Thiago Martinho da Costa

Resgate - Despacho de ambulância automatizado

Vitória, ES

2015

Thiago Martinho da Costa

Resgate - Despacho de ambulância automatizado

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

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Departamento de Informática

Orientador: Prof. Dr. Vítor E. Silva Souza

Vitória, ES

2015

Thiago Martinho da Costa
Resgate - Despacho de ambulância automatizado/ Thiago Martinho da Costa. – Vitória, ES, 2015-
105 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, 2015.

1. FrameWeb. 2. Java™ EE 7. 3. JSF. 4. CDI. 5. JPA. 6. LAS-CAD I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. Resgate - Despacho de ambulância automatizado

CDU 02:141:005.7

Thiago Martinho da Costa

Resgate - Despacho de ambulância automatizado

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

Trabalho aprovado. Vitória, ES, 14 de dezembro de 2015:

Prof. Dr. Vítor E. Silva Souza
Orientador

Prof. André Luiz de Castro Leal, D. Sc.
Universidade Federal Rural do Rio de Janeiro

Beatriz Franco Martins, B.Sc.
Programa de Pós-Graduação em Informática – Ufes

Vitória, ES
2015

Agradecimentos

Em primeiro lugar agradeço a Deus por mais esta conquista. Por me dar ânimo, paciência, dedicação e força de vontade para enfrentar todas as dificuldades e situações adversas com as quais tive que lidar durante a caminhada.

Aos meus pais, Glici e Valter, por terem me apoiado, dado muita força e carinho e me ajudado a chegar onde cheguei e principalmente por terem compreendido a minha recorrente falta de disponibilidade e as reclamações semanais.

A todos os meus amigos, em especial ao meu irmão Teemu e Guilherme, que aguentaram meus momentos de desespero e dificuldade. Aos amigos da UFES, Jordana, Miguel, Mateus, Higor, Thiago e Alexandre, por terem me auxiliado, entendido e ajudado durante todos esses anos. Ao pessoal da Infobase Consultoria e Informática, onde aprendi a ser um profissional e tive a oportunidade de colocar em prática o que aprendi durante o curso de graduação.

Por último, mas não menos importante, ao Prof. Dr. Vítor Souza pela excelente orientação, por sempre ter "5 minutinhos" disponíveis para mim, ao Prof. Dr. Ricardo Falbo, por facilitar a minha escolha de área de atuação e à Prof. Ma. Rosane Caruso por seus auxílios, dicas e conversas, que tornaram a minha vida dentro da Universidade menos difícil.

Resumo

Na década de 1990, um sistema de despacho de ambulâncias foi encomendado para a capital da Inglaterra, Londres, devido a necessidade de uma nova forma de gerenciar o controle de ambulâncias para a cidade. No primeiro momento a ideia foi implantada de forma errônea e a falha do sistema *London Ambulance Computer-Aided Dispatch* (LAS-CAD), relatada inicialmente em (FINKELSTEIN, 1993), se tornou um caso muito famoso de estudo dentro da área de Engenharia de Software, conforme declarado por (SOUZA; MYLOPOULOS, 2013).

Em (SOUZA, 2007), foi proposto o método *FrameWeb - Framework-based Design Mehtod for Web Engineering* que sugere a utilização de vários *frameworks* afim de obter uma construção mais rápida e eficiente do sistema além de propor uma arquitetura de *software* dada através de modelos contendo informações relevantes para desenvolvimento de ferramentas voltada para a *web*.

Este projeto, portanto, visa modelar, projetar e desenvolver a base desse sistema de ambulâncias para que, no futuro, possa ser evoluído para uma versão adaptativa de modo a ser utilizado nas pesquisas dessa área, além de prover uma implementação para um sistema que é bastante utilizado como exemplo em pesquisas, facilitando, assim, futuros testes nesta área. Outro objetivo consiste em experimentar o *framework FrameWeb*, provendo *feedback* para esta pesquisa em andamento.

Baseando-se no *FrameWeb* e partindo dos requisitos elicitados em (SOUZA; MYLOPOULOS, 2013), foi desenvolvido um sistema, na plataforma Java™ EE 7, de despacho de ambulâncias automatizado chamado **Resgate** que utiliza diversos *frameworks* e padrões de desenvolvimento voltados para a *web*, como por exemplo, *JavaServer Faces (JSF)*, *Contexts Depedency Injection (CDI)* e *Java Persistence API*.

Palavras-chaves: *FrameWeb*. Java™ EE 7. JSF. CDI. JPA. LAS-CAD.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UML	Unified Modeling Language
LAS-CAD	London Ambulance Computer-Aided Dispatch
A-CAD	Adaptative Computer-Aided Ambulance Dispatch
DAO	Data Access Object
HTML	HyperText Markup Language
MDT	Mobile Data Terminal
JSF	JavaServer Faces
JPA	Java™ Persistence API
CDI	Contexts and Dependency Injection
WebApp	Web Application
Java™ EE 7	Java™ Enterprise Edition 7
CRUD	Create, Read, Update, Delete
API	Application Programming Interface
SQL	Structured Query Language
JDBC	Java Database Connectivity

Sumário

1 Introdução

Na década de 1990, um sistema de despacho de ambulâncias foi encomendado para a capital da Inglaterra, Londres, devido a necessidade de uma nova forma de gerenciar o controle de ambulâncias para a cidade. No primeiro momento a ideia foi implantada de forma errônea e a falha do sistema *London Ambulance Computer-Aided Dispatch* (LAS-CAD), relatada inicialmente em (FINKELSTEIN, 1993), se tornou um caso muito famoso de estudo dentro da área de Engenharia de Software (SOUZA; MYLOPOULOS, 2013).

Devida a alta criticidade de vários componentes do LAS-CAD, este se tornou um bom caso de estudo para sistemas adaptativos (SOUZA; MYLOPOULOS, 2013). Em (SOUZA; MYLOPOULOS, 2012), e mais recentemente na forma de artigo em (SOUZA; MYLOPOULOS, 2013), foram realizados experimentos com simulações utilizando o conceito de sistemas adaptativos aplicando ao *Adaptive Computer-Aided Ambulance Dispatch* (A-CAD), ou seja, o sistema adaptativo de controle de despacho de ambulância. No entanto, com o intuito de obter uma avaliação mais precisa da solução, considera-se interessante realizar os experimentos não com simulações, mas com um sistema real. Este trabalho, portanto, visa modelar, projetar e desenvolver a base desse sistema de ambulâncias para que, no futuro, possa ser evoluído para uma versão adaptativa de modo a ser utilizado nas pesquisas dessa área.

O sistema desenvolvido começa a atuar no momento em que um cidadão efetua uma chamada para o serviço de emergência local. Com os dados identificados através da chamada, cabe ao atendente registrar a chamada recebida, associá-la a um incidente já registrado no sistema ou criar um novo e liberar o despacho de ambulância para o atendimento. O sistema também permite operações de controle de cadastros, tais como, cadastro de funcionários, de ambulâncias, de hospitais e de equipamentos.

Com o poder de uma das tecnologias mais importantes da história da humanidade, a Internet, tem-se várias ferramentas e *frameworks* de alto nível que auxiliam no desenvolvimento do sistema proposto de forma a ser compatível com a *web*. Inicialmente o desenvolvimento de *softwares* com acesso disponível via *web* (*WebApp – Web Application*) era uma coisa limitada. Devido à restrição de tecnologia no passado, sistemas de alta complexidade simplesmente eram bem complicado de serem construídos.

Com a evolução das tecnologias e da facilidade de acesso, sistemas cada vez mais complexos puderam ser construídos chegando ao ponto de existirem plataformas inteiras construídas e disponíveis apenas via *web*. Claro que quanto mais complexa uma ferramenta é, mais difícil é documentar e adotar uma abordagem de desenvolvimento que auxilie de fato na construção desses sistemas.

Pensando nisso, (SOUZA, 2007) propôs o método *FrameWeb* (*Framework-based Design Method for Web Engineering*) que sugere o uso de diferentes ferramentas e *frameworks* com o intuito de obter rapidez e eficiência na construção de sistemas de informação para a *web*. A proposta do *FrameWeb* trabalha com recomendações desde a fase de desenvolvimento, ou seja, levantamento de requisitos, análise de requisitos, projeto de sistema até uma extensão da linguagem UML cujo produto final são diferentes modelos que são produzidos durante a fase de projeto.

A proposta do presente trabalho é o desenvolvimento de um Sistema de Despacho Automatizado de Ambulância (Resgate), um *WebApp* para auxiliar os serviços de emergência no despacho e gerenciamento de ambulâncias. O projeto seguiu técnicas de Engenharia de Software, Engenharia de Requisitos e Engenharia Web, aplicando o método *FrameWeb* para o projeto arquitetural e desenvolvendo a aplicação na plataforma Java™ EE 7 (DEMICHIEL; SHANNON, 2013).

1.1 Objetivos

O objetivo geral deste trabalho é o desenvolvimento de um sistema *web* de controle de despacho de ambulância automatizado utilizando os conhecimentos adquiridos ao longo do curso.

Há também alguns objetivos secundários que estão ligados ao desenvolvimento do sistema Resgate:

- Verificar as necessidades do sistema que será criado e escrever documentos com as especificações dos requisitos;
- Definir todo o projeto do sistema com suas estruturas e arquiteturas;
- Contribuir com a pesquisa realizada em (SOUZA; MYLOPOULOS, 2013), provendo um protótipo de sistema real com o qual as propostas desta pesquisa podem ser experimentadas na prática.

1.2 Atividades Realizadas

Para a condução do projeto de graduação descrito neste documento, foi seguida uma metodologia composta das seguintes atividades:

- Pesquisa: estudo a respeito do *FrameWeb* e sobre os diversos *frameworks* nos quais o método foi inicialmente testado para o desenvolvimento de uma aplicação Java *Web* utilizando a plataforma Java™ EE 7 e do *framework* Nemo-Utils;

- Levantamento de requisitos: fase de levantamento de todos os requisitos das funcionalidades do sistema;
- Especificação de requisitos: fase de análise de todos os requisitos levantados anteriormente, onde os mesmos foram definidos com mais detalhes e criados diagramas para melhor compreensão e modelagem dos casos de uso;
- Projeto e implementação: fase de definição das tecnologias e arquiteturas que serão utilizadas para implementar o sistema. Em seguida, o sistema foi implementado conforme projetado;
- Apresentação do Projeto: apresentação final do projeto. O projeto e a ferramenta foram entregues e demonstrados.

1.3 Organização da Monografia

Essa monografia é dividida em 5 capítulos incluindo a introdução.

No capitulo 2 realiza-se um levantamento dos principais temas abordados ao longo deste trabalho que são: Engenharia de Requisitos, Projeto de Sistemas e implementação e *FrameWeb*.

No capitulo 3 é feita a especificação e análise dos requisitos do Resgate (Sistema automatizado de despacho de ambulância) sistema *web* que foi desenvolvido utilizando o *FrameWeb*.

No capitulo 4 apresenta-se a fase de projeto de sistemas utilizando o método *FrameWeb* e a implementação do sistema.

No capitulo 5 tem-se as conclusões retiradas deste trabalho.

2 Referencial Teórico

Este capítulo apresenta as principais bases teóricas para o desenvolvimento do Resgate – o Sistema de Despacho de Ambulância Automatizado, sendo elas: especificação de requisitos (Seção 2.1), projeto de sistemas e implementação (Seção 2.2) e o método *FrameWeb* (Seção 2.3).

2.1 Especificação de Requisitos

A especificação de requisitos compreende a primeira etapa do processo de desenvolvimento de um *software*. Nesta etapa são definidas as funcionalidades que o sistema deve conter e os critérios de qualidade sob os quais o sistema deverá operar. É caracterizada por um esforço conjunto entre clientes, usuários e especialistas de domínio a fim de traçar uma linha conjunta de entendimento entre todas as partes envolvidas a respeito de como um *software* se comportará e de como será utilizado pelo usuário. Trata-se de uma atividade complexa que não se resume somente a perguntar às pessoas o que elas desejam, mas sim analisar cuidadosamente a organização, o domínio da aplicação e os processos de negócio no qual o sistema será utilizado (SOMMERVILLE; KOTONYA, 1998).

Durante essa fase, as partes envolvidas fazem uma análise profunda nas funções que o *software* deverá conter, especificando assim, os requisitos do sistema. Para requisitos existem diversas definições na literatura, valendo destacar:

- Requisitos de um sistema são descrições dos serviços que devem ser fornecidos por esse sistema e as suas restrições operacionais (SOMMERVILLE et al., 2003);
- Um requisito de um sistema é a característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir seus objetivos (PFLEEGER, 2004);
- Um requisito é alguma coisa que o produto tem de fazer ou uma qualidade que ele precisa apresentar (ROBERTSON; ROBERTSON, 2006).

Com base nessas e em outras definições, pode-se dizer 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 (FALBO, 2012). Os requisitos de *software* de um sistema podem ser divididos em funcionais e não funcionais.

Requisitos funcionais são declarações de serviços que o sistema deve prover, descrevendo o que o sistema deve fazer (SOMMERVILLE et al., 2003). Já os não funcionais,

descrevem restrições sobre os serviços ou funções oferecidas pelo sistema (SOMMERVILLE et al., 2003), as quais limitam as opções para criar uma solução para o problema (PFLEEGER, 2004).

Um dos modelos criados durante a fase de requisitos, 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, isto é, os usuários do sistema (SOUZA, 2007). Com isso, o usuário tem a descrição passo a passo de todas as funcionalidades do sistema e de como pode prosseguir para executar determinada tarefa.

Os casos de uso cadastrais de baixa complexidade que envolvem as funcionalidades de criar, alterar, excluir e consulta, podem ser descritos na forma de tabela como proposto por (FALBO, 2012). Existem diferentes tipos de usuários no sistema, que são chamados de atores. Através dessa caracterização é possível identificar quais usuários podem e quais não podem executar determinada tarefa no sistema.

Ainda como produto dessa fase do desenvolvimento de um *software*, a modelagem conceitual estrutural visa modelar de forma computacional os requisitos e os casos de uso identificados anteriormente. Por meio deste modelo, é possível identificar de forma clara os tipos de entidade e os tipos de relacionamentos presentes no sistema. O propósito da modelagem conceitual estrutural, segundo o paradigma orientado a objetos, é definir as classes, atributos e associações que são relevantes para tratar o problema a ser resolvido (FALBO, 2012).

2.2 Projeto de Sistemas e Implementação

A fase de Projeto durante a especificação de um *software* tem por objetivo definir e especificar uma solução a ser implementada (FALBO, 2011). É a fase onde retira-se da etapa de levantamento e especificação de requisitos o "o que" deve ser feito e busca o "como" deve ser feito. É nesta fase que são determinadas as tecnologias de implementação e de armazenamento que serão utilizadas durante o processo de desenvolvimento do sistema. Esta fase é decomposta em etapas, sendo que a primeira consiste em definir uma arquitetura de *software*, seguido do detalhamento e projeto dos elementos da arquitetura e por fim o projeto detalhado.

A arquitetura de *software* de um programa ou sistema computacional é a estrutura ou estruturas do sistema, que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles (PRESSMAN, 2005), ou seja, a etapa de arquitetura de *software* tem como objetivo obter os relacionamentos dos componentes do sistema assim como identificar tais componentes. Ao final desta etapa, segue-se para o detalhamento que em (FALBO, 2011) é definida como tendo o objetivo de projetar em um maior nível de detalhes cada um dos elementos estruturais definidos na

arquitetura, o que envolve a decomposição de módulos em outros módulos menores. Com o resultado dessa etapa, já é possível identificar os primeiros módulos de implementação, onde no caso do paradigma orientado a objetos, seriam as classes.

Por fim, o detalhamento do projeto tem por objetivo refinar e detalhar os elementos mais básicos da arquitetura do *software*: as interfaces, os procedimentos e as estruturas de dados. Deve-se escrever como se dará a comunicação entre os componentes da arquitetura, a comunicação do sistema em desenvolvimento com outros sistemas e com as pessoas que irão utilizá-lo (FALBO, 2011).

Ao final da fase de Projeto de *software*, tem-se a arquitetura na qual o sistema será montado juntamente com o projeto dos componentes do sistema. Esses componentes, como proposto em (FOWLER, 2002), podem ser classificados da seguinte forma: interface com o usuário, lógica de negócio e persistência de dados. A interface com o usuário é o componente responsável por gerenciar todo o contato do usuário com o sistema, seja para receber dados quanto para apresentar dados. O componente de lógica de negócio é encarregado de aplicar todas as regras de negócios e requisitos do sistema. É uma camada intermediária de um sistema, que recebe os dados vindo da interface com o usuário e após aplicada a lógica de negócio repassa os dados para a camada seguinte. A persistência de dados tem como tarefa guardar os dados de forma a ser possível recuperá-los quando necessário em um sistema de banco de dados integrando com o sistema.

