bittencourt Pimenta	21/06/2016	*versão inicial*
1.1	Vítor E. Silva Souza	28/06/2016	*primeira revisão*
1.2	Rodrigo Bittencourt Pimenta	28/06/2016	*alterações referente à primeira revisão*
1.3	Vítor E. Silva Souza	01/07/2016	*segunda revisão*
1.4	Rodrigo Bittencourt Pimenta	25/10/2016	*Alterações referente à segunda revisão*
1.5	Vítor E. Silva Souza	08/11/2016	*terceira revisão*
1.6	Rodrigo Bittencourt Pimenta	08/11/2016	*Alterações referente à terceira revisão*
1.7	Vítor E. Silva Souza	18/11/2016	*quarta revisão*
1.8	Rodrigo Bittencourt Pimenta	18/05/2017	*Alterações referente à quarta revisão*
1.9	Vítor E. Silva Souza	14/06/2017	*quinta revisão*
2.0	Rodrigo Bittencourt Pimenta	06/06/2019	*quinta revisão - atualização de diagramas e correções de erros*

Vitória, ES

2016

1 Introdução

Este documento apresenta os **Requisitos de Usuário** e a **Especificação dos Requisitos** do sistema Sigme - Sistema de Informação Gerencial do Movimento Espírita. A atividade de análise de requisitos foi conduzida aplicando-se técnicas de modelagem de casos de uso, modelagem de classes e modelagem de comportamento dinâmico do sistema. Os modelos apresentados foram elaborados usando a UML.

Este documento está organizado em seções, da seguinte maneira: a Seção 2 contém a descrição geral do problema, a Seção 3, as listas de **Requisitos de Usuário** levantados junto ao cliente, a Seção 4 apresenta os subsistemas identificados, mostrando suas dependências na forma de um diagrama de pacotes; a Seção 5 apresenta o modelo de casos de uso, incluindo as descrições dos atores e dos casos de uso, com seus respectivos diagramas; a Seção 6 apresenta o modelo conceitual estrutural do sistema, na forma de diagramas de classes; a Seção 7 apresenta o dicionário do projeto, contendo as definições das classes identificadas.

2 Descrição do Propósito dos Módulos

O sistema Sigme - Sistema de Informação Gerencial do Movimento Espírita foi desenvolvido inicialmente com o intuito de facilitar algumas operações existentes na instituição Federação Espírita do Estado do Espírito Santo (Feees). As primeiras funcionalidades implementadas foram as de cadastro básicas, sendo elas as instituições e os usuários do sistema. Com novas análises, vimos a necessidade de incluir outros módulos ao sistema, sendo eles o de **Gerenciamento de Evento** e de **Gestão**. Abaixo descrevemos esses módulos.

2.1 Módulo de Gerenciamento de Eventos

Grande parte das instituições realizam eventos com diversas finalidades como para a divulgação da doutrina espírita. Logo, nosso sistema deve permitir que um usuário cadastrado crie um evento, escolha a instituição à qual o evento está ligado, determine o endereço onde será realizado, informe a data e hora inicial do evento, estipule o preço de participação e quaisquer outras informações pertinentes. Após o evento cadastrado, os demais usuários do sistema podem escolher quais eventos eles querem se inscrever, preenchendo um formulário com informações básicas, que podem ser editadas antes da realização do mesmo. O sistema deve permitir que os usuários vejam quais eventos eles já estão cadastrados, ou quais eles participaram. Qualquer usuário pode se inscrever para qualquer evento, porém apenas aqueles que fazem parte da gestão atual de uma instituição já cadastrada podem criar e/ou gerenciar eventos. Não é permitido a um usuário, com exceção do administrador, cadastrar um evento para uma instituição da qual ele não faz parte.

2.2 Módulo de Gerenciamento de Gestão

As instituições, atualmente, possuem uma eleição para determinar quem será o responsável por determinadas atividades. O comum é que se tenha o presidente da instituição, o(s) vice-presidente(s), um secretário e demais diretores responsáveis por áreas. Tendo isso como base, o sistema deve permitir que as instituições cadastrem suas gestões. Caso ocorra a inclusão de uma nova gestão, de uma instituição já cadastrada, devemos salvar as informações da anterior, de forma que se tenha um histórico das gestões já existentes. Como temos diversas formas de gestão/gerenciamento em cada instituição, temos que permitir diferentes tipos de gestões baseadas nos cargos, ou papéis, que os espíritas participantes daquela instituição podem possuir. O sistema teria uma gestão e

cargos padrões, utilizando como exemplo a utilizada na Feees. No entanto, novos tipos de cargos e de gestões poderiam ser incluídos, de acordo com a necessidade de cada usuário. Uma vez cadastrada a nova gestão, quem faz parte dela pode editar suas informações, gerenciar os espíritas responsáveis por cada cargo e adicionar uma nova gestão baseada na atual, que seria incluída quando novas eleições fossem realizadas, por exemplo.

3 Requisitos de Usuário

3.1 Requisitos Funcionais

Tabela 1 – Requisitos Funcionais

Identificador	Descrição	Prioridade	Depende
RF-1	O sistema deve permitir a criação de eventos.	Alta	
RF-2	O sistema deve permitir a edição dos dados de um evento.	Média	RF-1
RF-3	O sistema deve permitir que um usuário se inscreva num evento.	Alta	RF-1
RF-4	O sistema deve exibir uma lista de inscritos para um determinado evento.	Alta	RF-1, RF-3
RF-5	O sistema deve gerar os crachás de um evento com base na lista de inscritos.	Alta	RF-1, RF-3
RF-6	O sistema deve mostrar ao usuário os eventos em que está inscrito.	Alta	RF-1, RF-3
RF-7	O sistema deve manter um histórico de eventos que o usuário participou.	Média	RF-1, RF-3
RF-8	O sistema deve permitir a criação de um novo tipo de gestão.	Alta	
RF-9	O sistema deve permitir a edição de um tipo de gestão existente.	Alta	RF-8
RF-10	O sistema deve permitir a criação de novos tipos de cargos para um tipo de gestão existente.	Alta	RF-8, RF-9
RF-11	O sistema deve permitir a edição dos tipos de cargos existentes em determinado tipo de gestão.	Alta	RF-8, RF-9
RF-12	O sistema deve permitir a criação de uma nova gestão para uma determinada instituição espírita.	Alta	
RF-13	O sistema deve permitir a edição da gestão atual de determinada instituição espírita.	Alta	RF-12
RF-14	Ao gerenciar a gestão de uma instituição, o sistema deve permitir associar o espírita da mesma com um tipo de cargo, de acordo com o tipo de gestão escolhida.	Média	RF-13

RF-15	O sistema deve permitir gerenciar as associações entre o espírita e o tipo de cargo existente na gestão atual da instituição.	Média	RF-13, RF-14
RF-16	O sistema deve manter um histórico das gestões que já existiram.	Média	RF-12, RF-13
RF-17	O sistema deve permitir a criação de uma nova gestão com base na gestão anterior.	Alta	RF-12, RF-13

3.2 Regras de Negócio

Tabela 2 – Requisitos Funcionais

Identificador	Descrição	Prioridade	Depende
RN-1	O administrador do sistema não deve ter restrições de acesso a nenhuma funcionalidade relacionada a eventos ou à gestão.	Alta	
RN-2	Deve ser exibido o nome dos espíritas que fizeram parte da gestão ainda que os mesmos não façam mais parte do sistema.	Alta	RF-16
RN-3	Somente quem participa da gestão atual da instituição pode inserir ou gerenciar uma gestão.	Alta	RF-12, RF-13, RF-17
RN-4	Somente o administrador do sistema pode criar e editar os tipos de gestões e os papéis que esse tipo possui.	Alta	RF-17
RN-5	Os espíritas devem pertencer à casa espírita em questão para que possam ser inseridos na gestão.	Alta	RF-17
RN-6	Um usuário só pode criar ou gerenciar um evento de uma instituição se o mesmo pertencer à sua gestão atual.	Alta	RF-1, RF-2, RF-4