2.3 O método *FrameWeb*

O método *FrameWeb* foi proposto por (SOUZA, 2007) e trata-se de um método com o intuito de auxiliar a fase de Projeto na construção de sistemas de informação para a *web* que utilizem *frameworks*, devido ao fato de serem propostos modelos de projeto que retratem mais fielmente a implementação do sistema. Este método assume que existem tipos determinados de *frameworks* utilizados durante todo o resto do desenvolvimento do *software*, que engloba a definição de uma arquitetura básica para o sistema e a implementação.

Mesmo sendo um método focado na etapa de Projeto de Sistemas, o *FrameWeb* espera que toda as fases anteriores do desenvolvimento de *software* tenham sido concluídas e que diagramas de casos de uso e de classes de domínio tenham sido construídos. A figura 1 representa um processo de desenvolvimento de *software* sugerido pelo *FrameWeb*.

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

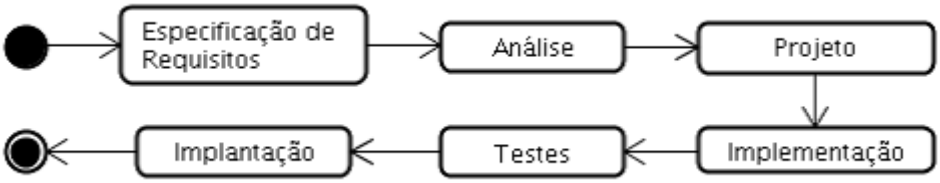

Figura 1 – Um processo de desenvolvimento de software simples sugerido pelo *FrameWeb*.

que os diagramas fiquem mais próximos da implementação (SOUZA, 2007).

Como se trata de um método que tem como objetivo ser uma alternativa na fase de Projeto de Sistemas incentivando o uso de *frameworks*, a parte da implementação se torna menos complicada. É possível simplificar o processo de desenvolvimento, teste e até implantação com a utilização dessas ferramentas tornando o processo de desenvolvimento do *software* uma tarefa um pouco menos complexa.

Assim como proposto em (FOWLER, 2002) o *FameWeb* define uma arquitetura de *software* lógica baseada no padrão camada de serviço, sendo decomposto em três camadas diferentes: lógica de apresentação, lógica de negócio e lógica de acesso a dados conforme indicado na figura 2.

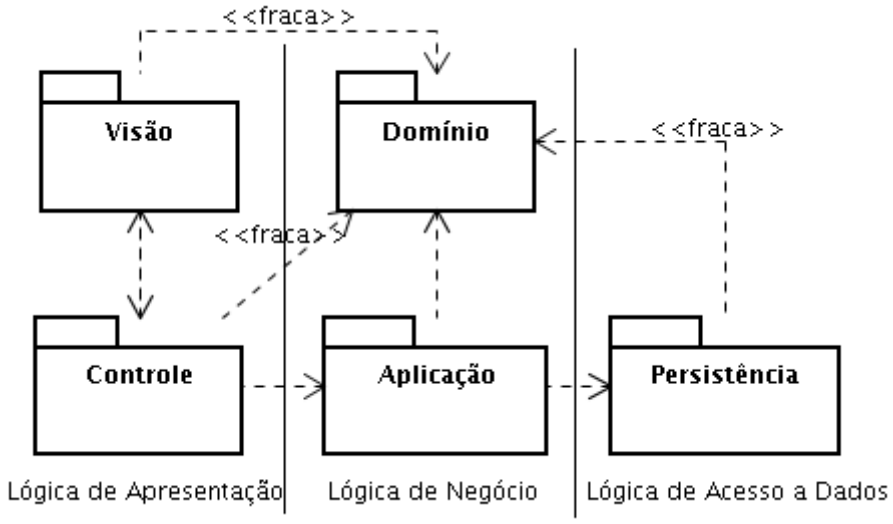

Figura 2 – Arquitetura padrão para Sistema de Informação *Web* baseada no padrão Camada de Serviço (FOWLER, 2002).

A lógica de apresentação é a camada que engloba a parte de interação com o usuário. O pacote Visão contém páginas *Web*, folhas de estilo, imagens, *scripts* que executam do lado do cliente, modelos de documentos e outros arquivos relacionados exclusivamente com a exibição de informações ao usuário. O pacote Controle envolve classes de ação e outros arquivos relacionados ao Controlador Frontal (SOUZA, 2007). Logo, toda a interação do usuário com o sistema é possível devido ao pacote de Visão e todo o gerenciamento dessas

informações, tais como, a obtenção de informações inseridas pelo usuário e disponibilização de informações para o usuário, são de responsabilidade do pacote Controle.

Em seguida, a camada de lógica de negócio contém os pacotes de aplicação e de domínio. No pacote de domínio encontra-se todas as classes que foram identificadas ainda na fase de Análise de Requisitos como sendo do domínio do problema. Já o pacote de aplicação é responsável por aplicar a lógica de negócio do sistema, sendo a implementação da parte lógica dos casos de uso. Tal camada se relaciona diretamente com os elementos de domínio do problema e com a lógica de apresentação. Os dados recebidos pelos controladores são passados para a camada de lógica de negócio através do pacote de aplicação, que por sua vez devolve a computação realizada para o pacote de controle. A relação entre o pacote de Controle e Domínio é estereotipada como fraca porque não são feitas alterações em entidades de domínio persistentes: objetos de Domínio são utilizados na camada de apresentação apenas para exibição de seus dados ou como parâmetros nas evocações de métodos entre um pacote e outro (SOUZA, 2007).

Por fim, a camada de Lógica de Acesso a Dados possui apenas o pacote de Persistência que é responsável por gerenciar a persistência de dados do sistema. O método *FrameWeb* assume a utilização de um *framework* de mapeamento objeto/relacional para a lógica de persistência de dados através do padrão de projeto *Data Access Object* (DAO) (ALUR et al., 2003). Com isso, toda a computação realizada pelo pacote de aplicação que necessita ser armazenada é enviada para o pacote de persistência. Da mesma forma, se para realizar uma computação a aplicação necessite de algum dado que esteja persistido, tal dado é solicitado a camada de Lógica de Acesso a Dados. A relação fraca entre o pacote de Persistência e o pacote de Domínio se dá pelo fato de os elementos do domínio serem necessários para a persistência apenas para efetuar o mapeamento para o banco de dados relacional.

Ao final da fase de Projeto utilizando o método *FrameWeb* é esperado que tenha como produto quatro modelos diferentes, a saber: Modelo de Domínio, Modelo de Persistência, Modelo de Navegação e Modelo de Aplicação. Tais modelos são extensões a UML propostos pelo método aqui discutido, com o intuito de representar mais detalhadamente os elementos considerados comuns para ferramentas *Web*. Cada modelo visa representar as informações levantadas durante o processo de desenvolvimento para todas as camadas da arquitetura de software. Os dados representados por cada modelo, segundo a especificação do método *FrameWeb* descrita por (SOUZA, 2007), são:

- Modelo de Domínio: digrama de classes da UML que representa os objetos de domínio do problema e seu mapeamento para a persistência em banco de dados relacional;
- Modelo de Persistência: diagrama de classes da UML que representa as classes DAO existentes, responsáveis pela persistência das instâncias das classes de domínio;

- Modelo de Navegação: diagrama de classe da UML que representa os diferentes componentes que formam a camada de Lógica de Apresentação, como páginas Web, formulários HTML e classes de ação;
- Modelo de Aplicação: diagrama de classes da UML que representa as classes de serviço, que são responsáveis pela codificação dos casos de uso, e suas dependências.

3 Especificação de Requisitos

Este capitulo apresenta a especificação de requisitos funcionais do Resgate – o Sistema de Despacho de Ambulância Automatizado. A elicitação de requisitos se deu a partir de análise de documentos, partindo dos requisitos elicitados em (SOUZA; MYLOPOULOS, 2013), e entrevistas com o primeiro autor deste mesmo artigo, orientador deste trabalho.

3.1 Descrição do Escopo

Quando um cidadão de uma cidade necessita de atendimento de emergência para ele ou para uma outra pessoa qualquer, a primeira coisa a ser feita é telefonar para uma central de emergência, como a de ambulância por exemplo. Ao realizar a chamada, o atendente da central verifica a gravidade da situação, colhe todos os dados necessários com o cidadão e solicita que uma ou mais ambulâncias façam o atendimento ao incidente transmitindo todas as informações necessárias.

Esse processo leva um certo tempo, pois o funcionário precisa encontrar uma ambulância que esteja bem localizada em relação ao local do incidente, que além disso atenda aos requisitos impostos, como quantidade de paramédicos por exemplo. Após localizar a ambulância, o funcionário precisa entrar em contato com ela e verificar sobre a disponibilidade ou não do atendimento para só depois de fato a ambulância ser despachada para o local do incidente. A medida que o atendimento vai avançando, os funcionários da ambulância têm de ir reportando o estado desse incidente a medida do possível.

Com o sistema que foi desenvolvido, toda essa tarefa de procurar por uma ambulância, entrar em contato com ela, atualizar estado e etc., passará a ser feito de forma automática, sendo necessário ao funcionário apenas confirmar, ou não, os dados sugeridos pelo sistema.

Desta forma, pode-se obter uma melhor precisão em relação ao posicionamento de ambulâncias, quantidade de ambulâncias necessárias para atendimento a incidentes, melhor utilização da equipe médica disponível nas estações e principalmente economia de tempo. O funcionário não mais precisará se comunicar diretamente com a central de ambulâncias e nem precisará esperar que a ligação com o cidadão termine para que ele possa transmitir os dados referentes a essa ligação para os outros funcionários responsáveis pelo atendimento. Enquanto o funcionário está obtendo as informações do incidente com o cidadão que efetuou a ligação, ele também já vai inserindo os dados no sistema e assim que completar todos necessários, o despacho de ambulância já poderá ser efetivado.

3.2 Modelo de Casos de Uso

A Tabela 1 representa como foi realizada a divisão em subsistemas do Resgate.

Tabela 1 – Subsistemas do Resgate

Subsistema Cadastro	Envolve toda a funcionalidade relacionada a cadastros do sistema. Engloba o cadastro de estações, hospitais, ambulâncias, equipamentos e funcionários.
Subsistema Despacho	Envolve a funcionalidade relacionada despacho de ambulâncias, gerenciamento de incidentes e de chamadas.

A Tabela 2 descreve os atores identificados para o sistema de despacho de ambulância automatizado e suas respectivas funções.

Tabela 2 – Descrição dos atores identificados

Adminstrador	Funcionário do órgão que presta serviço de emergência e responsável pela manutenção dos cadastros do sistema.
Funcionário	Funcionário do órgão que presta serviço de emergência responsável pelo atendimento ao cidadão e despacho de ambulância ou atendimento ao incidente.
MDT	Sistema presente nas ambulâncias responsável por receber as instruções de despacho e informar o estado atualizado do despacho.
Google *Maps*	Sistema de mapa que oferece o cálculo de distância entre posições geográficas.

O administrador é um funcionário do órgão de emergência que é responsável por toda a parte de cadastro (cadastro de ambulância, de funcionário, de hospital, de ambulância e de equipamento) e manutenção desses cadastros no sistema.

Já um funcionário pode ser um atendente que é responsável pelo atendimento telefônico ao cidadão que está requisitando o serviço de emergência. Esse tipo de funcionário é responsável por registrar chamadas incluindo dados relevantes passados pelo cidadão, assim como informar ao sistema quando um incidente está pronto ou não para ter ambulâncias enviadas. Dessa mesma forma, um administrador também pode receber ligações e despachar ambulâncias. Um funcionário também pode ser um tripulante da ambulância que efetuará o atendimento a um incidente. Esse tipo de funcionário, por sua vez, não tem acesso ao sistema.

O *Mobile Data Terminal* (MDT) é um sistema presente nas ambulâncias que é responsável por receber as instruções de despacho enviadas pelo Resgate, assim como informar o estado atualizado do despacho. É responsável também por enviar localizações geográficas periódicas da ambulância no formato de latitude e longitude para que se possa manter um controle.

O *Google Maps* é um sistema provido pela *Google Inc.* que é um serviço de mapa que será utilizado com o intuito de efetuar cálculos relacionados a distâncias entre posições geográficas.

As Figuras 3 e 4 contém respectivamente os diagramas de casos do sistema estando descrito quais são as funções que estão sendo abrangidas e quais os atores que poderão executá-las.

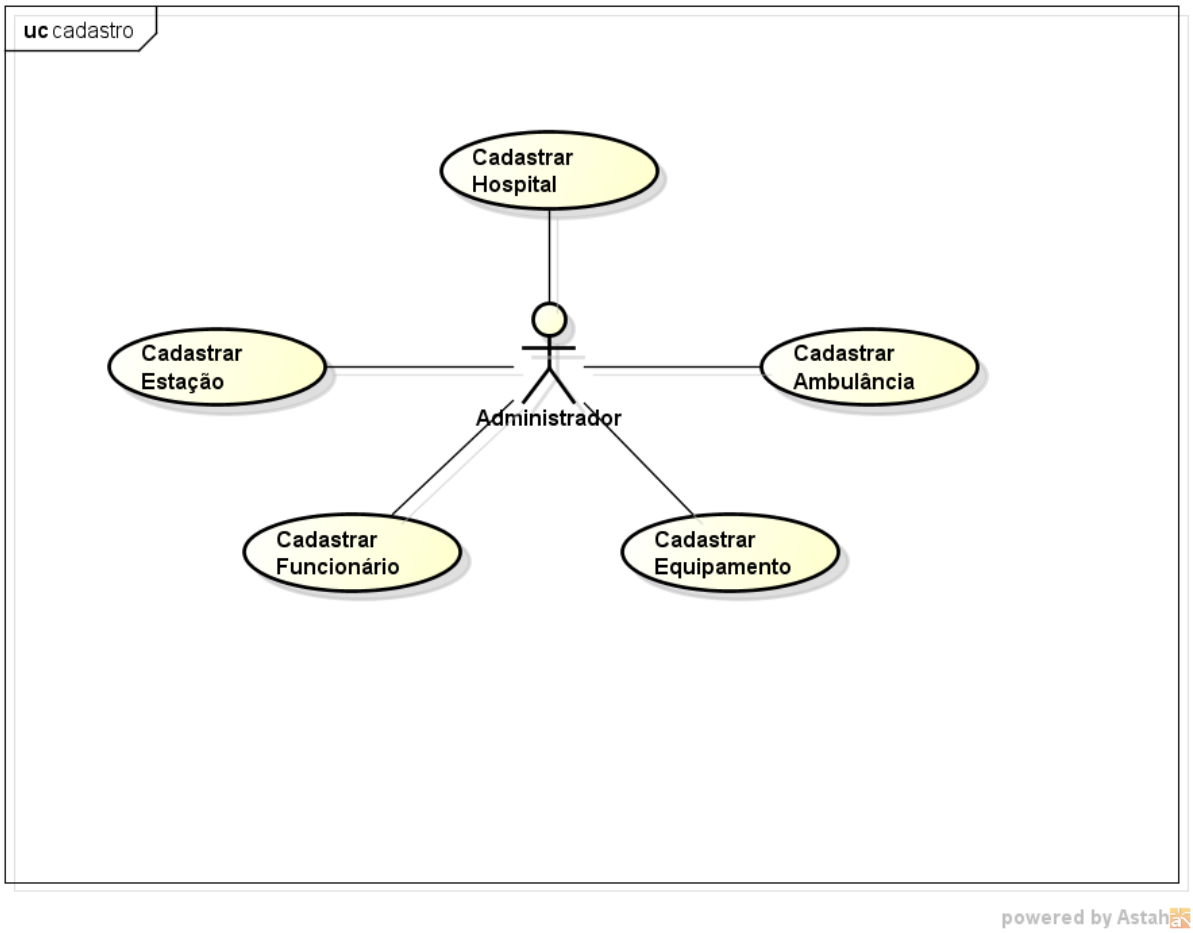

Figura 3 – Diagrama de casos de uso do ator **Adminstrador**.

Um usuário do sistema Resgate com o papel de administrador do sistema possui a autonomia para realizar as operações básicas de cadastro no sistema tais como, **Cadastrar Hospital**, **Cadastrar Ambulância**, **Cadastrar Equipamento**, **Cadastrar Funcionário** e , **Cadastrar Estação**. Esses casos de uso são considerados operações CRUD (*Create*, *Read*, *Update*, *Delete*), ou seja, o administrador do sistema é o único tipo de usuário que tem controle sobre criação, edição e deleção de tais tipos de dados.

Esses casos de uso de cadastros funcionam como base para o funcionamento do sistema, por meio dos quais todas as partes necessárias para efetuar o atendimento a cidadãos em situações de emergência são inseridas no sistema. Com isso, é possível ao administrador inserir novos funcionários de qualquer função e alocá-los em estações. Da

mesma forma, é possível alterar ou excluir algum funcionário que por algum motivo não faz mais parte do corpo de atendimento. Caso alguma ambulância apresente algum defeito momentâneo, é permitido ao administrador indicar que ela não está disponível para ser despachada, assim como para equipamentos. Também é permitido ao administrador do sistema cadastrar estações e associar às mesmas ambulâncias, funcionários e equipamentos, de forma a facilitar o gerenciamento e distribuição dos recursos pelo órgão prestador de serviços de emergência. Todos os hospitais que são passíveis de recebimento de cidadãos resgatados por ambulâncias devem também ser inseridos no sistema pelo administrador.

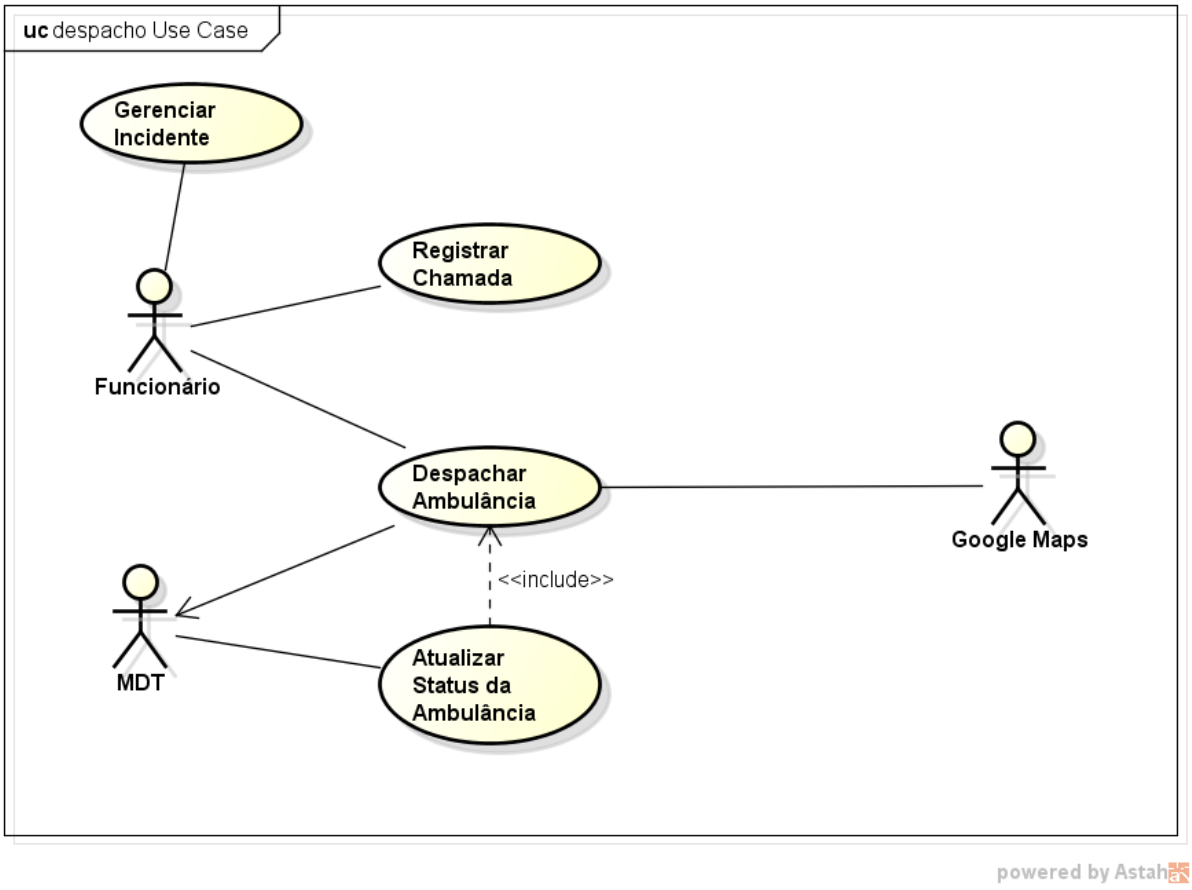

Figura 4 – Diagrama de casos de uso do ator **Adminstrador**.

Já um usuário com o papel de funcionário atendente tem como função no sistema o atendimento ao cidadão que está requisitando atendimento de emergência. A interação do funcionário no sistema inicia-se através do caso de uso **Registar Chamada**. Tal caso de uso começa quando o funcionário recebe uma nova chamada para auxiliar um cidadão em uma situação adversa. Os dados do incidente são coletados pelo funcionário e uma busca por chamadas similares levando em consideração a descrição da chamada, número de origem e nome do cidadão é realizada. Caso a busca encontre um resultado, o funcionário associa a nova chamada ao incidente já cadastrado no sistema, caso contrário após a confirmação dos dados é inserido um novo incidente.

O funcionário também pode **Gerenciar Incidente**, onde é possível alterar ou consultar dados de um incidente. Para esse caso de uso, o funcionário efetua uma busca no sistema a fim de obter o registro procurado e em seguida efetua as ações que deseja.

Também é permitido ao funcionário, **Despachar Ambulância**. Para isso, ele lista todos os incidentes abertos e indica qual deve ter ambulâncias direcionadas para atendimento. Em seguida, o sistema efetua verificações de ambulâncias, funcionários e equipamentos disponíveis em todas as estações. Apenas ambulâncias que estejam de acordo com as configurações indicadas pelo atendente durante o atendimento ao cidadão podem ser despachadas.

Uma vez obtidos todos os elementos necessários para atendimento (ambulâncias, funcionários e equipamentos), o sistema solicita ao ator Google Maps que identifique a distância entre as ambulâncias e o local do incidente. Após essa etapa, uma listagem em ordem da menor distância para a maior é mostrada ao funcionário para que ele decida quais ambulâncias devem ser despachadas. Depois de ter sido indicado pelo funcionário quais as ambulâncias devem ser despachadas para o incidente, o sistema envia as informações do despacho para o ator MDT que fica localizado dentro das ambulâncias. Ao final desse processo, o despacho de ambulâncias para um incidente foi concluído.

O MDT é uma ferramenta com software próprio que tem como tarefas ficar enviado notificações constantes ao sistema Resgate informando o estado de cada ambulância que esteja em atendimento (caso de uso **Atualizar Status da Ambulância**). Tais informações podem variar dependendo do estado do atendimento e podem ser:

- Posições geográficas: o MDT envia a latitude e longitude da ambulância em tempo real;
- Confirmar Atendimento ao Incidente: o MDT informa ao sistema que a ambulância está disponível e atenderá ao incidente conforme as instruções de despacho recebidas através do caso de uso **Despachar Ambulância**;
- Informar em rota para hospital: o MDT informa ao sistema que a ambulância já fez o socorro à vítima e segue a caminho do hospital indicado previamente pelo funcionário no caso de uso **Registrar Chamada**;
- Informar no hospital: o MDT informa ao sistema que a ambulância está deixando a vítima no hospital;
- Informar em rota para estação: o MDT informa que a ambulância já concluiu o atendimento ao cidadão e está retornando para estação para aguardar novas instruções de despacho;
- Informar em espera: o MDT informa que a ambulância já se encontra na estação e

está disponível para despacho;

- Informar problema com ambulância: o MDT informa que a ambulância apresenta algum problema inesperado e não poderá prosseguir com o atendimento. Em seguida, o funcionário é notificado do problema e o ciclo do caso de uso **Despachar Ambulância** é novamente iniciado.

A documentação técnica completa dos requisitos do sistema encontra-se nos Apêndices desta monografia.

3.3 Modelo Estrutural

O modelo conceitual estrutural visa capturar e descrever as informações (classes, associações e atributos) que o sistema deve apresentar para prover as funcionalidades descritas na Seção 3.2. Na prática, o modelo estrutural visa retirar as informações do mundo real e as colocar em um contexto de desenvolvimento de *software*. Neste modelo é possível identificar como que as ações executadas pelo usuário são tratadas no sistema.

A Figura 5 apresenta o diagrama de classe gerado para o subsistema **Cadastro** que engloba todas as funcionalidades básicas de cadastros para o sistema.

Por meio da análise do modelo estrutural tem-se uma visão mais clara das dependências entre as entidades do sistema, assim como o grau das relações. É ilustrado também quais são os possíveis tipos de funcionários (**StaffFunction**) e de tipos de equipamentos (**EquipmentType**) são possíveis de serem cadastrados no sistema. Vale ressaltar que o único ator que tem acesso a esse módulo é o **Adminstrador**.

Quando o **Administrador** do sistema efetua o cadastro de um novo hospital, por exemplo, o sistema cria um novo objeto da classe **Hospital** e atribui a esse objeto os dados preenchidos na tela de inclusão, tais como nome, rua, código postal, bairro e cidade. A mesma coisa vale para todos os outros cadastros, ou seja, quando alguma ação relacionada ao diagrama de caso de uso da Figura 3 é executada, o sistema cria um objeto da respectiva classe, registras as informações nessa nova instância e salva no banco de dados.

É importante também identificar restrições de integridade do sistema, limitações que não são possíveis de serem representadas no diagrama mas são importante para manter a consistência e integridade de dados a todo o tempo. Para esse módulo a seguinte restrição foi identificada:

- Funcionários (classe **Employee**) e equipamentos (**Equipment**) só podem ser alocados a ambulâncias (**Ambulance**) que pertençam à estação (**Station**) à qual o funcionário está alocado;

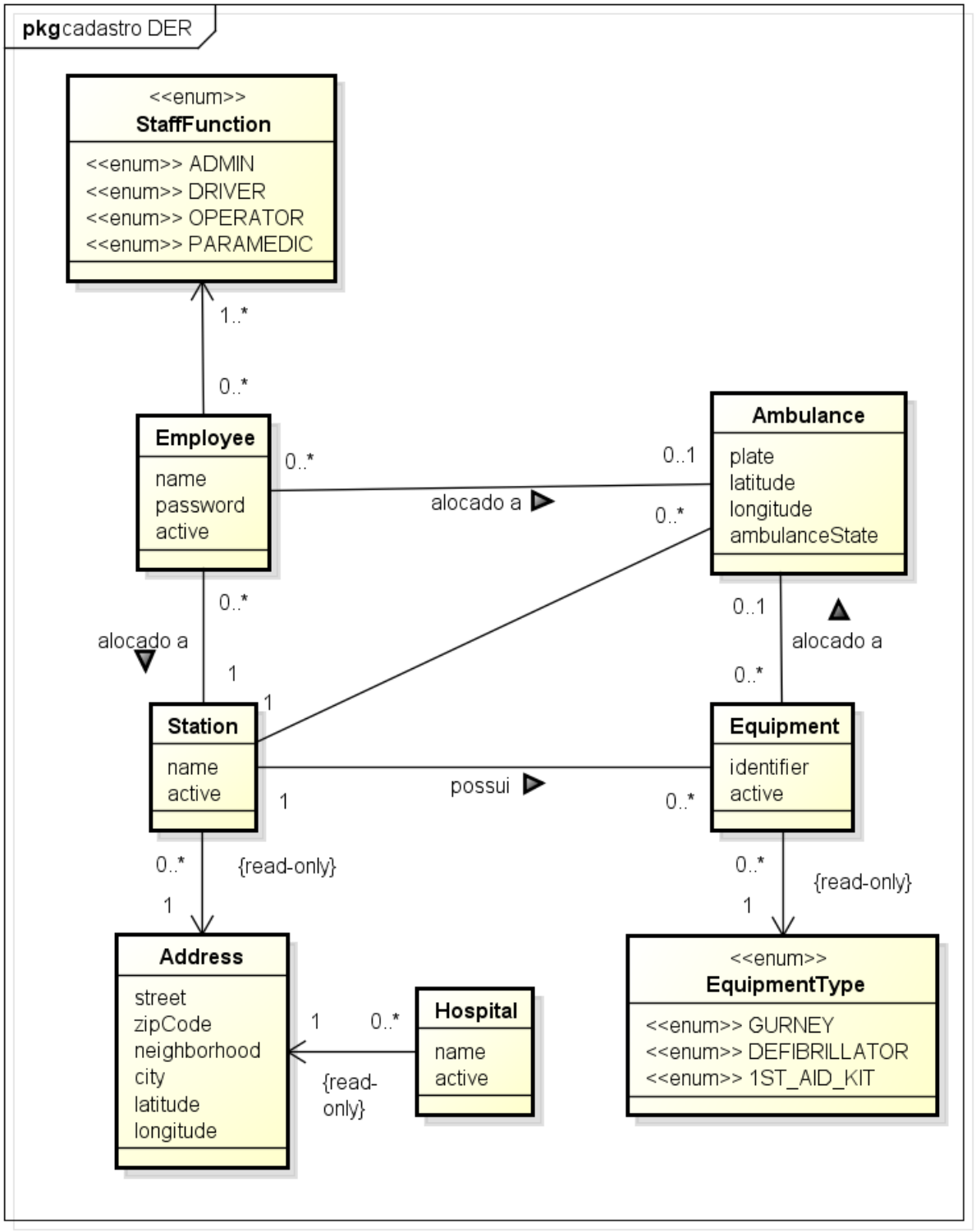

Figura 5 – Diagrama de classe do subsistema **Cadastro**.

Já a Figura 6 apresenta o diagrama de classe gerado para o subsistema **Despacho** que engloba todas as funcionalidades de despacho de ambulância, gerenciamento de incidentes e chamadas.

Quando um **Funcionário** recebe uma chamada de emergência, é criada uma nova instância da classe **EmergencyCall**, e o **Funcionário** registra a descrição do incidente.

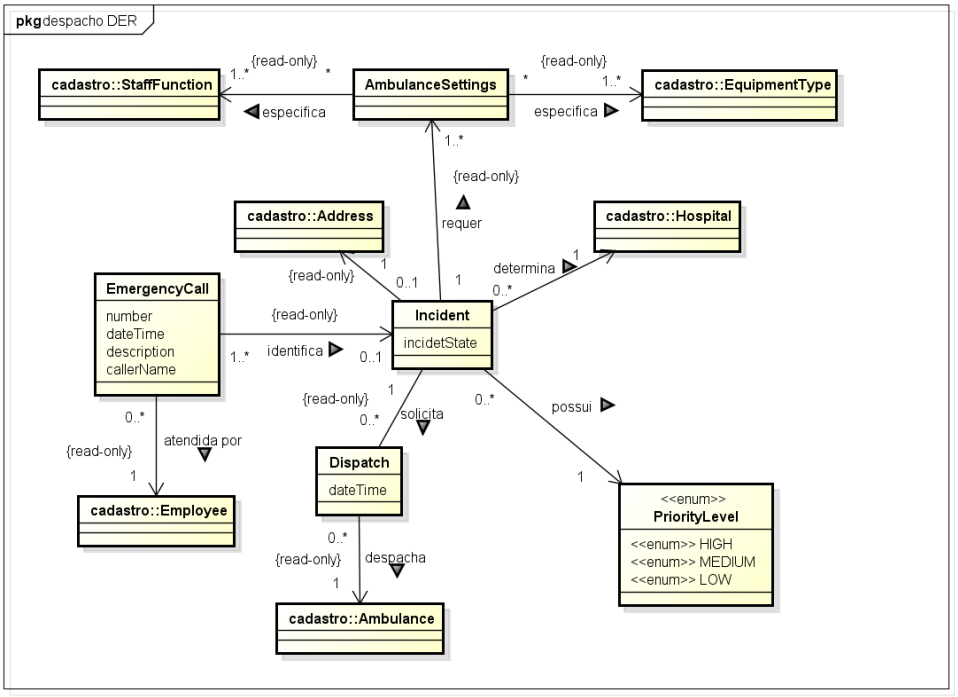

Figura 6 – Diagrama de classe do subsistema **Despacho**.

O sistema então, identifica o número de origem da ligação e a data/hora em que a chamada foi recebida e também registra esses dados nesse novo objeto criado. Como descrito no caso de uso **Registrar Chamada**, caso o **Funcionário** identifique a ligação como sendo de um incidente não registrado ainda no sistema, um novo objeto da classe **Incident** é criado e esse incidente é associado a chamada em questão. Por outro lado, se o incidente já existir, o objeto persistido é recuperado e associado a chamada.

Ao criar um novo incidente, o **Funcionário** também informa a qual hospital e quais são as configurações de ambulâncias necessárias para atendimento a situação de emergência. Essas duas conexões são feitas a partir de um objeto da classe **Incident** para um da classe **Hospital** e para um ou mais objetos **AmbulanceSettings**.

Ao autorizar o despacho uma ambulância, o **Funcionário** permite ao sistema efetuar a associação entre um objeto da classe **Incident** com um ou mais objetos da classe **Dispatch**. Isso se deve ao fato de que para um mesmo incidente, uma ou mais ambulâncias podem ser despachadas. Cada **Dispatch** se associa a uma **Ambulance**.

Assim como no diagrama do módulo anterior, existe uma restrição de integridade definida nesse subsistema, a saber:

- A data de despacho de um incidente tem que ser maior ou igual do que a data de criação desse incidente que fica registrado no objeto da classe **EmergencyCall**.