3.3 Requisitos Não Funcionais

Tabela 3 – Requisitos Não Funcionais

Identificador	Descrição	Categoria	Escopo	Prioridade
RNF-1	A ferramenta deve estar disponível como uma aplicação Web, acessível a partir dos principais navegadores disponíveis no mercado.	Portabilidade	Sistema	Alta
RNF-2	A ferramenta dever ser de aprendizado fácil, não sendo necessário nenhum treinamento especial para seu uso.	Facilidade de Aprendizado	Sistema	Média
RNF-3	A ferramenta deve ser de fácil operação, não sendo necessário uso contínuo para uma boa operação do sistema.	Facilidade de Operação	Sistema	Alta
RNF-4	O sistema deve controlar o acesso as funcionalidades.	Segurança de acesso	Sistema	Alta
RNF-5	O sistema deve estar integrado a um sistema de correio eletrônico de modo que seja possível o envio de email automaticamente.	Interoperabilidade	Funcionalidade	Alta
RNF-6	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.	Reusabilidade	Sistema	Média

4 Identificação de Subsistemas

A Figura 1 mostra os subsistemas identificados no contexto do presente projeto, os quais são descritos na Tabela 4. Vale ressaltar que os pacotes Core e People já existem no sistema Sigme e só foram modificados enquanto que o pacote Event foi criado.

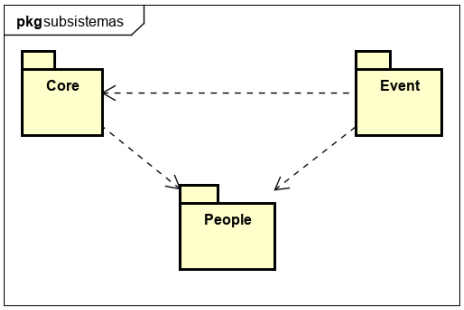

Figura 1 – Diagrama de Pacotes e os Subsistemas Identificados.

Tabela 4 – Subsistemas

Subsistema	Descrição
Core	Envolve todas as funcionalidades básicas do sistema, abrangendo as Instituições, Espíritas, Gestão, Tipo de Gestão, Papéis da Gestão, Região e as configurações globais que o sistema necessita para operar.
People	Envolve as funcionalidades bases para a definição das classe Espírita e Instituição. Abrange os endereços, cidades, tipos de telefone, telefone e estado.
Event	Envolve todas as funcionalidades referente a eventos do sistema, abrange os Eventos e as Inscrições feitas pelos usuários.

5 Modelo de Casos de Uso

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. Os atores identificados no contexto deste projeto estão descritos na Tabela 5.

Tabela 5 – Atores

Ator	Descrição
Administrador	É a pessoa responsável por administrar o sistema, tendo acesso a todas as funcionalidades do mesmo.
Espírita	Qualquer pessoa que realizar o cadastro no sistema.
Gestor	Espírita que faz parte da gestão atual de uma Instituição cadastrada no sistema.

A seguir, são apresentados os diagramas de casos de uso e descrições associadas, organizados por subsistema.

5.1 Subsistema Core

A Figura 2 apresenta o diagrama de casos de uso do subsistema Core que envolve o módulo de Gestão.

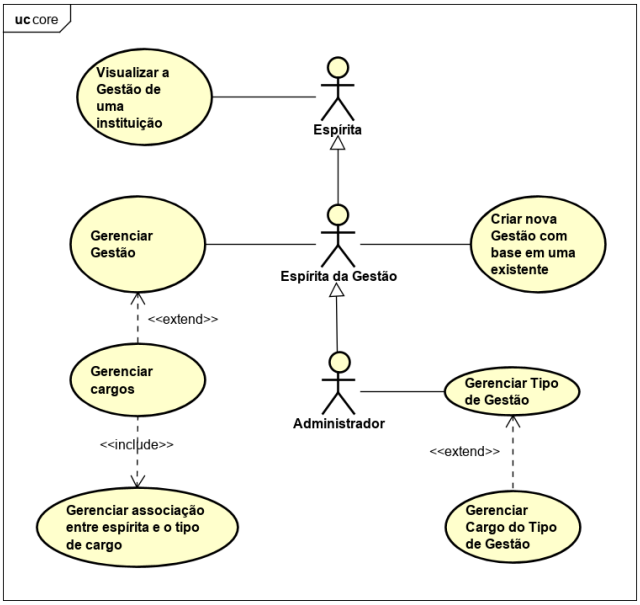

Figura 2 – Diagrama de Casos de Uso do Subsistema Core.

A seguir, são apresentadas as descrições de cada um dos casos de uso identificados. Os casos de uso cadastrais de baixa complexidade, envolvendo inclusão, alteração, consulta e exclusão, são descritos na Tabela 6.

Tabela 6 – Casos de Uso Cadastrais de Baixa Complexidade

Id	Nome	Ações	Observações	Requisitos	Classes
UC-1	Gerenciar Gestão	I	Informar: nome, descrição, instituição, tipo da gestão, data de início, data de fim.	RF-12, RN-2, RN-4	Gestão
		A		RF-13 RN-3	
		C			
		E			
UC-2	Gerenciar Tipo de Gestão	I	Informar: nome, descrição e o tipo de cargo/papéis existentes nesse tipo de gestão.	RF-8, RF-9	Tipo de Gestão
		A		RN-5	
		C			
		E	Só podem ser excluídos os tipos de gestões que não estão em uso.		
UC-3	Gerenciar Cargo do Tipo de Gestão	I	Informar: nome e descrição do cargo.	RF-10, RF-11 RN-5	Tipo de Cargo
		A			
		C			
		E	Só podem ser excluídos os cargos que não estão em uso.		

Descrição de Caso de Uso

Projeto: Sigme - Sistema de Informação Gerencial do Movimento Espírita
Identificador do Caso de Uso: UC-4
Caso de Uso: Visualizar a Gestão de uma instituição.
Descrição Sucinta: Este caso de uso é responsável por exibir as gestões das instituições para todos os usuários do sistema.

Tabela 7 – Fluxos de Eventos Normais

Nome do Fluxo	Precondição	Descrição
Verifica gestão atual		1. O usuário seleciona a instituição.
		2. O sistema exibe a gestão atual da instituição.
Verifica gestões passadas		1. O usuário seleciona a instituição.
		2. O sistema exibe uma lista de gestões passadas.
		3. O usuário seleciona uma das gestões passadas.
		4. O sistema exibe a gestão anterior para o usuário.

Requisitos Relacionados: RF-16
Classes Relacionadas: Instituição , Gestão

Descrição de Caso de Uso

Projeto: Sigme - Sistema de Informação Gerencial do Movimento Espírita
Identificador do Caso de Uso: UC-5
Caso de Uso: Criar uma nova gestão com base em alguma já existente.
Descrição Sucinta: Este caso de uso é responsável por permitir a criação de uma nova gestão com base na anterior.

Tabela 8 – Fluxos de Eventos Normais

Nome do Fluxo	Precondição	Descrição
Copiar gestão		1. O usuário informa que a nova gestão será baseada em uma anterior.
		2. O sistema lista as gestões anteriores para o usuário selecionar qual ele deseja.
		3. O usuário seleciona a gestão que ele quer copiar.
		4. O sistema exibe para o usuário a nova gestão, com os mesmos dados (estrutura de cargos, data de fim e data de início, e espíritas cadastros para os cargos) da gestão selecionada.