3.4 Modelo Dinâmico

O modelo dinâmico visa capturar o dinamismo de classes do sistema que fogem ao padrão *ativo* ou *inativo*. Em uma abordagem orientada a objetos, requisições de ação correspondem a mensagens trocadas entre objetos. As ações propriamente ditas e seus efeitos são tratados pelas operações das classes. Assim, a modelagem dinâmica está relacionada com as trocas de mensagens entre objetos e a modelagem das operações das classes (FALBO, 2012).

Um dos produtos dessa modelagem é o diagrama de estados de uma classe, uma máquina de estados que tem representação gráfica de todas as etapas em que um objeto pode transitar e também como ele pode transitar durante seu ciclo de vida. A Figura 7 representa o diagrama de estados para a classe **Ambulance**.

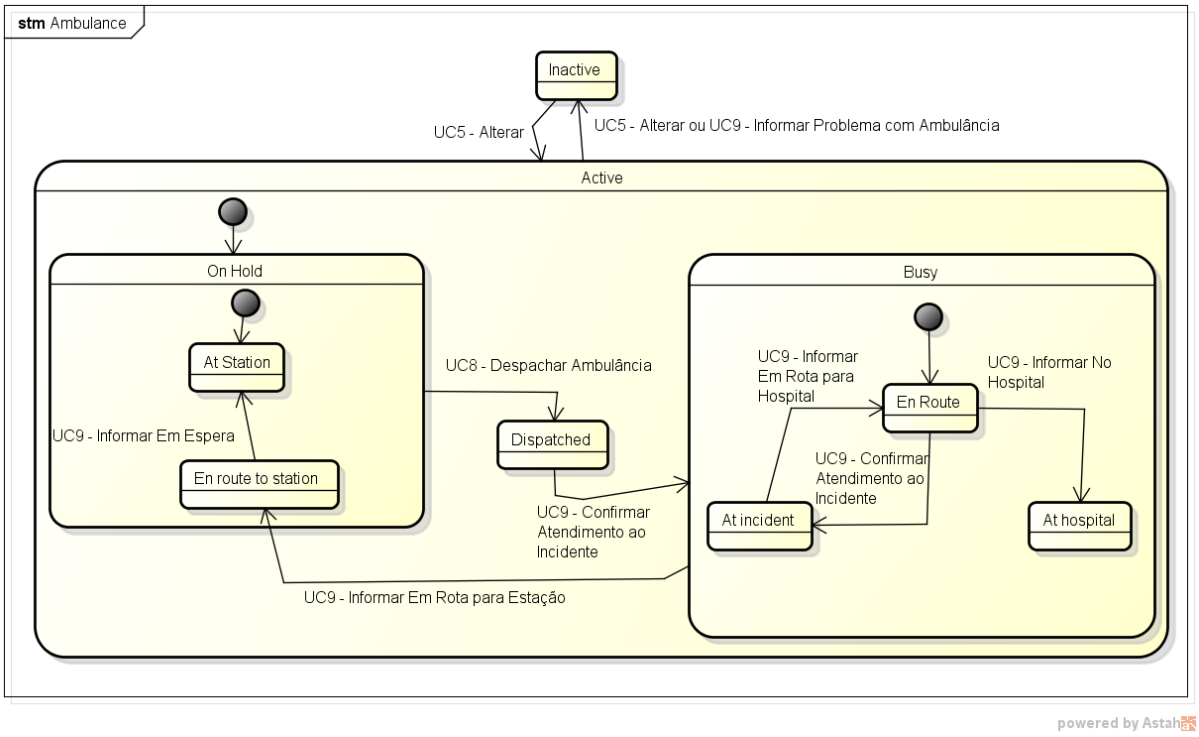

Figura 7 – Diagrama de estados da classe **Ambulance**.

Os objetos da classe **Ambulance** possuem dois grandes estados: **Inactive** e **Active**, que indicam se o objeto está ativo ou não no sistema. Essa característica pode ser alterada através da rotina estabelecida pelo caso de uso **Cadastrar Ambulância**.

Quando uma ambulância está esperando para ser despachada para um incidente, ela encontra-se no estado **At Station**. Através da rotina do caso de uso **Despachar Ambulância** o sistema permite ao atendente selecionar qual ambulância atenderá ao chamado e altera o estado para **Dispatched**. Já pelo caso de uso **Confirmar Atendimento ao Incidente** o ator **MDT**, ao confirmar o atendimento, faz com que o sistema altere o estado para **En Route**.

O **MDT** fica enviado constantemente para o sistema, a localização da ambulância através do caso de uso **Informar Posição Geográfica**. Ao receber essa localização o sistema faz uma verificação se a ambulância já chegou ao local do incidente e em caso positivo a ambulância tem seu estado alterado para **At Incident**.

Ao termino da prestação de socorro no local do incidente a ambulância segue com a vítima para o hospital indicado na configuração do incidente e o seu estado é alterado para **En Route**. Chegando ao hospital, o **MDT** informa onde a ambulância se encontra e o sistema, por sua vez, altera o estado para **At Hospital**. Por fim, após deixar a vítima aos cuidados da instituição médica, a instância tem seu estado alterado para **En Route to Station** e a partir desse momento a ambulância está novamente disponível para ser despachada e chegando na estação, o ciclo recomeça novamente.

Uma outra classe que também possui uma máquina de estados associada é a **Incident**, cujo diagrama encontra-se representado na Figura 8.

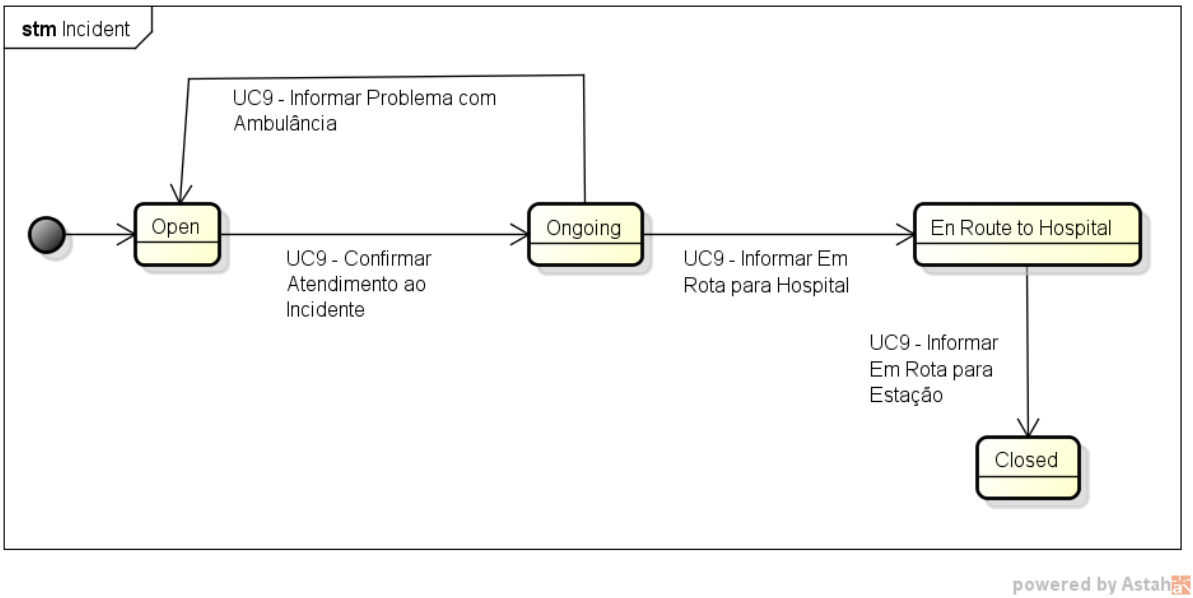

Figura 8 – Diagrama de estados da classe **Incident**.

Uma nova instância da classe **Incident** inicia seu ciclo de vida através do estado **Open**. Isso implica, que o incidente foi criado no sistema porém ele ainda está aguardando atendimento. Quando o ator **MDT** de alguma ambulância envia uma notificação para o sistema, por meio do caso de uso **Confirmar Antedimento ao Incidente**, o incidente passa para o estado **Ongoing** que indica que o atendimento foi iniciado.

Caso um incidente esteja sendo atendido por alguma ambulância que por motivo de força maior, não mais poderá seguir com a assistência ao cidadão (caso de uso **Informar Problema com Ambulância**), o estado do incidente volta a ser **Open**, ou seja, retorna ao inicio do ciclo. Por outro lado, se tudo ocorrer bem e o paciente for resgatado com sucesso pela ambulância, através do caso de uso **Informar Em Rota para Hospital**, o

estado é alterado para **En Route to Hospital**. Por fim, quando o paciente é deixado no hospital e a ambulância terminou o atendimento ao incidente, o estado é alterado para **Closed** e nenhuma ação mais é passível de ser realizada em cima deste incidente, a não ser a de consulta, e esse objeto não terá mais seu estado alterado sob qualquer circunstância.

4 Projeto Arquitetural e Implementação

O projeto é o processo criativo de transformar uma especificação de um problema em uma especificação de uma solução. No projeto de *software* utilizam-se a especificação e os modelos de requisitos gerados na fase de análise e especificação de requisitos. A partir dos requisitos, muitas soluções são possíveis e, portanto, muitos projetos diferentes podem ser produzidos. Uma solução é considerada adequada ao problema; se ela satisfizer a todos os requisitos especificados (PFLEEGER, 2004). Assim, o projeto é também uma atividade de tomada de decisão. Em suma, após ter analisado o problema, pode-se decidir como projetar a solução (FALBO, 2011).

Este capítulo descreve sucintamente o Projeto de Arquitetural e a Implementação da ferramenta Resgate. Na seção 4.1 estão descritas as tecnologias utilizadas para desenvolvimento do sistema, já na seção 4.2 está descrita a arquitetura de *software* sob a qual o sistema foi construído e a seção 4.3 descreve os modelos *FrameWeb* do sistema Resgate.

4.1 Tecnologias Utilizadas

A linguagem de programação escolhida para ser realizada a implementação do projeto foi Java™ EE 7 (Java Enterprise Edition 7) devido a independência da plataforma e por oferecer, de forma nativa, suporte a diversos *frameworks* e *APIs*.

O JSF ou *JavaServer Faces* (MANN, 2005) é uma *API* para construção de interfaces para usuários baseada em componentes para aplicações *web*. Com o auxilio dessa *API* é possível efetuar a criação das páginas para internet utilizando componentes visuais pré-prontos e a integração com o restante da aplicação.

JSF é um padrão Java™ para desenvolvimento *web* e faz parte do Java™ EE utilizado nesse trabalho. Tal padrão, possui um modelo de programação que permite a abstração de componentes HTML e a separação entre as camadas de apresentação e de aplicação.

O CDI ou *Contexts and Dependency Injection for Java™ EE* é um *framework* gratuito para a injeção de dependências disponível na plataforma de desenvolvimento Java™ EE 7. Com esta ferramenta é possível integrar as diferentes camadas de arquitetura e serviços de transação.

Segundo (JENDROCK et al., 2014), CDI é um conjunto de serviços que, combinados, facilitam o trabalho de desenvolvedores por permitir, entre outras coisas, que páginas JSF façam referências a essas classes injetadas utilizando linguagem de expressões unificada.

Para acesso a banco de dados foi utilizada a *API* JPA (*Java*^{TM} *Persistence API*) (DEMICHIEL; KEITH, 2006), que é uma *API* para persistência de dados por meio de mapeamento objeto-relacional das classes de domínio que possuem a anotação *@Entity* em sua declaração.

Após a definição de JPA como sendo um padrão a ser seguido para desenvolvimento em Java™, diversas implementações surgiram ou foram atualizadas para se adequar a nova realidade, como foi o caso do *Hibernate* (BAUER; KING, 2005). Para o desenvolvimento do projeto Resgate, foi utilizada essa implementação do padrão que abstrai o código SQL e toda a camada JDBC[1], uma vez que todas as expressões em SQL são geradas em tempo de execução pela própria ferramenta.

Nemo-utils é uma ferramenta desenvolvida pelo Núcleo de Estudos em Modelagem Conceitual e Ontologias (NEMO) da Universidade Federal do Espírito Santo (UFES) que tem como objetivo auxiliar a implementação de aplicações *web* que utilizam os *frameworks* JSF, JPA e CDI.

A ferramenta provê uma base sólida para as três camadas de desenvolvimento: interface com usuário, lógica de negócio e persistência. Para poder utilizar o *framework* o Resgate foi desenvolvido seguindo o padrão estabelecido pelo *nemo-utils* afim de tirar o máximo proveito e minimizar o tempo gasto com funcionalidades relacionadas a cadastro. Na Seção 4.3 o *framework* e sua integração com o Resgate são mostrados de forma mais sucinta.

4.2 Arquitetura de *Software*

A arquitetura de *software* da ferramenta Resgate baseia-se na combinação de camadas e módulos (ou subsistemas). O sistema foi dividido em dois módulos chamados de **Cadastro** e **Despacho**. Cada um desses módulos, por sua vez, está organizado em três camadas seguindo o modelo proposto pelo método *FrameWeb*, a saber: Lógica de Apresentação, Lógica de Negócio e Lógica de Acesso a Dados (vide Seção 2.3, Figura 1).

4.2.1 Divisão dos módulos

A Figura 9 contém a divisão em módulos da ferramenta construída. O módulo **Cadastro** tem como base o pacote **br.ufes.inf.nemo.resgate.data**, enquanto o módulo **Despacho** tem por base o pacote **br.ufes.inf.nemo.resgate.dispatch**.

Os pacotes **br.ufes.inf.nemo.resgate.data.controller** e **br.ufes.inf.nemo.res-**

[1] *Java*^{TM} *Database Connectivity*: é uma API que possibilita a conexão entre uma aplicação Java™ e um banco de dados. Mais informações em <http://www.oracle.com/technetwork/java/overview-141217.html>.

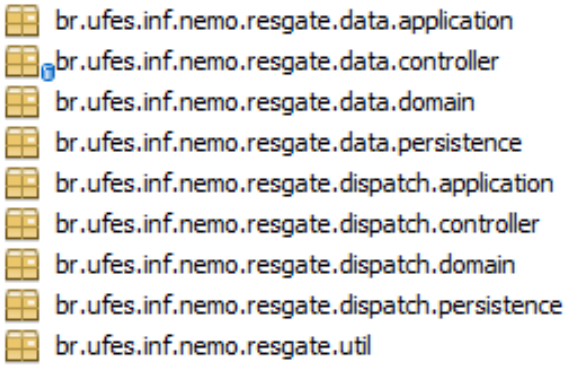

Figura 9 – Subdivisão dos módulos **Cadastro** e **Despacho** de acordo com a arquitetura de *software*.

gate.dispatch.controller contém as classes de controle do sistema que fazem parte da camada de Lógica de Apresentação, responsáveis por capturarem estímulos do usuário e enviar dados para serem processados na camada de Aplicação.

Já os pacotes **br.ufes.inf.nemo.resgate.data.application** e **br.ufes.inf.nemo.resgate.dispatch.application** são responsáveis por receberem os dados enviados pelas camadas e controle e de persistência e aplicar os casos de uso identificados na fase de análise.

A persistência fica por conta dos pacotes **br.ufes.inf.nemo.resgate.data.persistence** e **br.ufes.inf.nemo.resgate.dispatch.persistence**, que têm como objetivo efetuar a comunicação entre o sistema e o banco de dados.

4.3 Modelos *FrameWeb*

Esta seção contém os modelos do projeto do sistema Resgate baseado no método *FrameWeb* como descrito na Seção 2.3: Modelos de Domínio, Modelos de Navegação, Modelos de Aplicação e Modelos de Persistência.

4.3.1 Modelo de Domínio

Conforme apresentado na Seção 2.3, o Modelo de Domínio é um diagrama de classes da UML que representa os objetos de domínio do problema e seu mapeamento para a persistência em um banco de dados relacional. A partir dele são implementadas as classes de camada de domínio na atividade de implementação (SOUZA, 2007).

Todas as classes de domínio do Resgate herdam de *PersistentObjectSupport* da ferramenta *nemo-utils* e não sofrem alterações consideráveis. Devido a esse fato, a parte cadastral do sistema se comporta de forma muito similar à proposta pelo *framework*. A Figura 10 ilustra o Modelo de Domínio básico da ferramenta.

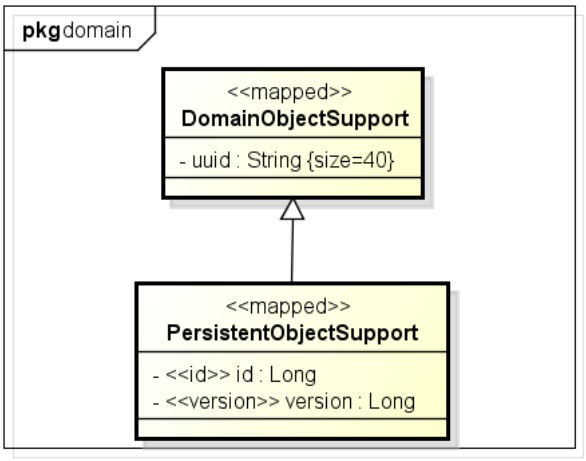

Figura 10 – Modelo de Domínio básico da ferramenta *nemo-utils*.

Pela definição de herança de programação orientada a objetos em Java™, as classes de domínio do Resgate consequentemente terão também os mesmos atributos da superclasse e seguirão a mesma estrutura apresentada.

As Figuras 11 e 12 apresentam os Modelos de Domínio para os dois módulos que compõem a ferramenta desenvolvida. Para fins de simplificação, a herança estabelecida entre as classes do Resgate e do *nemo-utils* não foram representadas no diagrama.

Diferente da abordagem original do *FrameWeb* proposto por SOUZA (2007), todos os atributos que não podem ser nulos tiveram a *tag {not null}* omitida e os que podem ser nulos tiveram a *tag {null}* acrescida de forma a diminuir a poluição visual com repetições desnecessárias no diagrama. Já a *tag {read-only}* indica que uma vez estabelecida a relação entre instância das classes, ela não poderá mais ser alterada e nem rompida. Vale ressaltar que tal notação não faz parte do escopo do método *FrameWeb*.

O modelo de domínio consiste no diagrama de classe montado na fase de Análise de Requisitos, com o acréscimo de informações acerca da plataforma de desenvolvimento escolhida, como tipos de atributos e navegabilidade de associações. Também são especificados dados a respeito da persistência das classes de domínio. A *tag {size=150}*, por exemplo, indica que o tamanho máximo de caracteres que o campo pode possuir no banco de dados é de 150 e já *tag {precision=timestamp}* indica que tanto a data quanto a hora devem ser guardadas.

Com a definição das tecnologias a serem utilizadas, também é possível definir os tipos enumerados presentes na ferramenta. O tipo **AmbulanceState** indica os estados pelo qual uma ambulância pode transitar, conforme ilustrado na Figura 7, e o **IncidentState** mostra os possíveis estados de um incidente, apresentado na Figura 8. Já o tipo **StaffFunction** contém as funções que cada funcionário pode assumir, a saber: administrador, motorista, atendente telefônico e paramédico. **EquipmentType** identifica o tipo de um equipamento presente em estações e em ambulâncias, sendo que seus possíveis valores são: maca,

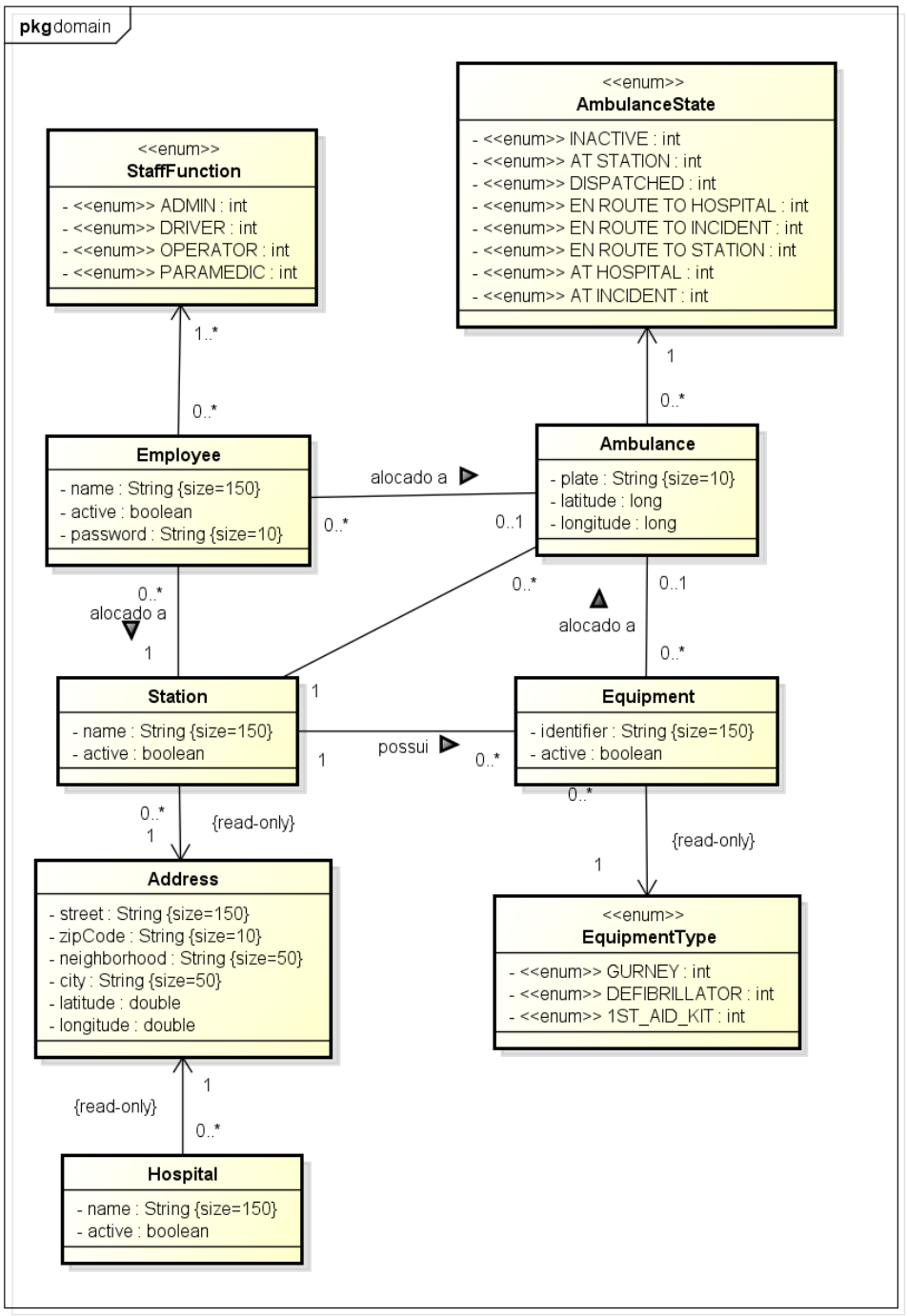

Figura 11 – Modelo de Domínio para o pacote **Cadastro**.

desfibrilador e kit de primeiros socorros. Por fim, **PriorityLevel** indica o nível de prioridade do incidente podendo ser alta, média ou baixa. Em eventuais alterações ou melhorias do sistema, esses tipos enumerados podem ter valores acrescidos facilmente.

4.3.2 Modelo de Navegação

Seguindo o definido na Seção 2.3, o Modelo de Navegação é um diagrama de classe UML que representa os diferentes componentes que formam a camada da Lógica

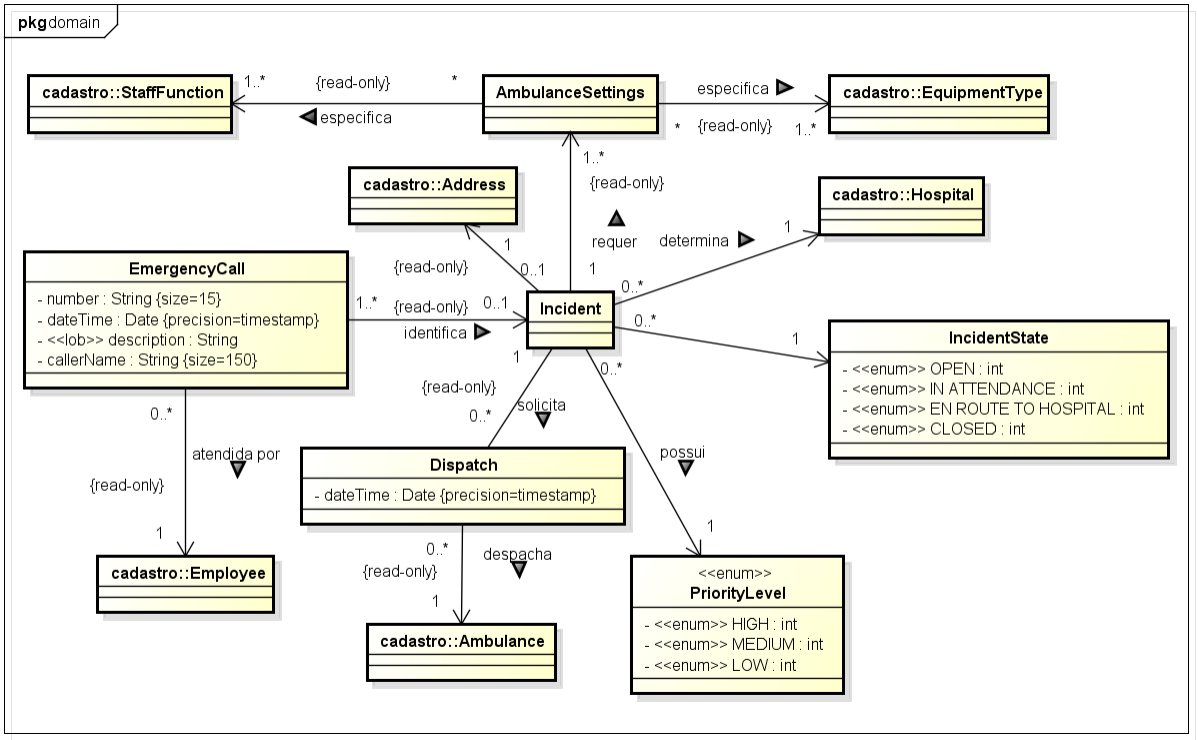

Figura 12 – Modelo de Domínio para o pacote **Despacho**.

de Apresentação, como páginas *Web*, formulários HTML e classes de ação do *framework* controlador frontal (SOUZA, 2007).

Conforme a especificação original do *FrameWeb*, vários esteriótipos podem ser utilizados para representar informações nos modelos. Para os Modelos de Navegação contidos neste projeto, a Tabela 3 provê o significado de cada componente presente.

Tabela 3 – Componentes do Modelo de Navegação

Componente	**Significado**
Sem esteriótipo	Uma classe de ação, para qual o *framework* controlador delega a execução da ação.
«page»	Uma página *web* estática ou dinâmica.
«form»	Um formulário HTML.

Os Modelos de Navegação são definidos por caso de uso, implicando que cada caso de uso terá o seu próprio modelo. As classes do subsistema **Cadastro** apresentam uma semelhança muito forte por serem classes que representam cadastros no sistema e todas herdam de *CrudController* da ferramenta *nemo-utils*. Por isso, para todas essas classes foi montado um único Modelo de Navegação genérico como mostrado na figura 13, que também funciona como um Modelo de Navegação para o próprio *nemo-utils*. Para fins de simplificação, a herança estabelecida entre as classes do Resgate e do *nemo-utils* não foram representadas no diagrama.

A navegação do usuário pelo modelo descrito na Figura 13 começa quando ele

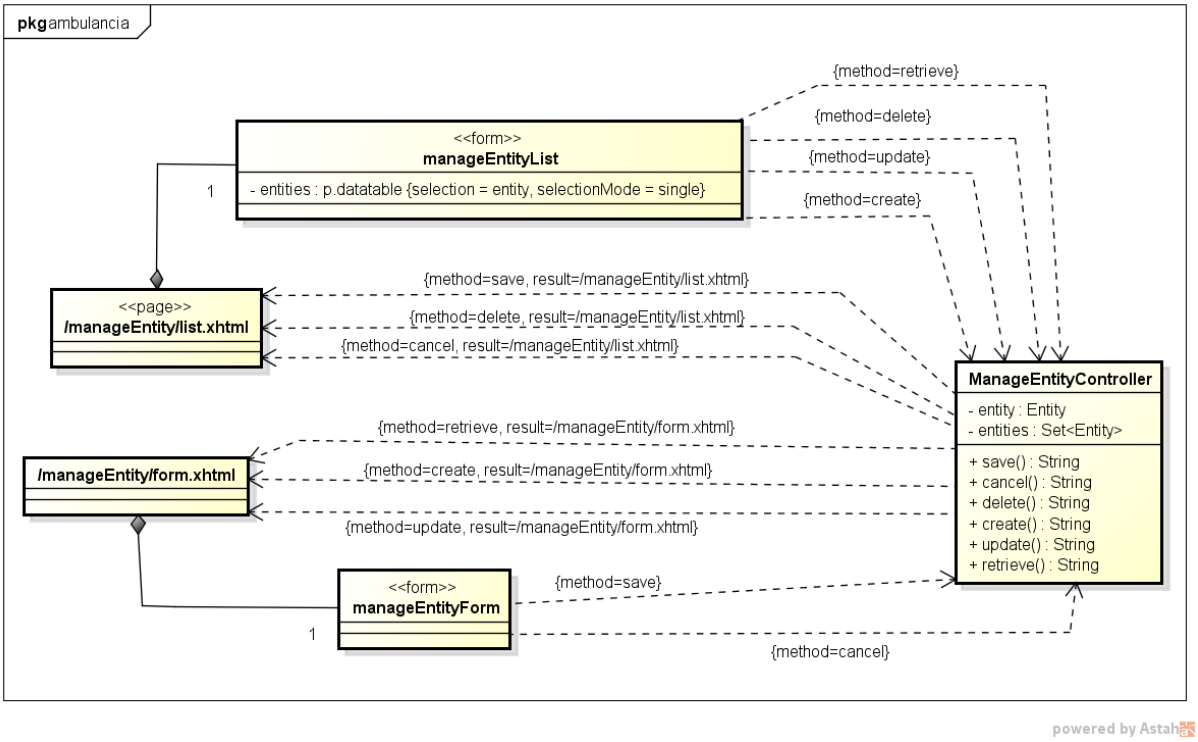

Figura 13 – Modelo de Navegação para o pacote **Cadastro**.

seleciona a opção de listar todos os dados de uma entidade pertencente ao subsistema **Cadastro**. Ao fazer isso, o sistema buscará a página *web* presente no diretório */manageEntity/* chamada *list.xhtml*. Para cada página deste tipo, existe um formulário que contém todos os campos necessários para o cadastro da entidade e este recebe o nome *manageEntityList*.

Como o modelo funciona de forma genérica, apenas o atributo que contém os valores a serem listados foi mostrado. Este campo contém duas *tags: {selection: entity}, {selectionMode: single}*, onde a primeira determina qual será o nome do atributo, na classe de controle (*ManageEntityController*), que receberá o objeto selecionado pelo usuário na tabela de dados (se houve seleção) e a segunda indica que apenas um objeto por vez poderá ser selecionado. Vale ressaltar que tais *tags* não fazem parte da proposta original de *FrameWeb*, mas são derivadas dos atributos do componente visual *p.datatable* utilizado.

Durante o carregamento da página, o controlador é responsável por solicitar a camada de aplicação os dados a serem exibidos e esta, por sua vez, é responsável por solicitar a camada de acesso a dados tais informações. De posse dos elementos a serem mostrados, o controlador termina de carregar a página e o usuário pode escolher qual ação deseja tomar em sequência. Caso exista algum objeto selecionado na tabela, as opções de visualizar, modificar, criar e deletar serão exibidas e caso contrário apenas a opção de criar uma nova entrada estará habilitada. Para cada possível ação escolhida pelo usuário, existe um método do controlador responsável por executar a tarefa selecionada. Pegando,

por exemplo, a opção de criar um novo objeto, o método **create** é chamado e este carrega a página *form.xhtml* que está dentro do mesmo diretório em que se encontra a página de listagem.

Da mesma forma que para a página anterior, existe também um formulário que contém todos os campos necessários para o cadastro de uma nova entidade, chamado *ManageEntityForm*. Após o preenchimento dos campos e ao clicar em salvar os dados, o método *save* de controlador é chamado e este efetua a transição entre as camadas até os dados serem persistidos. Em seguida, a página de listagem é novamente exibida.

Já para o subsistema **Despacho** foi montado o Modelo de Navegação para os casos de uso **Despachar Ambulancia** (Figura 14) e **Registrar Chamada** (Figura 15).