Requisitos Relacionados: RF-17
Classes Relacionadas: Gestão

Descrição de Caso de Uso

Projeto: Sigme - Sistema de Informação Gerencial do Movimento Espírita
Identificador do Caso de Uso: UC-6
Caso de Uso: Gerenciar cargos.
Descrição Sucinta: Este caso de uso é responsável por gerenciar os cargos de uma determinada gestão.

Tabela 9 – Fluxos de Eventos Normais

Nome do Fluxo	Precondição	Descrição
Gerenciar cargos da gestão	A gestão já está criada. O usuário que realizará a associação deve pertencer à gestão atual da instituição. O espírita que será atribuído ao cargo deve pertencer à instituição.	1. O usuário seleciona a gestão desejada. 2. O sistema exibe os cargos já registrados para aquela gestão. 3. O usuário cria um novo cargo ou seleciona um já existente. 4. É exibido os dados a serem preenchidos pelo usuário, ou os dados já incluídos no cargo. 5. O usuário seleciona os novos valores (tipo de cargo, nome do cargo e/ou espírita) para o cargo. 6. É exibida novamente a lista dos cargos associados à gestão com a modificação realizada.

Requisitos Relacionados: RF-14, RF-15, RN-4
Classes Relacionadas: Gestão, Cargo, Tipo de Cargo

5.2 Subsistema Event

A Figura 3 apresenta o diagrama de casos de uso do subsistema Event.

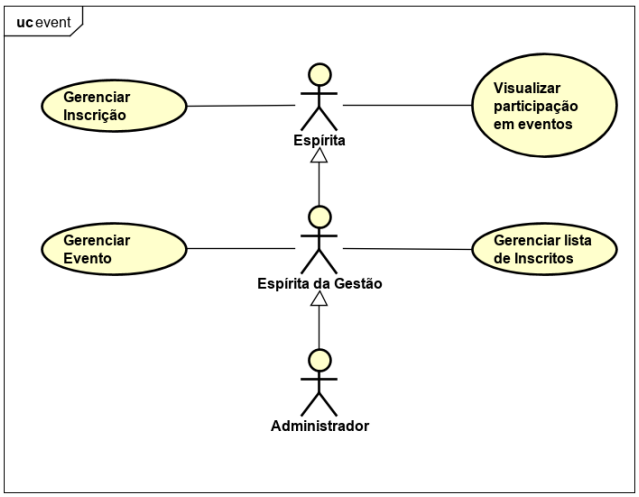

Figura 3 – Diagrama de Casos de Uso do Subsistema Event.

A seguir, são apresentadas as descrições de cada um dos casos de uso identificados. Os casos de uso cadastrais de baixa complexidade, envolvendo inclusão, alteração, consulta e exclusão, são descritos na Tabela 10.

Tabela 10 – Casos de Uso Cadastrais

Id	Nome	Ações	Observações	Requisitos	Classes
UC-7	Gerenciar Evento	I	Informar: nome, descrição, nome no crachá, preço, endereço, regional (opcional), data de início e fim do evento, data de início e fim da inscrição, instituição a que pertence e informações adicionais.	RF-1	Evento
		A		RF-2	
		C	Exibir os eventos que existem no sistema.		
		E	O evento pode ser excluído a qualquer momento. Caso exista inscrições para o evento as mesmas serão excluídas e informadas para os inscritos.	RN-6	

Descrição de Caso de Uso

Projeto: Sigme - Sistema de Informação Gerencial do Movimento Espírita
Identificador do Caso de Uso: UC-8
Caso de Uso: Gerenciar inscrição num evento.
Descrição Sucinta: Este caso de uso é responsável por permitir ao usuário gerenciar (inscrever, editar ou cancelar) sua inscrição num evento.

Tabela 11 – Fluxos de Eventos Normais

Nome do Fluxo	Precondição	Descrição
Realizar inscrição		1. O usuário seleciona o evento que quer se inscrever.
		2. São exibidos os dados principais do evento.
		2. O usuário informa o nome para o crachá e seleciona a instituição à que ele pertence e confirma os dados inseridos.
		3. O sistema realiza a inscrição do usuário no evento.
Editar Inscrição	O usuário está inscrito no evento	1. O usuário seleciona o evento.
		2. São exibidos o nome do crachá e a instituição inserida anteriormente.
		3. O usuário modifica o nome no crachá, ou a instituição à qual ele faz parte e confirma a modificação dos dados.
		4. É realizada a edição da inscrição no evento.
Cancelar Inscrição	O usuário está inscrito no evento.	1. O usuário seleciona o evento.
		2. São exibidos os dados da inscrição.
		3. O usuário confirma o cancelamento da inscrição.
		4. É cancelada a inscrição do usuário no evento.

Requisitos Relacionados: RF-3
Classes Relacionadas: Evento, Inscrição

Descrição de Caso de Uso

Projeto: Sigme - Sistema de Informação Gerencial do Movimento Espírita
Identificador do Caso de Uso: UC-9
Caso de Uso: Visualizar participação em eventos.
Descrição Sucinta: Este caso de uso é responsável por exibir ao usuário os eventos em que ele está inscrito e os que ele já se inscreveu.

Tabela 12 – Fluxos de Eventos Normais

Nome do Fluxo	Precondição	Descrição
Visualizar lista de eventos de um usuário		1. O usuário solicita a lista de eventos que se inscreveu.
		2. São exibidos todos os eventos que o usuário se inscreveu desde que se cadastrou no sistema.

Requisitos Relacionados: RF-7, RF-6
Classes Relacionadas: Evento, Inscrição

Descrição de Caso de Uso

Projeto: Sigme - Sistema de Informação Gerencial do Movimento Espírita
Identificador do Caso de Uso: UC-10
Caso de Uso: Gerenciar lista de inscritos.
Descrição Sucinta: Este caso de uso é responsável por exibir a lista de inscritos no evento.

Tabela 13 – Fluxos de Eventos Normais

Nome do Fluxo	Precondição	Descrição
Visualizar lista de inscritos		1. O usuário informa o evento.
	O usuário deve pertencer a gestão atual a qual o evento está vinculado.	2. O sistema exibe a lista de inscritos no evento.
Imprimir crachás		1. O usuário informa o evento.
	O usuário deve pertencer a gestão atual a qual o evento está vinculado.	2. O sistema gera os crachás dos inscritos, pronto para impressão.

Requisitos Relacionados: RF-4, RF-5, RN-6
Classes Relacionadas: Evento, Gestão, Inscrição

6 Modelo Estrutural

O modelo conceitual estrutural visa capturar e descrever as informações (classes, associações e atributos) que o sistema deve representar para prover as funcionalidades descritas na seção anterior. A seguir, são apresentados os diagramas de classes de cada um dos subsistemas identificados no contexto deste projeto. Na Seção 7 – Dicionário de Projeto – são apresentadas as descrições das classes, atributos e operações presentes nos diagramas desta seção.

6.1 Subsistema Core

A Figura 4 apresenta o diagrama de classe do sistema Core referente à adição do módulo de Gestão.

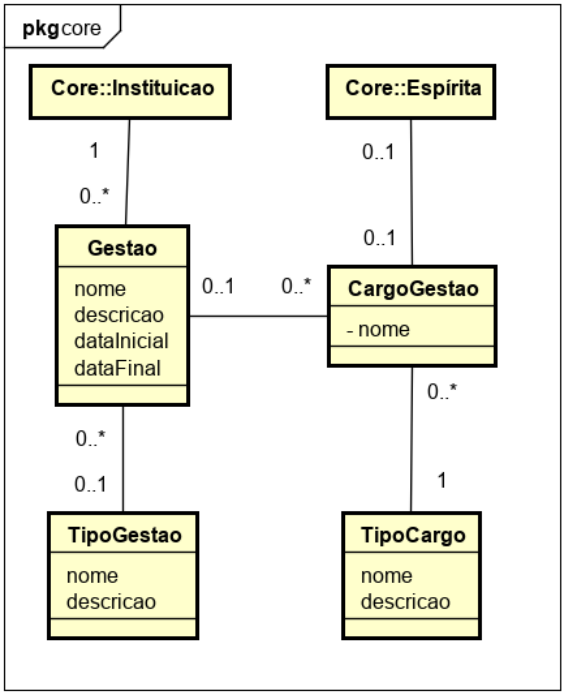

Figura 4 – Diagrama de Classes do Subsistema Core.

6.2 Subsistema Event

A Figura 5 apresenta o diagrama de classes do subsistema Event.

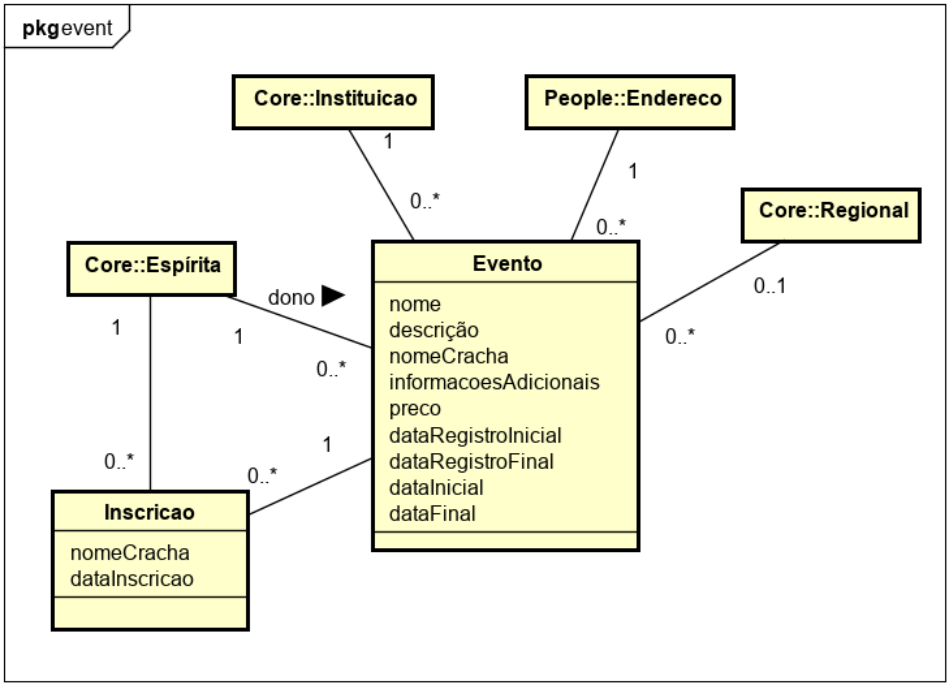

Figura 5 – Diagrama de Classes do Subsistema Event.

7 Dicionário de Projeto

Esta seção apresenta as definições das classes (e seus atributos), servindo como um glossário do projeto. Vale destacar que eventuais operações que estas classes vierem a ter não são listadas e descritas nesta fase do projeto.

7.1 Classes

7.1.1 Gestão (Management)

Propriedade	Tipo	Obrigatório?	Descrição
nome	Texto	Sim	Nome completo da Gestão.
descricao	Texto	Sim	Descrição sucinta Gestão.
dataInicial	Data	Sim	Data inicial da Gestão.
dataFinal	Data	Sim	Data final da Gestão.
instituição	Instituicao	Sim	Instituição a qual essa gestão está relacionada.
tipoGestao	TipoGestao	Sim	Tipo de Gestão.
cargos	Lista de Cargo	Não	Cargos que a Gestão possui.

7.1.2 Tipo de Gestão (ManagementType)

Propriedade	Tipo	Obrigatório?	Descrição
nome	Texto	Sim	Nome completo da Gestão.
descricao	Texto	Sim	Descrição sucinta Gestão.
tipoCargo	Lista de TipoCargo	Sim	Tipo de Cargos que a gestão possui.
gestao	Gestao	Sim	Gestão.

7.1.3 Cargo (ManagementRole)

Propriedade	Tipo	Obrigatório?	Descrição
nome	Nome do Cargo	Sim	Nome do cargo.
espírita	Espírita	Não	Espírita a qual determinado cargo foi designado.
tipoCargo	TipoCargo	Sim	Tipo do cargo.
gestao	Gestao	Sim	Gestão do cargo.

7.1.4 Tipo de Cargo (ManagementRoleType)

Propriedade	Tipo	Obrigatório?	Descrição
nome	Texto	Sim	Nome completo da Gestão.
descricao	Texto	Sim	Descrição sucinta Gestão.

7.1.5 Evento (Event)

Propriedade	Tipo	Obrigatório?	Descrição
nome	Texto	Sim	Nome completo do evento.
descricao	Texto	Sim	Descrição do evento.
nomeCracha	Texto	Não	Nome do evento que aparecerá no crachá.
informacoesAdicionais	Texto	Não	Informações adicionais do evento
preco	Texto	Não	Preço para participar no evento.
dataRegistroInicial	Data	Não	Data inicial para se registrar no evento.
dataRegistroFinal	Data	Não	Data final para se registrar no evento.
dataInicial	Data	Sim	Data de início do evento
regional	Regional	Não	Região à qual o evento pertence.
endereço	Endereço	Sim	Endereço do evento.
espiritaDono	Espírita dono do evento	Sim	Espírita responsável e que criou o evento.
instituição	Instituição	Sim	Instituição responsável pelo evento.
inscricoes	Lista de Inscricao.	Não	Lista de inscritos do evento.

7.1.6 Inscrito (Subscriber)

Propriedade	Tipo	Obrigatório?	Descrição
espirita	Espirita	Sim	Espírita associado a tal inscrição.
evento	Evento	Sim	Evento associado a tal inscrição.
nomeCraca	Texto	Sim	Nome do espírita que aparecerá no crachá.
dataInscricao	Data	Sim	Data em que a inscrição foi realizada.

7.1.7 Instituição (Institution)

Essa classe já existia no sistema, mas foi necessário realizar algumas alterações por conta das classes do módulo de Gestão e suas relações com esta classe. Abaixo iremos explicitar somente as propriedades que foram adicionadas para o projeto.

Propriedade	Tipo	Obrigatório?	Descrição
gestao	Gestão	Não	Gestão atual da instituição.
historicoGesao	Lista de Gestão	Não	Histórico de gestões da instituição.
cargosGestao	Lista de Cargos da Gestão	Não	Lista dos Cargos atuais da Gestão.

Documento de Projeto de Sistema

Sigme - Sistema de Informação Gerencial do Movimento Espírita

Registro de Alterações:

Versão	Responsável	Data	Alterações
1.0	Rodrigo Bittencourt Pimenta	25/10/2016	Versão inicial
1.1	Rodrigo Bittencourt Pimenta	02/04/2019	Revisões
2.0	Rodrigo Bittencourt Pimenta	06/06/2019	*quinta revisão - atualização de diagramas e correções de erros*

Vitória, ES

2016

1 Introdução

Este documento apresenta o documento de projeto (*design*) do sistema Sigme - Sistema de Informação Gerencial do Movimento Espírita. Este documento está organizado da seguinte forma:

- Seção 2 apresenta a plataforma de software utilizada na implementação da ferramenta.
- Seção 3 trata de táticas utilizadas para tratar requisitos não funcionais (atributos de qualidade).
- Seção 4 apresenta o projeto da arquitetura de software e suas subseções explicam cada uma de suas camadas.