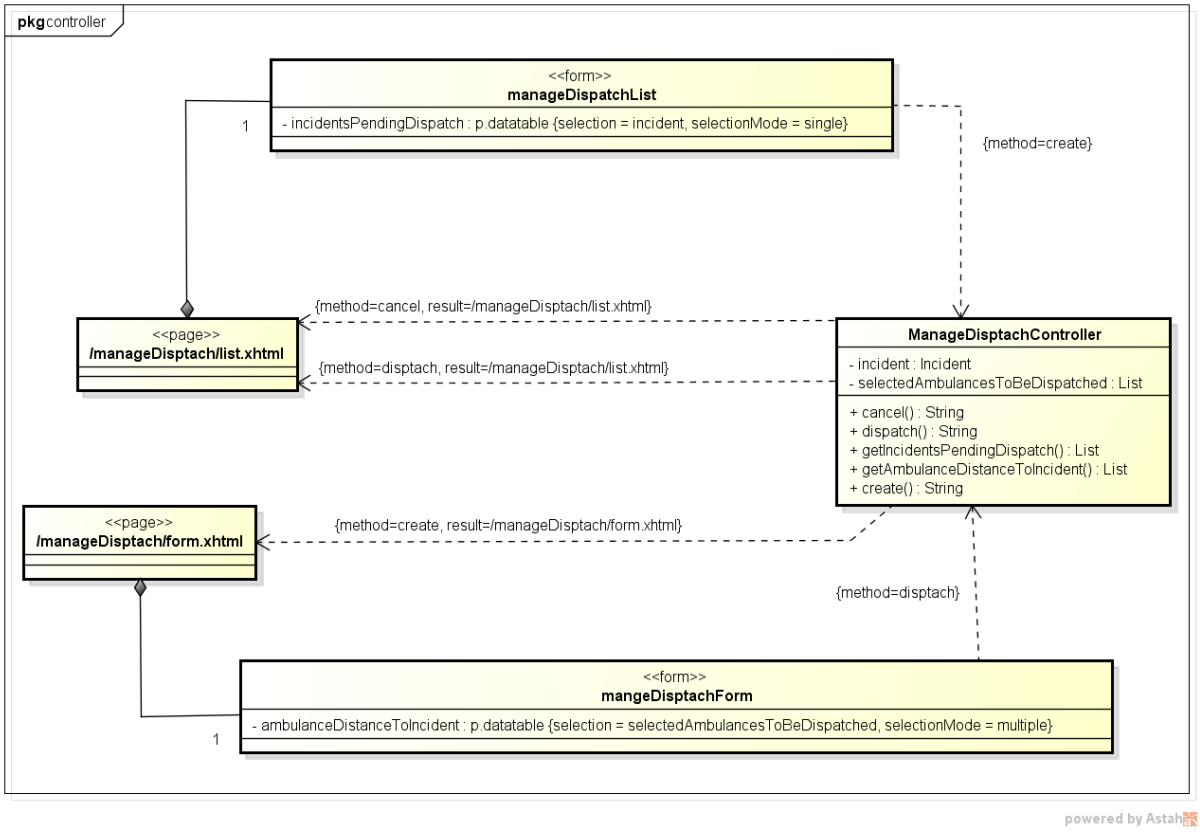

Figura 14 – Modelo de Navegação para o caso de uso **Despachar Ambulância**.

Para os modelos representados nas Figuras 14 e 15, a navegação pelo diagrama se dá de forma similar à apresentada na Figura 13, salvo algumas peculiaridades. No modelo apresentado na Figura 14 ao abrir a tela de listagem, o usuário não tem outra opção a não ser a ação de despachar ambulâncias para o incidente selecionado. Tal operação, ocorre por meio da chamada do método *create* presente na classe *ManageDispatchController* que, por sua vez, apresenta para o usuário a página que contém o formulário para despacho. Conforme descrito no caso de uso **Despachar Ambulância**, após solicitar o despacho para incidentes, o sistema lista as ambulâncias disponíveis em ordem crescente de distância e essa informação é mostrada no formulário *manageDispatchForm*. Ao selecionar as ambulâncias

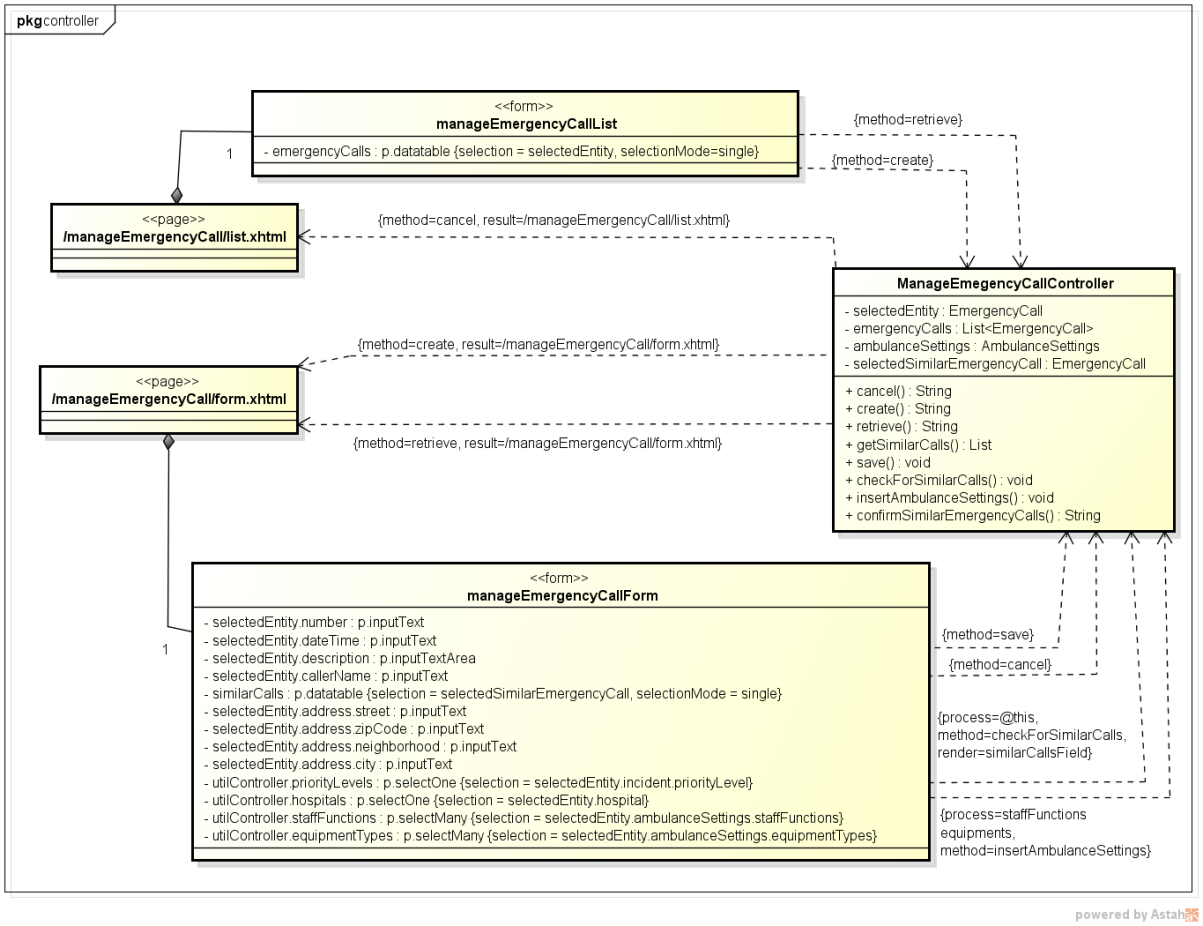

Figura 15 – Modelo de Navegação para o caso de uso **Registrar Chamada**.

para despacho, o método *dispatch* realiza as chamadas para a classe responsável pela execução do caso de uso e assim que obtém uma resposta, a tela de listagem é novamente renderizada.

Já o modelo apresentado na Figura 15 se difere dos dois anteriores como visto no componente *manageEmergencyCallForm*. Neste formulário, é possível verificar todos os campos que devem ser preenchidos pelo usuário para a criação de uma chamada de emergência no sistema. Vale destacar, que este contém duas requisições AJAX (GARRETT et al., 2005) que são responsáveis por verificar a existência de chamadas similares (conforme descrito no caso de uso **Registrar Chamada**) e por inserir configurações de ambulâncias necessárias para o incidente a ser criado através dos métodos *checkForSimilarCalls* e *insertAmbulanceSettings* respectivamente presentes no controlador *ManageEmergencyCallController*. Chamadas AJAX também não foram contempladas na proposta original de *FrameWeb* e são tema da pesquisa atual sobre o método em (MARTINS; SOUZA, 2015).

4.3.3 Modelo de Aplicação

Conforme apresentado na Seção 2.3, o Modelo de Aplicação é um diagrama de classes UML que representa as classes de serviço, que são responsáveis pela codificação dos casos de uso e seus dependências (SOUZA, 2007). Este modelo é utilizado como um guia de implementação de classes e interfaces dos pacotes da aplicação e na configuração de injeção de dependência.

Através do Modelo de Aplicação, é possível identificar como cada classe se relaciona a fim de prover a funcionalidade especificada em cada caso de uso. Todas as classes de aplicação do sistema herdam de *CrudServiceBean* do pacote *nemo-utils*. A Figura 16 representa o modelo básico de aplicação da ferramenta.

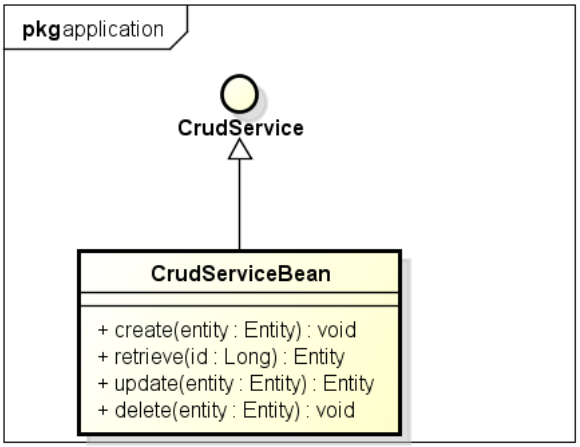

Figura 16 – Modelo de Aplicação básico da ferramenta *nemo-utils*.

Já as Figuras 17 e 18 representam os Modelos de Aplicação para os subsistemas do Resgate. Para fins de simplificação, a herança estabelecida entre as classes do Resgate e do *nemo-utils* não foram representadas no diagrama.

Através do Modelo de Aplicação apresentado na Figura 17, é possível identificar todas as interfaces e classes de aplicação do subsistema **Cadastro** e de qual interface de persistência elas dependem. Por ser um módulo focado no cadastro de elementos no sistema, nenhuma dessas classes possui métodos diferentes dos oferecidos pelo *framework nemo-utils*. As ligações entre as classes do diagrama indicam a interdependência entre as três camadas que compõe o sistema através dos prefixos *controller::* e *persistence::* presentes no diagrama que apresentam a conexão entre a camada de aplicação e interface com o usuário e entre a de aplicação e de acesso a dados respectivamente, e essas dependências são satisfeitas pelo CDI automaticamente.

Pegando como exemplo o caso de uso **Cadastrar Hospital**, a transição dos dados ocorre da seguinte forma: o controlador da página de cadastro do hospital (*ManageHospitalController*) recebe uma solicitação de criação de um novo hospital no sistema pelo

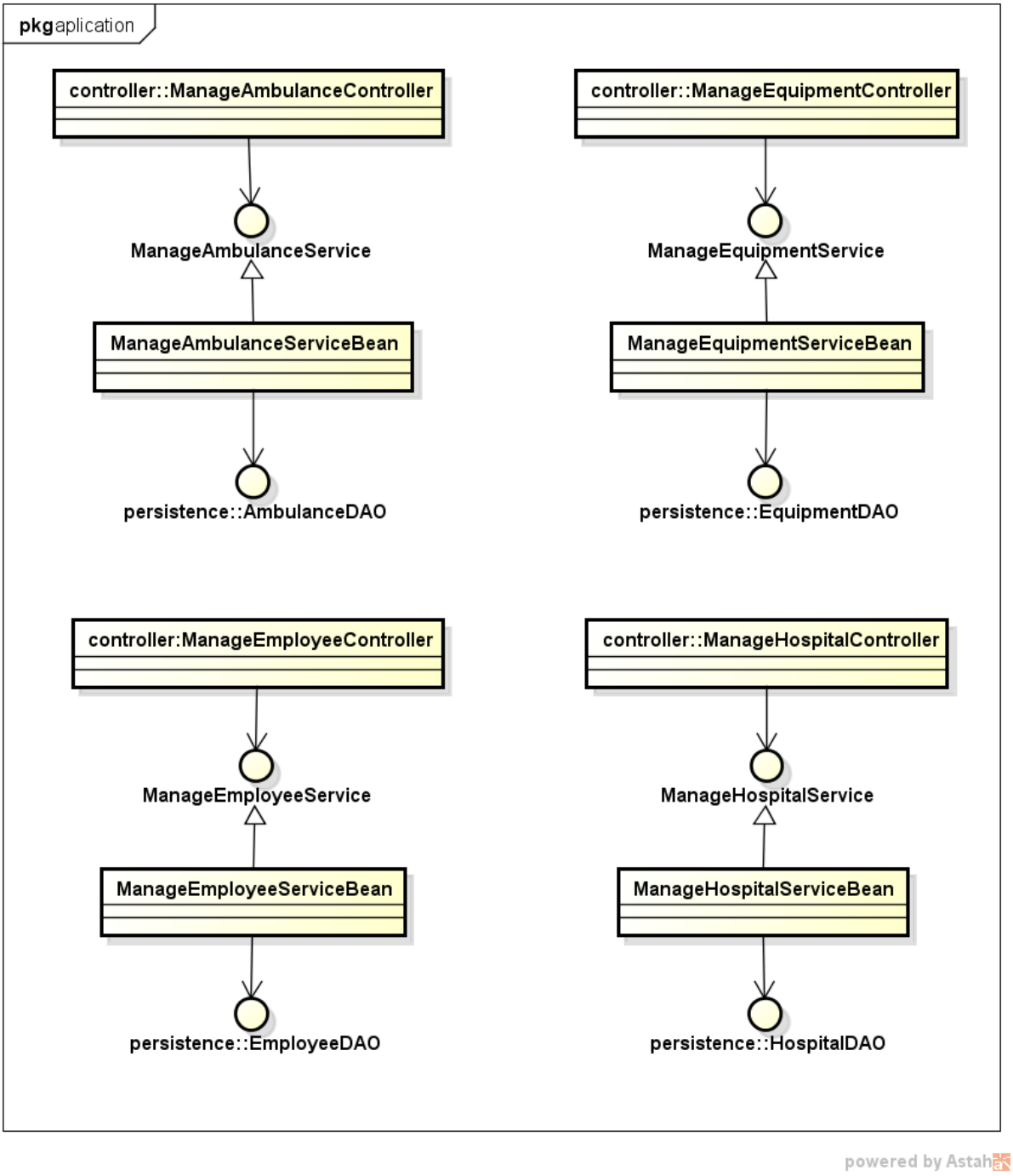

Figura 17 – Modelo de Aplicação para o módulo **Cadastro**.

usuário. Essa classe possui uma dependência com a *ManageHospitalServiceBean* através da interface *ManageHospitalService* que contém os métodos necessários para a execução do caso de uso. Essa associação é satisfeita através da injeção de dependência provida pelo CDI permitindo assim, ao controlador, solicitar a camada de aplicação a execução do caso de uso. Nesta etapa, são realizadas verificações e validações do dado a ser inserido no sistema, de acordo com o especificado pelo domínio do problema e caso esteja tudo de acordo com o que foi definido, a camada de aplicação, também através da injeção de dependência, repassa os dados para a camada de acesso a dados para que esta possa

realizar a persistência.