2 Plataforma de Desenvolvimento

Na Tabela 1 são listadas as tecnologias utilizadas no desenvolvimento da ferramenta, bem como o propósito de sua utilização.

Tecnologia	Versão	Descrição	Propósito
Java	8	Linguagem de programação orientada a objetos e independente de plataforma.	Desenvolvimento de aplicativos em linguagem de programação orientada a objetos e independente de plataforma.
Java EE	7	Uma série de especificações bem detalhadas, dando uma receita de como deve ser implementado um software que faz cada um desses serviços de infraestrutura (CAELUM, 2016b).	Reduzir a complexidade do desenvolvimento, implantação e gerenciamento de aplicações, de modo que o desenvolvedor não se preocupe demasiadamente com segurança, escalabilidade e desempenho.
WildFly	10.1.0	Servidor de Aplicações para Java EE.	Prover os recursos necessários para execução de aplicações Web dentro da plataforma Java EE.
JSF	2.0	Framework web que estabelece o padrão para construção de interfaces de sistemas para web.	Melhorar a produtividade, permitindo a construção de interfaces para web usando um conjunto de componentes pré-construídos, ao invés de criar interfaces inteiramente do zero.
PrimeFaces	5.1	Biblioteca de componentes de interface gráfica para as aplicações web baseadas em JSF (CAELUM, 2016a).	Reutilizar componentes avançados de interface gráfica.
NemoUtils	1.1	Conjunto de classes a frameworks para auxiliar o desenvolvimento de projetos, especialmente os baseados em Java para web.	Diminuir a preocupação em criar classes para o cadastramento de classes básicas.
JPA	2.0	API para persistência de dados por meio de mapeamento objeto-relacional.	Eliminar trabalho para montar scripts em SQL e facilitar a manutenção dos mesmos.
CDI	1.2	API que cuida da parte de injeção de dependências	Integração das diferentes camadas da arquitetura e serviços de transação.
MySQL Server	5.6.16	Sistema Gerenciador de Banco de Dados (SGBD) Relacional gratuito.	Persistência dos dados manipulados pela ferramenta.
FrameWeb (SOUZA, 2007)	-	Método baseado em Frameworks para o Projeto de Sistemas de Informação Web	Propõe uma arquitetura básica para desenvolvimento de WIS (*Web-bases Information Systems*) de forma a aumentar a produtividade.

Tabela 1 – Plataforma de Desenvolvimento e Tecnologias Utilizadas

Na Tabela 2 vemos os softwares que apoiaram o desenvolvimento de documentos e também do código fonte.

Tecnologia	Versão	Descrição	Propósito
Eclipse Java EE IDE for Web Developers	4.6.1	Ambiente de desenvolvimento (IDE) para a linguagem Java.	Facilitar a atividade de implementação de software.
MySQL Workbench	6.3.7	Aplicativo que permite gerenciar os esquemas e os dados em um SGBD MySQL.	Administração do SGBD.
Astah Professional	7.1.0	Software para modelagem UML (Unified Modeling Language)	Facilidade o desenvolvimento dos diagramas necessários para a Engenharia de Software, tais como Diagrama de Classes, Diagrama de Casos de Uso, entre outros.
Apache Maven	3.2.5	Ferramenta de gerência de projeto baseada em project object model (POM).	Simplificar o download das dependências do projeto.
TexStudio	5.6.1	Editor de LaTeX.	Escrever a documentação do sistema.

Tabela 2 – Softwares de Apoio ao Desenvolvimento do Projeto

3 Atributos de Qualidade e Táticas

Na Tabela 3 são listados os atributos de qualidade considerados neste projeto, com uma indicação se os mesmos são condutores da arquitetura ou não e as táticas a serem utilizadas para tratá-los.

Categoria	Requisitos Não Funcionais	Tática
Portabilidade	RNF-1	1- Organizar a arquitetura da ferramenta segundo uma combinação de camadas e partições. 2- A camada de lógica de negócio deve ser organizada segundo o padrão Camada de Serviço. 3- A camada de gerência de dados deve ser organizada segundo o padrão DAO. 4- Utilizar uma linguagem que seja compatível com os principais navegadores do mercado.
Facilidade de Aprendizado, Facilidade de Operação	RNF-2, RNF-3	Prover ao usuário a capacidade de entrar com comandos que permitam operar o sistema de modo mais amigáveis. Para tal, as interfaces do sistema devem permitir, sempre que possível, a entrada por meio de seleção ao invés da digitação de campos.
Segurança de Acesso	RNF-4	Identificar usuários usando login e autenticá-los por meio de senha.
Reusabilidade	RNF-5	Reutilizar componentes e frameworks existentes, tais como nemo-utils e os demais frameworks implementados pelo JAVA EE. Quando não houver componentes disponíveis e houver potencial para reúso, devem-se desenvolver novos componentes para reúso.

Tabela 3 – Atributos de Qualidade e Táticas Utilizadas

4 Arquitetura de Software

A arquitetura de software do sistema Sigme, baseia-se na combinação de módulos e camadas. Cada um desses módulos, por sua vez, está organizado em três camadas seguindo o proposto pelo FrameWeb, a saber: Camada de Apresentação (*Presentation Tier*), Camada de Negócios (*Business Tier*) e Camada de Acesso a Dados (*Data Access Tier*). De forma a dar suporte para a construção da aplicação, a ferramenta de apoio nemo-utils será utilizada. Tal ferramenta provê classes que auxiliam na implementação dos casos de uso cadastrais que seguem o modelo de arquitetura a ser utilizado.

A primeira camada contém os pacotes de Visão (*View*) e Controle (*Control*), a segunda contém o de Domínio (*Domain*) e o de Aplicação (*Application*) e a terceira somente o pacote de Persistência (*Persistence*). Cada pacote será explicado melhor nas próximas seções onde serão descritos os três módulos do Sigme (*Core*, *People* e *Event*). A Figura 1 apresenta a visão geral das camadas e seus pacotes juntamente com o relacionamento que existe entre eles e as tecnologias Java EE utilizadas em cada pacote.

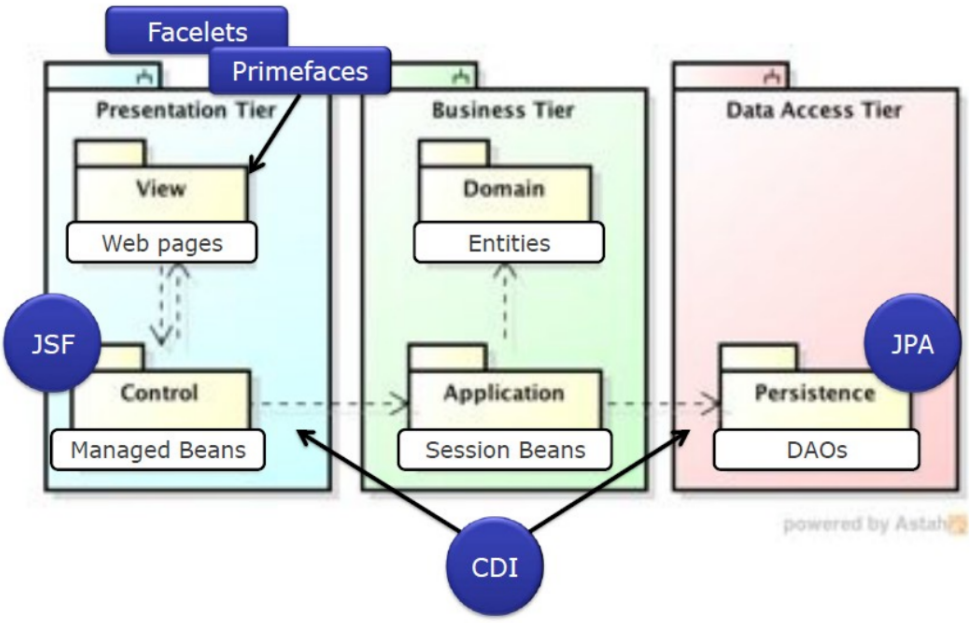

Figura 1 – Arquitetura de Software do Sistema (NASCIMENTO, 2015)

A Figura 2 apresenta a subdivisão de cada módulo nas camadas descritas acima, a saber: a Camada de Apresentação (`controller`), Camada de Negócios (`domain`, `application` e `exceptions`) e Camada de Acesso a Dados (`persistence`).