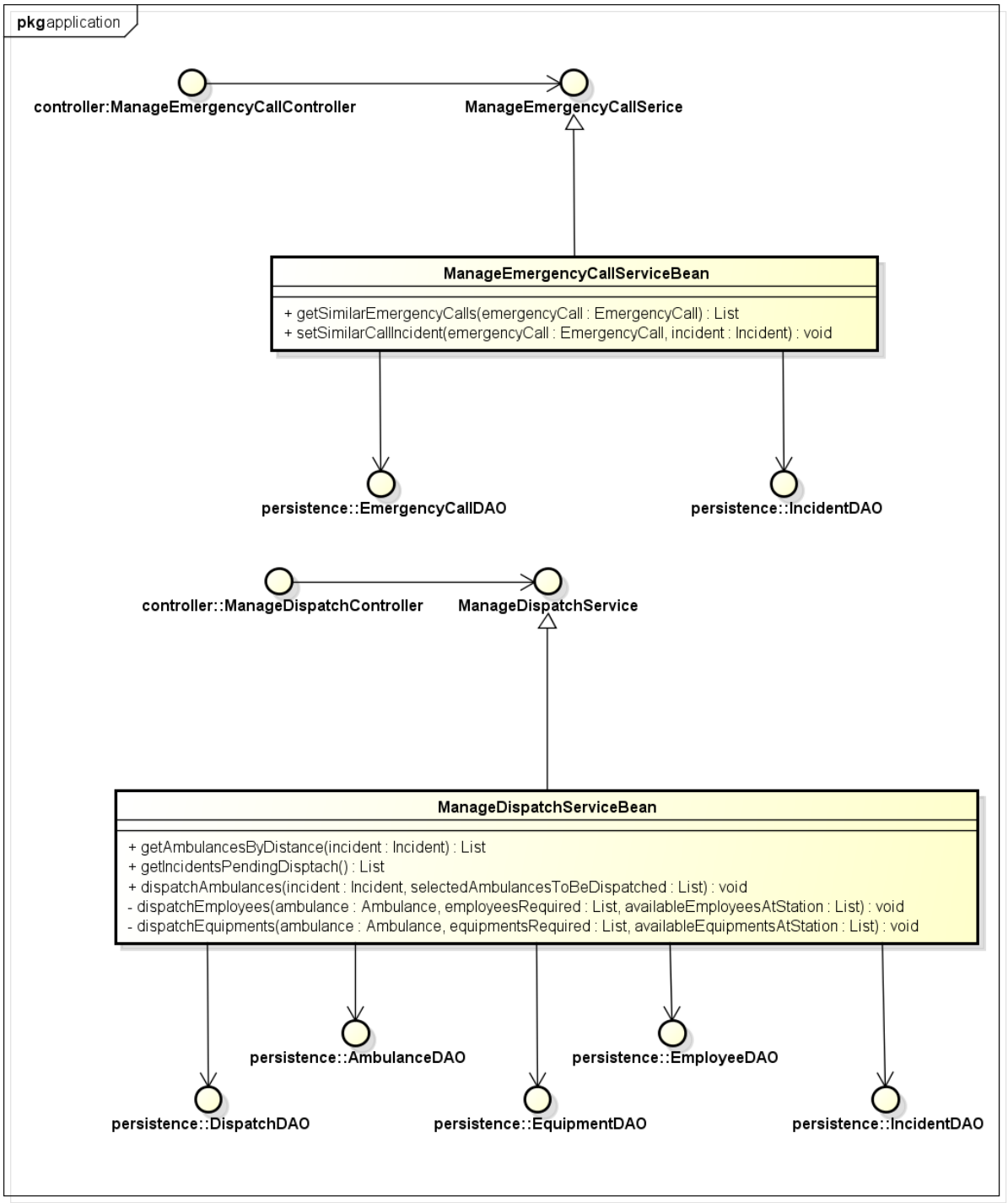

Figura 18 – Modelo de Aplicação para o módulo **Despacho**.

A Figura 18 contém o Modelo de Aplicação para os casos de uso **Registrar Chamada** e **Despachar Ambulância** cujo fluxo de execução se dá de forma similar ao apresentado na Figura 17. Pela figura, é possível observar a dependência da classe *ManageEmergencyCallServiceBean* com o elemento da camada de interface com o usuário, através da ligação com *controller::ManageEmergencyCallController*, e com os elementos da camada de acesso a dados, por meio de *persistence::EmergencyCallDAO* e *persis-*

tence::IncidenteDAO. Da mesma forma, é possível observar as dependências entre a classe de aplicação *ManageDispatchServiceBean* com as duas outras camadas. Tais dependências, assim como as representadas na Figura 17 são satisfeitas pelo CDI automaticamente.

4.3.4 Modelo de Persistência

Conforme apresentado na Seção 2.3, o Modelo de Persistência é um diagrama de classes UML que representa os *DAOs* existentes, responsáveis pela persistência das instâncias das classes de domínio. Esse diagrama guia a construção das classes *DAO*, que pertencem à Lógica de Acesso a Dados (SOUZA, 2007).

O Modelo de Persistência é composto por todas as classes do domínio mapeadas para um banco de dados através do mapeamento objeto/relacional feito pelo JPA. Seguindo o padrão DAO, essas classes são oferecidas, por meio de interfaces, para o nível superior (de aplicação) cujas operações possíveis são ilustradas no modelo.

Todas as classes de persistência do sistema herdam de *BaseJPADAO* do pacote *nemo-utils* que oferece vários métodos básicos de recuperação, exclusão e criação de novas informações no banco de dados. A Figura 19 representa o modelo básico de persistência da ferramenta.

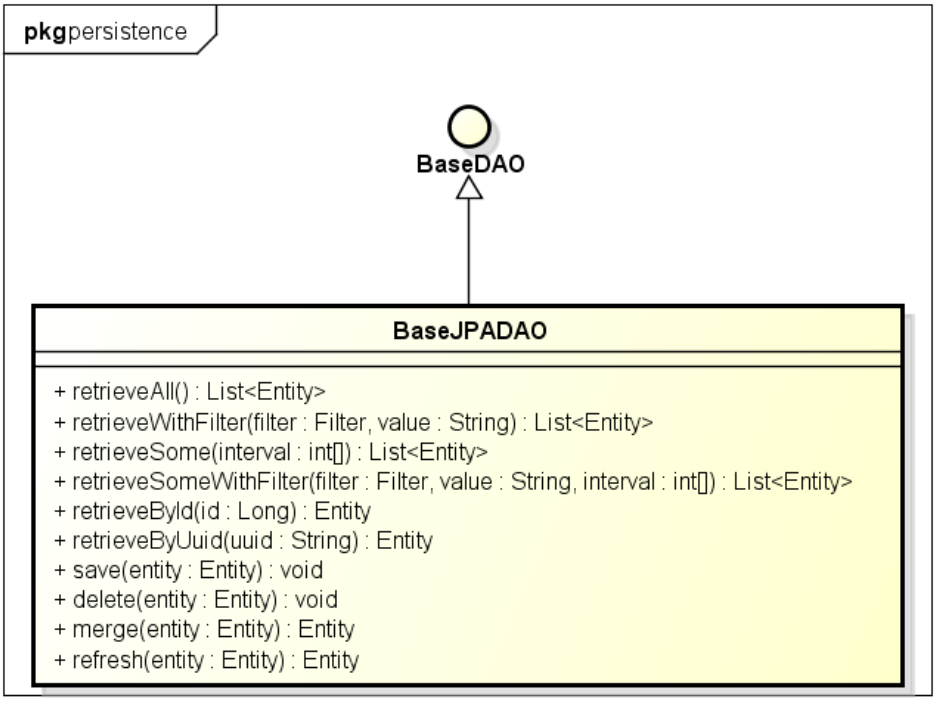

Figura 19 – Modelo de Persistencia para o módulo **Cadastro**.

As Figuras 20 e 21 contém, respectivamente, os Modelos de Persistência dos módulos **Cadastro** e **Despacho**. Todas as classes seguem o padrão da ferramenta *nemo-utils* e, com o intuito de tornar a visibilidade e o entendimento mais fácil, nem todas as classes

foram incluídas nos modelos e para fins de simplificação, a herança estabelecida entre as classes do Resgate e do *nemo-utils* não foram representadas no diagrama. O acesso ao banco de dados é feito através do *framework* JPA utilizando, mais especificamente a implementação Hibernate.

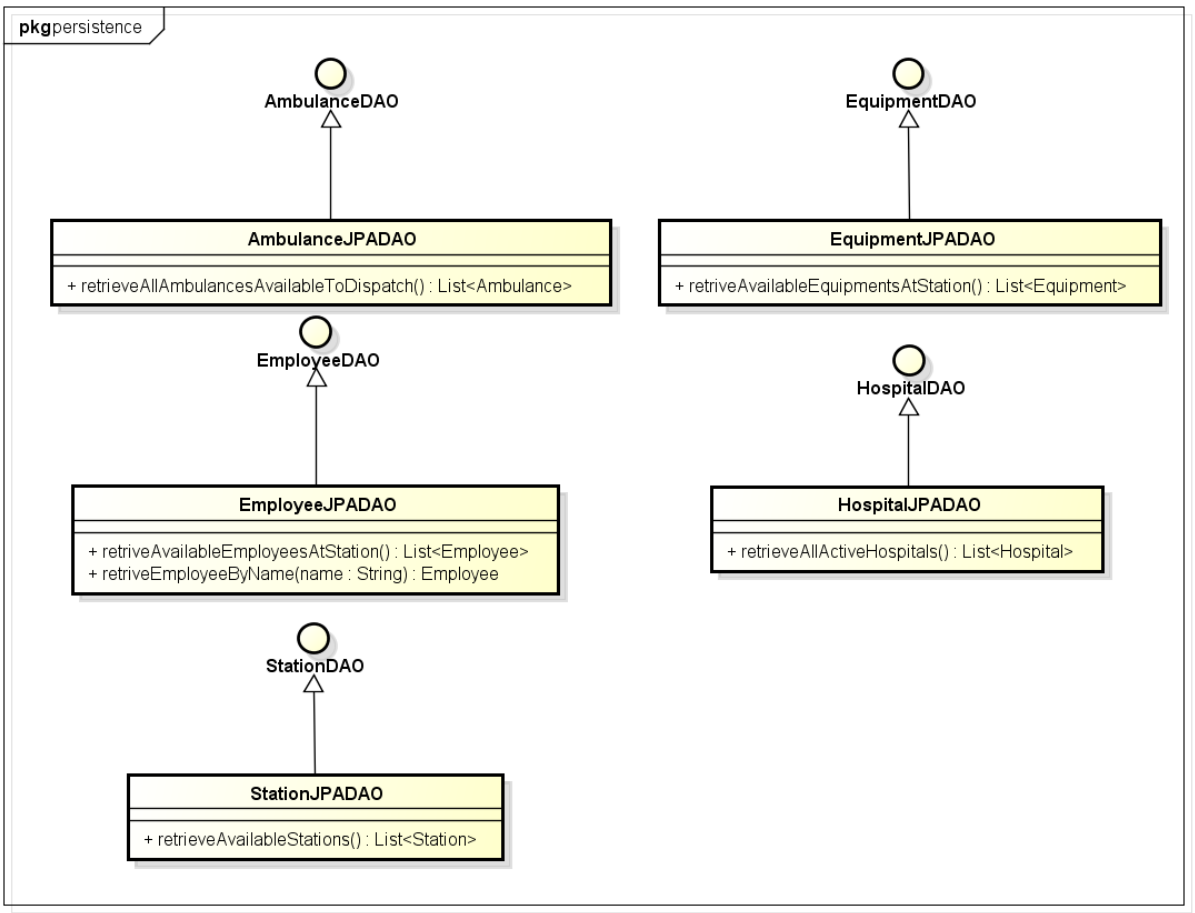

Figura 20 – Modelo de Persistencia para o módulo **Cadastro**.

Para o caso de uso **Cadastrar Ambulância** é solicitado ao usuário que indique a qual estação a ambulância que está sendo cadastrada irá pertencer. Para o utilizador ter como definir essa propriedade do objeto, o controlador (**ManageAmbulanceController**) solicita a aplicação (**ManageAmbulanceService**) que liste todas as estações possíveis, onde a aplicação, por sua vez, solicita a camada de persistência, através da classe (**StationDAO**), esses dados. Conforme mencionado nas Seções 4.3.2 e 4.3.3 todas as dependências entre as classes são satisfeitas automaticamente através do CDI. Pelo Modelo de Persistência apresentado na figura 20, é possível verificar a existência de um método na classe **StationJPADAO** que obtém todas as estações disponíveis no sistema e é através desse procedimento que a lista de estações é obtida. Na sequência, esses valores são retornados para a camada de aplicação e depois de navegação e por fim fica disponível para o usuário.

A Figura 21 concentra as classes do módulo **Despacho** e mostra quais são os

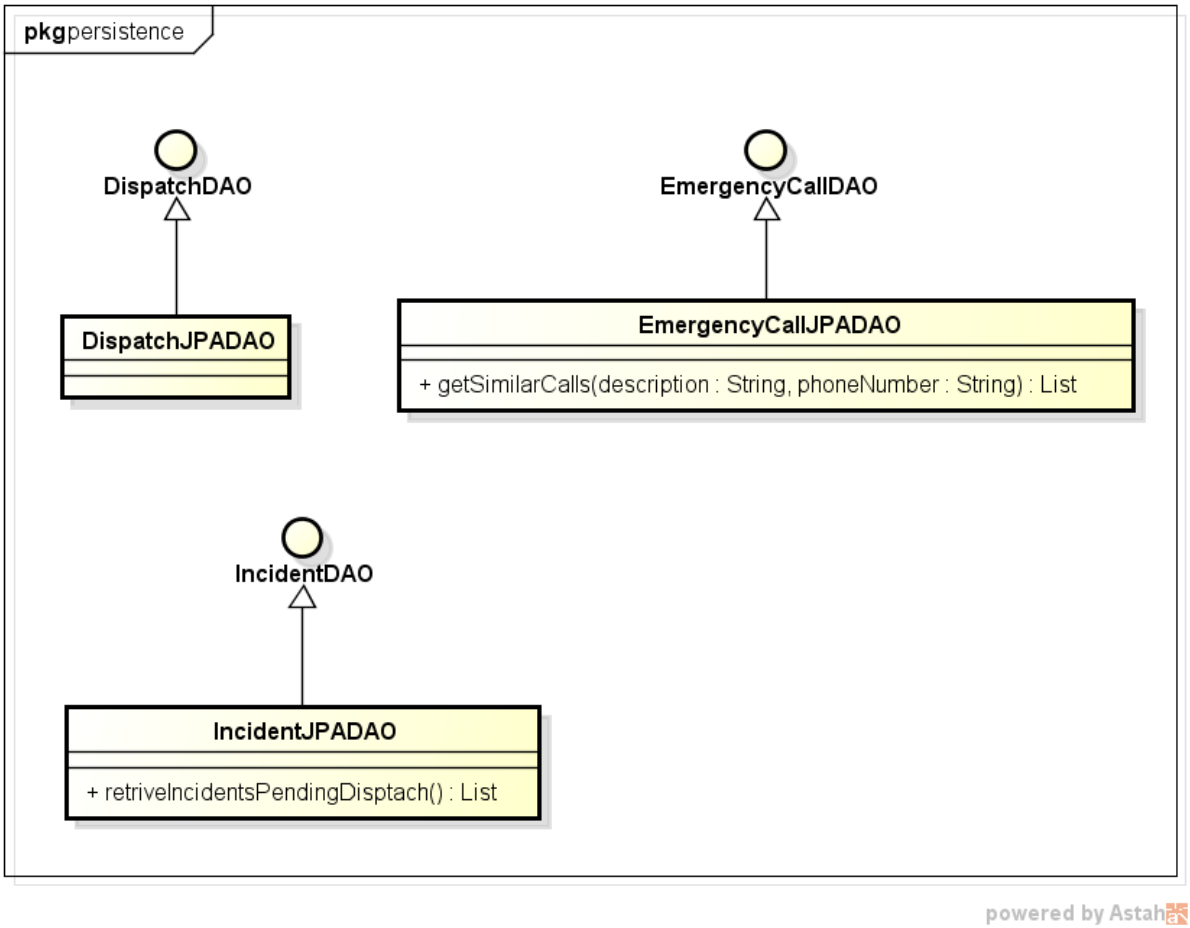

Figura 21 – Modelo de Persistencia para o módulo **Despacho**.

métodos disponíveis para acesso a dados de cada uma delas, além das que existem no *nemo-utils*. O fluxo de execução para este módulo, segue o mesmo principio que o apresentado na Figura 20, onde as classes de persistência realizam operações na base dados conforme é solicitado pelas classes de aplicação.

Uma das tarefas descritas no caso de uso **Registrar Chamada** é a de verificar chamadas similares no sistema. Isso ocorre quando o usuário solicita ao sistema que verifique tal informação, na tela de inclusão de uma nova chamada na camada de interface com o usuário. O sistema por sua vez, envia para a camada de aplicação a descrição e o número de telefone de origem e solicita que este faça essa busca. Na sequência, os dados são enviados para a camada de persistência através do método *getSimilarCalls*. Nesta última etapa, por meio de APIs de acesso oferecidas pelo Hibernate, é possível buscar na base dados por chamadas que contenham o mesmo número e mesma descrição no sistema e caso existam, elas são retornadas para a camada superior até chegar para o usuário. Conforme mencionado anteriormente, todas essas dependências entre classes de camadas diferentes são satisfeitas através da injeção de dependência provida pelo CDI.

A documentação técnica completa do Projeto do sistema encontra-se nos Apêndices

desta monografia.

4.4 Apresentação do Sistema

Depois da definição do escopo do projeto, dos requisitos funcionais e não funcionais na fase de Análise de Requisitos como mostrado no Capítulo 3, das tecnologias utilizadas, da Arquitetura de *Software* e dos modelos construídos como mostrado nas Seções 4.1, 4.2 e 4.3, a fase de implementação é responsável por juntar todas essas informações de forma consistente a fim de construir o sistema. Esta seção tem por objetivo mostrar o resultado final do projeto, indicando suas principais funcionalidades.

O primeiro contato com o sistema é através da tela de *Login*, que permite a autenticação no sistema para o **Atendente** poder utilizar a ferramenta, conforme mostra a Figura 22. Essa função é importante para manter um controle sobre quais **Atendentes** do órgão prestador de serviços de emergência foram responsáveis por criar os incidentes.

Figura 22 – Tela de *Login* do Resgate.

Enquanto não houver autenticação, nenhuma ação é possível de ser realizada no sistema, sendo que as funções de menu são liberadas de acordo com o controle de acesso aplicado ao utilizador do sistema.