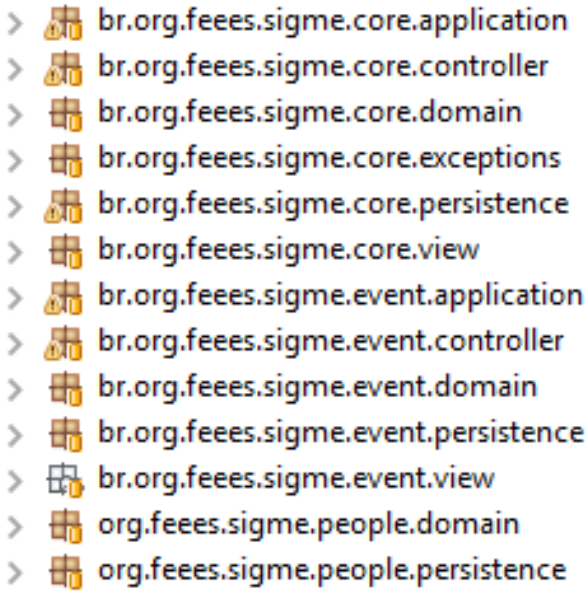

Figura 2 – Subdivisão dos módulos sigme.core, sigme.event. sigme.people

4.1 Camada de Apresentação

As funcionalidades criar, visualizar, editar e excluir (abreviadas de CRUD, do inglês *create, read, update and 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 na Figura 3 através de um modelo de apresentação genérico.

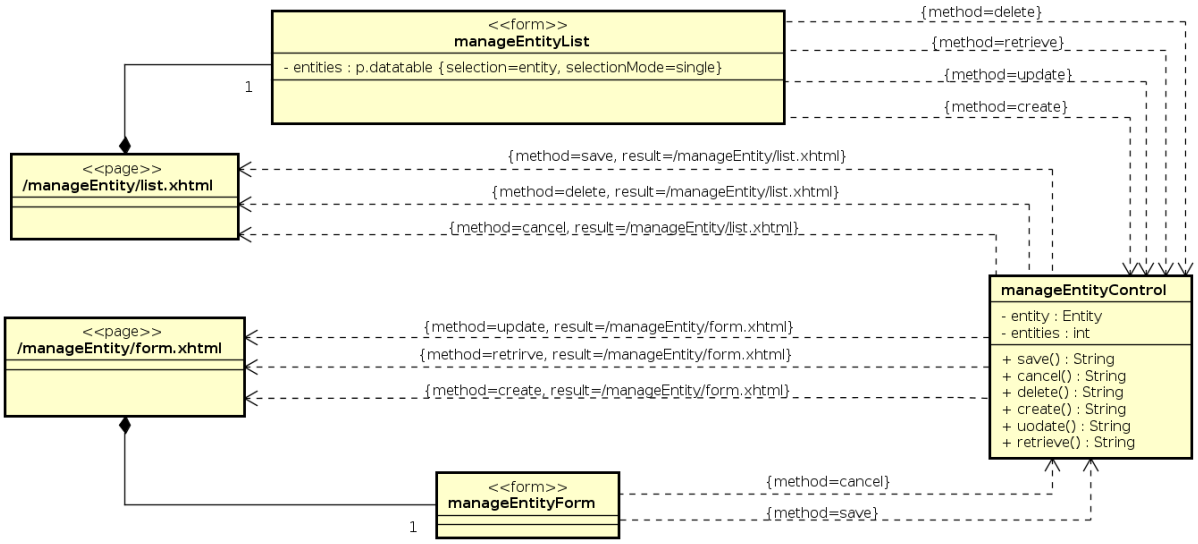

Figura 3 – Modelo de Navegação de um CRUD nemo-utils, usado como base para funcionalidades dos cadastros do sistema SAE (NASCIMENTO, 2015).

Algumas funcionalidades implementadas não são satisfeitas pelo modelo de navegação da Figura 3. A Figura 4 exibe o modelo de navegação do módulo de gerenciamento de eventos, que possui as funcionalidades adicionais de (i) se inscrever num evento a partir de lista de eventos, (ii) gerenciar um evento e (iii) imprimir a lista de inscritos num evento.

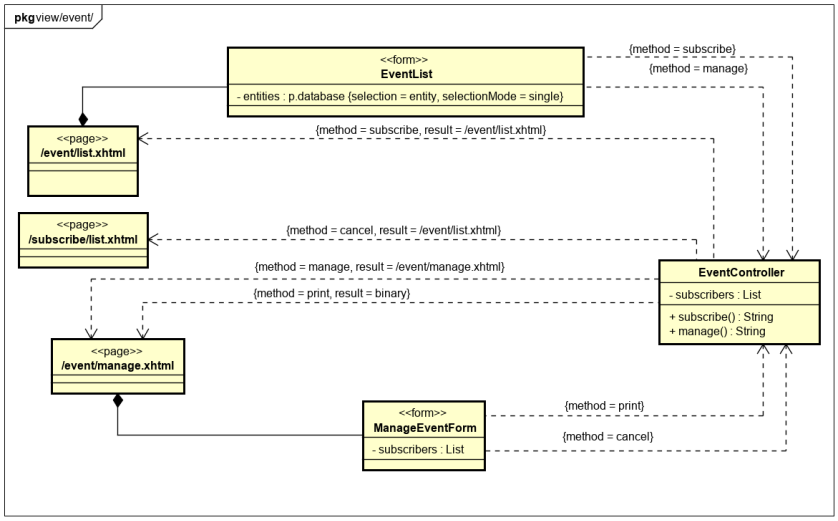

Figura 4 – Modelo de Navegação das funcionalidades do Módulo de Gerenciamento de Eventos

4.2 Camada de Negócios

4.2.1 Dominio

Todos os atributos que são não nulos tem a *tag* `not null` e os que são nulos tiveram a tag `null` omitida, conforme a proposta original do FrameWeb.

Todas as classes de domínio estendem `PersistentObjectSupport` do pacote nemo-utils, sendo que essa herança não é mostrada nos diagramas com o intuito de não poluí-los com várias associações.

As figuras 5 e 6 mostram os Modelos de Entidade para os módulos *sigme.core* e *sigme.event*, respectivamente.

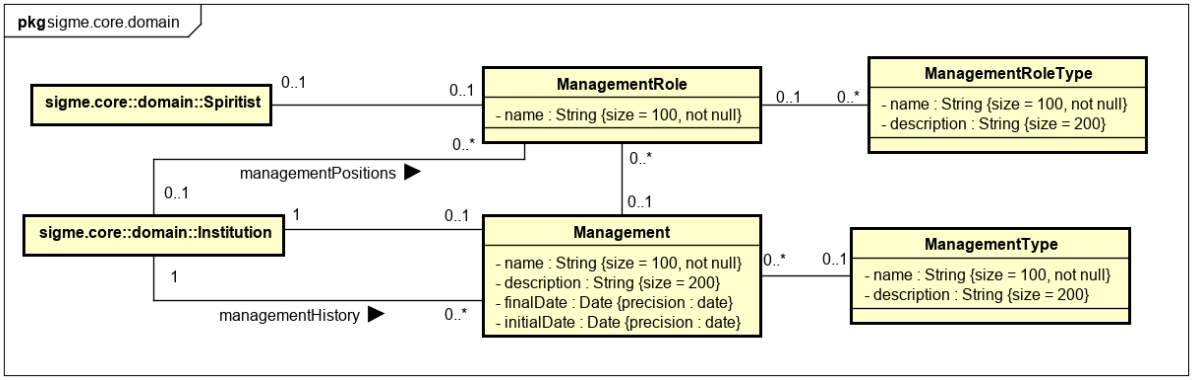

Figura 5 – Modelo de Entidade do módulo *sigme.core*.

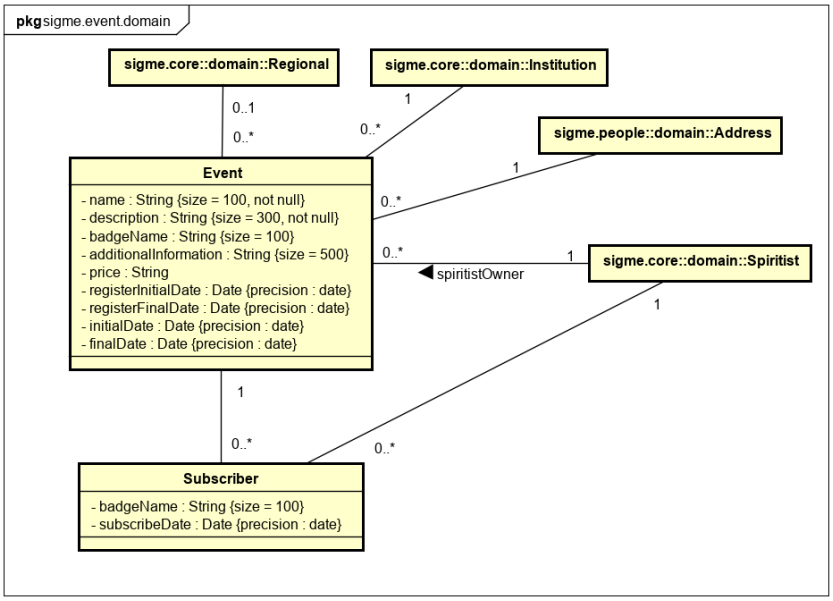

Figura 6 – Modelo de Entidade do módulo *sigme.event*.

4.2.2 Aplicação

Todas as classes de aplicação que são de casos de uso cadastrais estendem de `CrudServiceBean` do pacote nemo-utils. Da mesma forma dos diagramas anteriores essa herança não é mostrada no diagrama com o intuito de não poluir o diagrama com várias associações.

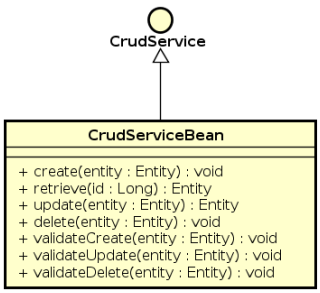

Figura 7 – Modelo de Aplicação genérica da ferramenta nemo-utils (NASCIMENTO, 2015).

As Figuras 8 e 9 mostram os Modelos de Aplicação para os módulos *sigme.core* e *sigme.event*, respectivamente. O módulo *sigme.people*, por se conter somente ferramentas de auxílio para a melhor exibição dos dados, não possui camada de aplicação, portanto não foi exibido.

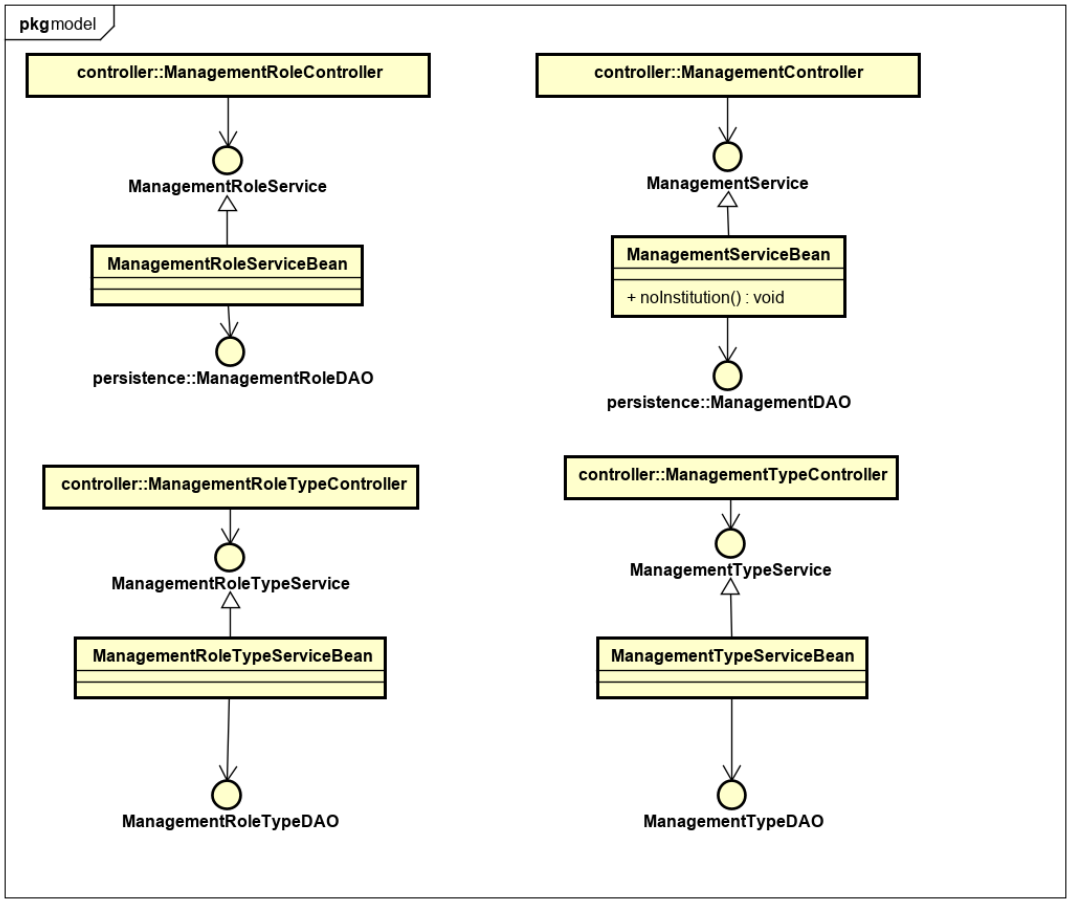

Figura 8 – Modelo de Aplicação do módulo *sigme.core*.

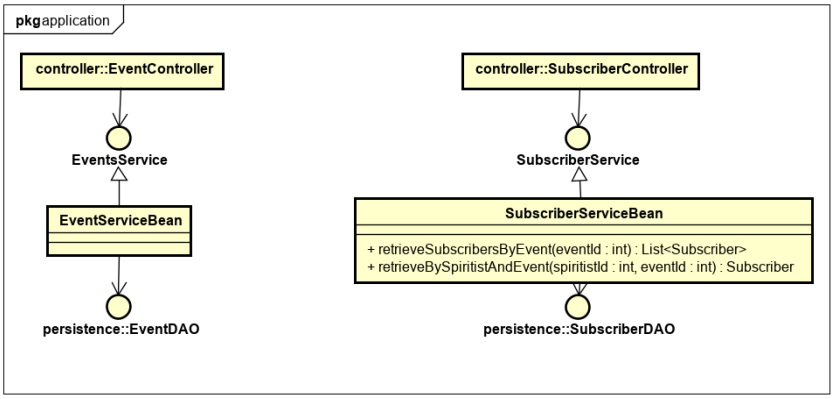

Figura 9 – Modelo de aplicação do módulo *sigme.event*.

4.3 Camada de Acesso a Dados

Como exibido nas camadas anteriores, apresentamos abaixo os modelos de persistência desenvolvidos para o sistema Sigme. O nome das classes é uma sugestão do FrameWeb para simplificar o processo de software. Como estamos usando a ferramenta nemo-utils, mostramos na Figura 10 o Modelo de Persistência genérico definido por ela.

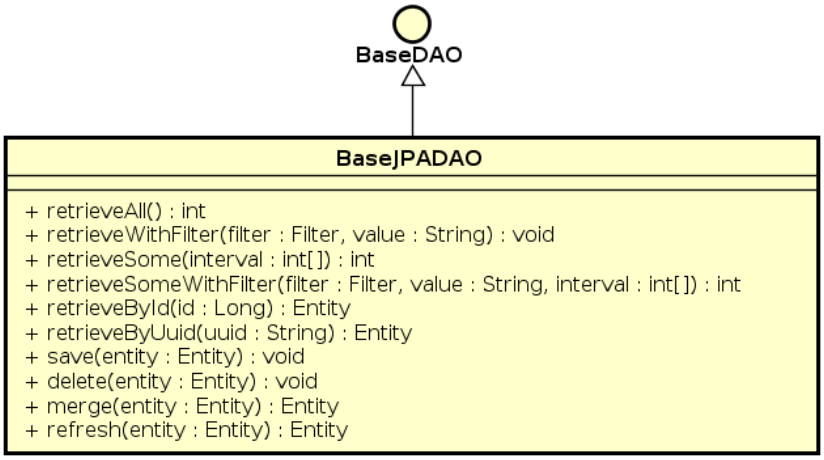

Figura 10 – Modelo de Persistência genérica da ferramenta nemo-utils (NASCIMENTO, 2015).

As Figuras 11 e 12 mostram os Modelos de Persistência para os módulos *sigme.core* e *sigme.event*, respectivamente. O módulo *sigme.people*, por se conter somente ferramentas de auxílio para a melhor exibição dos dados, não possui camada de persistência, portanto não foi exibido.

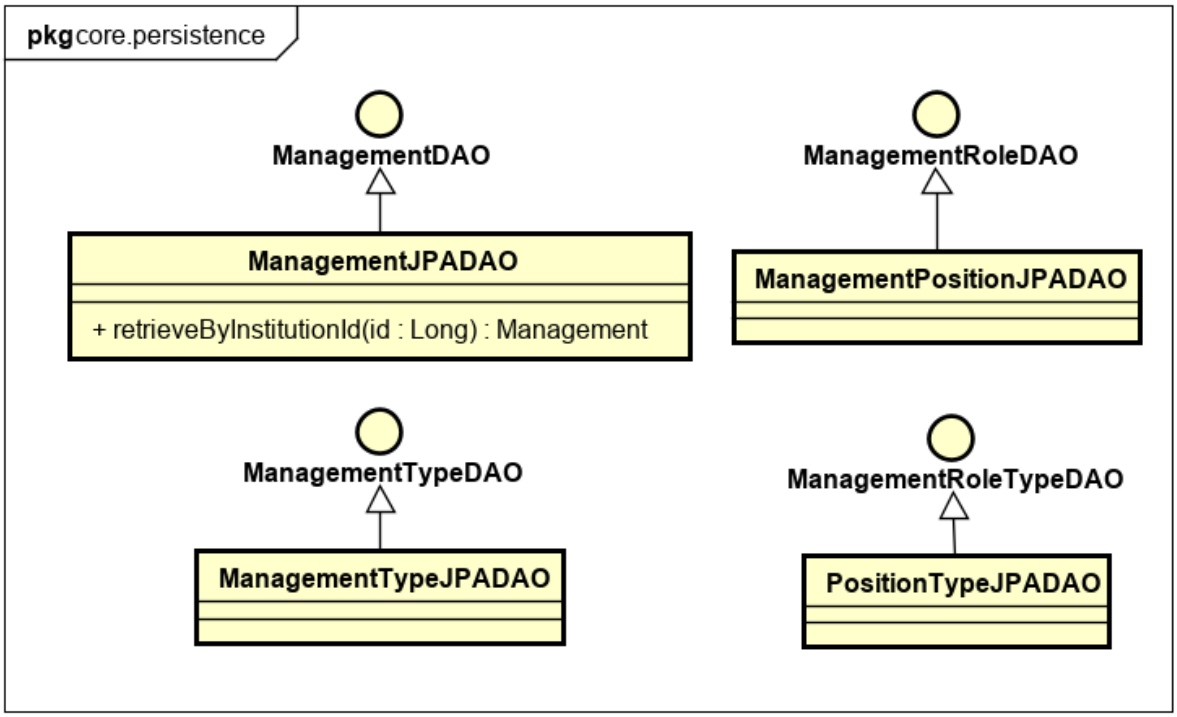

Figura 11 – Modelo de aplicação do módulo *sigme.core*.

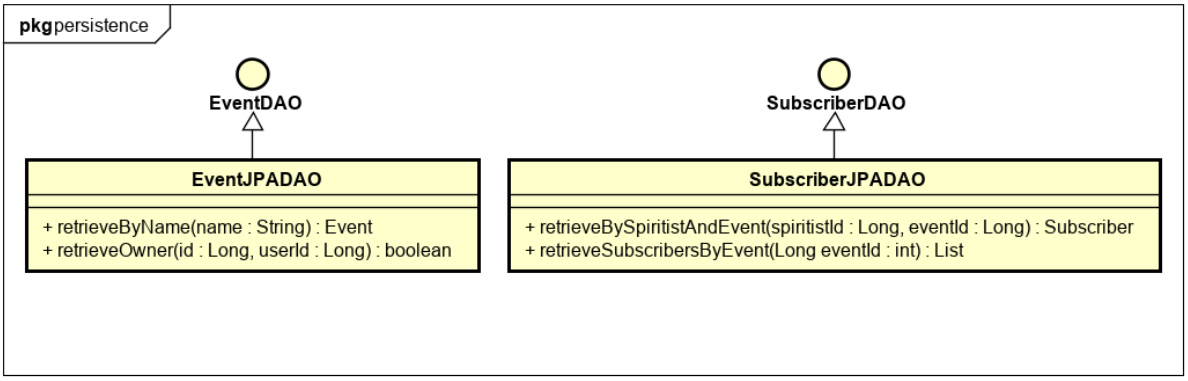

Figura 12 – Modelo de aplicação do módulo *sigme.event*.

Referências

CAELUM. *Introdução ao PrimeFaces*. 2016. Disponível em: <http://www.devmedia.com.br/introducao-ao-primefaces/33139>. Citado na página 2.

CAELUM. *O que é Java EE?* 2016. Disponível em: <https://www.caelum.com.br/apostila-java-web/o-que-e-java-ee/>. Citado na página 2.

NASCIMENTO, B. M. *SAE - Sistema de Acompanhamento de Egressos*. [S.l.], 2015. Citado 4 vezes nas páginas 5, 7, 10 e 12.

SOUZA, V. E. S. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. [S.l.], 2007. Citado na página 2.