No caso do **Adminstrador**, uma vez autenticado, ele tem permissão para gerenciar toda a parte cadastral do sistema. Ao clicar em qualquer um dos *links* de cadastro possíveis na tela, o **Adminstrador** é direcionado à tela de listagem da classe solicitada. Este tipo de tela se comporta de forma uniforme para todos os módulos do sistema e é representado na Figura 23.

Na tela de listagem é possível visualizar todas as entradas presentes no banco de dados para o tipo de cadastro solicitado. Abaixo da tabela com as informações, botões

Figura 23 – Tela de lisatgem do Resgate.

indicam quais são as possíveis ações na tela.

Ao solicitar a inclusão de um novo dado, a tela, como a apresentado na Figura 24, é mostrada. Todas as páginas de formulários do sistema seguem o mesmo padrão, onde apenas os dados disponíveis para preenchimento variam.

Figura 24 – Tela de formulário do Resgate.

Os campos que possuem o símbolo "*" são de preenchimento obrigatório indicando que o cadastro não poderá ser concluído enquanto todos não forem preenchidos. Ao salvar os dados inseridos ou ao cancelar o cadastro, o **Adminstrador** é encaminhado novamente para a tela de listagem.

Para os casos de uso que fogem do escopo de cadastros básicos a estrutura é um pouco diferente. Para as telas de **Incidente** e **Despacho**, por exemplo, não é possível nem criar um novo registro de forma direta e nem excluir os existentes através da interface independente do nível de acesso que o utilizador possui.

Esses dois elementos do sistema, são criados no momento em que uma nova chamada é incluída no sistema (caso de **Incidente**) e quando um novo despacho é realizado (caso de **Despacho**). A Figura 25 mostra a tela para inclusão de uma nova chamada no sistema.

A tela é composta de vários campos que contém informações relevantes a respeito

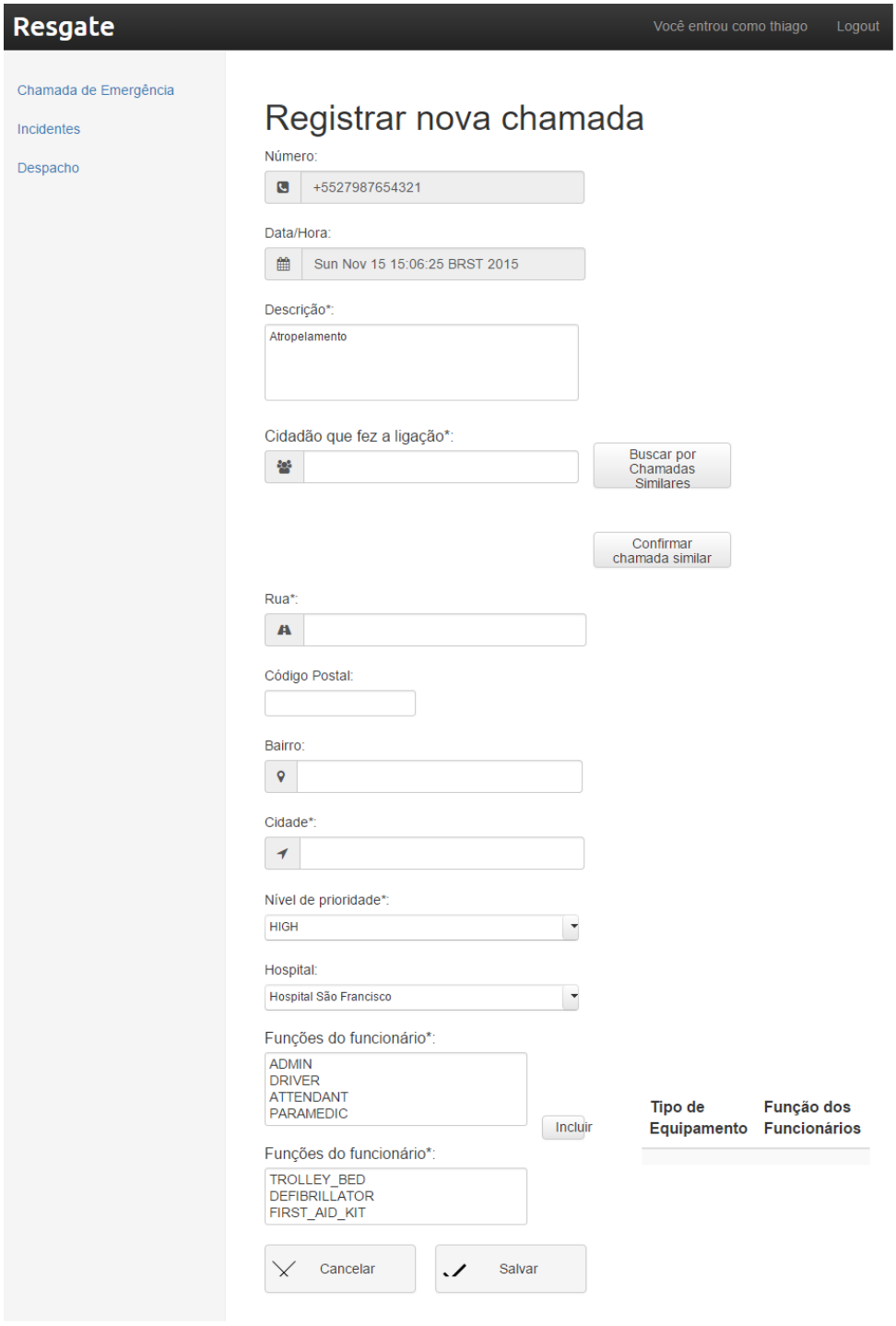

Figura 25 – Tela de **Registrar Chamada** do Resgate.

do contato feito com o cidadão que está requisitando o serviço de emergência oferecido. Inicialmente a data/hora da ligação já vem registrada no formulário e não é permitido alterá-la. Conforme descrito no caso de uso **Registrar Chamada** na Seção 3.3, após o preenchimento da descrição é possível buscar por chamadas similares no sistema e, caso sejam encontradas, elas serão inseridas na tabela de dados conforme mostrado.

Caso o **Atendente** indique que a chamada é similar a uma outra, o sistema executa os passos do caso de uso relacionado e em seguida carrega a tela de listagem de chamadas no sistema que segue o padrão mostrado na Figura 23. Caso contrário, o **Atendente** segue com o preenchimento dos dados e pode inserir quantas funções de funcionários juntamente com tipos de equipamentos (necessários para o atendimento ao incidente) quanto for necessário. Ao salvar os dados, novamente a tela de listagem é carregada.

A principal funcionalidade do Resgate é o despacho de ambulâncias e tal ação pode ser executa através do menu **Despacho** na tela do sistema. Ao acessar a função, o **Atendente** tem uma lista de todos os incidentes abertos e que ainda não tiveram ambulâncias despachadas, conforme mostrado na Figura 26.

Figura 26 – Tela de listagem de incidentes pendentes de despacho do Resgate.

Ao selecionar um incidente e solicitar o despacho de ambulâncias, o sistema mostra a tela indicada na Figura 27.

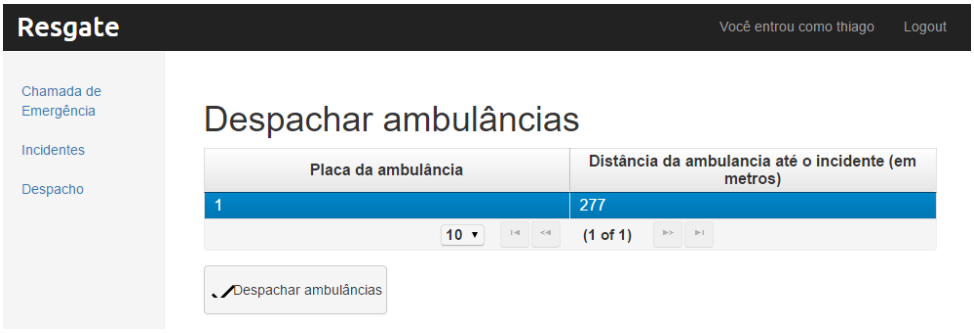

Figura 27 – Tela de ambulâncias disponíveis para despacho do Resgate.

Nesta tela o **Atendente** tem uma lista de ambulâncias disponíveis para despacho ordenada de forma crescente em relação a distância da ambulância até o local do incidente. Para solicitar o despacho, basta ao **Atendente** indicar para o sistema quais as ambulâncias devem ser despachadas e em seguida, a tela de listagem de incidentes pendentes de despacho é novamente aberta.

5 Considerações Finais

Neste capítulo, são apresentadas as conclusões tiradas do projeto além de sugestões para o *FrameWeb* e propostas para trabalhos futuros.

5.1 Conclusões

Dado que o desenvolvimento de aplicações voltadas para *web* está em constante expansão, várias ferramentas e *frameworks* vieram a fim de facilitar a vida de desenvolvedores, sendo um exemplo desses o *FrameWeb*, proposto por (SOUZA, 2007), que foi utilizado nesse trabalho.

O *FrameWeb* propõe a utilização de um conjunto variado de *frameworks* para as diferentes camadas de um *software*. O método também apresenta uma nova forma de documentar o projeto do sistema através de Modelos de Apresentação, Domínio, Aplicação e Persistência que são diagramas de classe UML. Esse modelos se diferem dos já existentes, por proverem informações mais detalhadas devido ao fato de serem voltados para o desenvolvimento *web*.

Para a construção da ferramenta, toda a documentação indicada pela Engenharia de Software foi feita. Inicialmente foi montado o Documento de Requisitos, contendo os requisitos funcionais e não funcionais além da descrição do escopo do problema. Na sequência, o Documento de Especificação de Requisito contendo a definição dos atores, casos de uso e diagrama de classe. Por fim, o Documento de Projeto contendo os modelos proposto pelo *FrameWeb* além da arquitetura de *software*.

Após a conclusão da documentação do *software* um sistema foi desenvolvido utilizando a linguagem Java™ EE 7 onde todos os casos de uso tiveram seus fluxos normais de execução implementados, conforme mostrado neste projeto. O sistema tem capacidade de ser escalado para anteder um conjunto maior de cidades, estados ou países assim como pode ser traduzido para qualquer idioma.

Ao final de todas as etapas, ficou evidente o quão difícil é ter um *software* documentado de forma integral. Dependendo do tipo da alteração, por menor que seja, é necessário atualizar vários documentos a fim de manter a consistência. Porém, ficou claro a importância de ter feito tal controle para a manutenção e expansão do sistema.

Este trabalho também, tinha como objetivo contribuir com a pesquisa realizada em (SOUZA; MYLOPOULOS, 2013) buscando a realização de testes em uma plataforma não adaptativa real. Com o desenvolvimento do Resgate, os autores agora possuem mais um meio de para validarem os dados simulados durante o experimento.

5.2 Considerações sobre o método *FrameWeb*

A utilização do método *FrameWeb* para o desenvolvimento do sistema foi uma escolha considerada como bem produtiva e positiva. Com o auxílio do método, os diagramas da fase de Projeto de Sistemas contém informações mais precisas e detalhadas. Os diversos *frameworks* e a arquitetura de *software* sugerida, fazem com que o sistema apresente um baixo acoplamento entre suas camadas.

Entretanto, algumas dificuldades para construção dos modelos foram encontradas ao longo do caminho. A maior parte foi identificada no Modelo de Navegação, que por vezes se mostra confuso e difícil de ser construído por querer representar várias informações em um único diagrama como anotações a respeito de métodos, eventos e elementos a serem renderizados.

De forma geral, é possível listar as dificuldades encontradas utilizando o *FrameWeb*:

- A sua não adequação total para a plataforma de desenvolvimento escolhida para este trabalho, Java™ EE 7;
- A *tag {not null}* não ser padrão para os atributos no Modelo de Domínio;
- Não ter suporte ao AJAX;
- Não oferecer *tags* que permitam configurar os componentes de bibliotecas utilizadas (PrimeFaces, BootsFaces);
- Não ter suporte direto a bibliotecas de componentes;
- A sua grande dependência de específicos *frameworks*, como o Struts 2;

Devido a estes fatos, algumas modificações tiveram de ser aplicadas afim de tornar o *framework* adequado para as tecnologias aqui utilizadas, onde algumas dessas modificações vieram de novas propostas que estão sendo elaboradas em (MARTINS; SOUZA, 2015).

5.3 Trabalhos Futuros

Várias melhorias podem ser aplicadas ao Resgate afim de tornar este um *software* mais completo, confiável e seguro. Dentre as possíveis, vale citar:

- Evolução do sistema de despacho de ambulância através da implementação de suporte a fluxos alternativos como, por exemplo, o tratamento de erros em tempo de execução;
- Transformar o sistema em adaptativo conforme proposto por (SOUZA; MYLOPOULOS, 2013);

- Inclusão de relatórios de tempo total gasto para atendimento a cidadãos, tipos de incidentes mais comuns, ligações recebidas no Órgão de controle de situações que retratam falsos incidentes (trotes), entre outras melhorias;

Referências

ALUR, D. et al. *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. [S.l.]: Sun Microsystems, Inc., 2003. Citado na página 25.

BAUER, C.; KING, G. Hibernate in action. Manning Greenwich, 2005. Citado na página 40.

DEMICHIEL, L.; KEITH, M. Java persistence api. *JSR*, v. 220, 2006. Citado na página 40.

DEMICHIEL, L.; SHANNON, B. Java platform, enterprise edition 7 (java ee 7) specification. *JSR, JCP*, 2013. Citado na página 18.

FALBO, R. A. *Projeto de Sistemas: Notas de aula*. 2011. Disponível em: <http://www.inf.ufes.br/~falbo/files/Notas_Aula_Projeto_Sistemas_2.pdf>. Acesso em: 13 set. 2015. Citado 3 vezes nas páginas 22, 23 e 39.

FALBO, R. A. *Engenharia de Requisitos: Notas de aula*. 2012. Disponível em: <http://www.inf.ufes.br/~falbo/files/Notas_Aula_Engenharia_Requisitos.pdf>. Acesso em: 13 set. 2015. Citado 3 vezes nas páginas 21, 22 e 35.

FINKELSTEIN, A. *Report of the inquiry into the London ambulance service (electronic version)*. [S.l.], 1993. Citado 2 vezes nas páginas 7 e 17.

FOWLER, M. *Patterns of enterprise application architecture*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2002. Citado 3 vezes nas páginas 9, 23 e 24.

GARRETT, J. J. et al. Ajax: A new approach to web applications. 2005. Citado na página 47.

JENDROCK, E. et al. *The Java EE 7 Tutorial*. [S.l.]: Addison-Wesley Professional, 2014. v. 1. Citado na página 39.

MANN, K. D. *Java Server Faces in Action*. [S.l.]: Dreamtech Press, 2005. Citado na página 39.

MARTINS, B. F.; SOUZA, V. E. S. A model-driven approach for the design of web information systems based on frameworks. In: ACM. *Proceedings of the 21st Brazilian Symposium on Multimedia and the Web*. [S.l.], 2015. p. 41–48. Citado 2 vezes nas páginas 47 e 60.

PFLEEGER, S. L. Engenharia de software: teoria e prática. *2ª Edição, Prentice Hall*, 2004. Citado 3 vezes nas páginas 21, 22 e 39.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. [S.l.]: Palgrave Macmillan, 2005. Citado na página 22.

ROBERTSON, S.; ROBERTSON, J. *Mastering the requirements process*. 2. ed. [S.l.]: Addison Wesley, 2006. Citado na página 21.

SOMMERVILLE, I.; KOTONYA, G. *Requirements engineering: processes and techniques*. [S.l.]: John Wiley & Sons, Inc., 1998. Citado na página 21.

SOMMERVILLE, I. et al. *Engenharia de software*. [S.l.]: Addison Wesley São Paulo, 2003. v. 6. Citado 2 vezes nas páginas 21 e 22.

SOUZA, V. E. S. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. Tese (Doutorado) — Dissertação (Programa de Pós-Graduação em Informática)—Universidade Federal do Espírito Santo, 2007s, 2007. Citado 12 vezes nas páginas 7, 18, 22, 23, 24, 25, 41, 42, 44, 48, 51 e 59.

SOUZA, V. E. S.; MYLOPOULOS, J. Requirements-based software system adaptation. *PhD diss, PhD Thesis, University of Trento, Italy*, 2012. Citado na página 17.

SOUZA, V. E. S.; MYLOPOULOS, J. Designing an adaptive computer-aided ambulance dispatch system with Zanshin: an experience report. *Software: Practice and Experience (online first: http://dx.doi.org/10.1002/spe.2245)*, Wiley, 2013. Citado 6 vezes nas páginas 7, 17, 18, 27, 59 e 60.

Apêndices

Documento de Requisitos

Projeto: **Resgate** – Despacho de ambulâncias automatizado.

Registro de Alterações:

Versão	Responsável	Data	Alterações
1.0	Thiago Martinho da Costa	12/03/2015	Inclusão das seções 1, 2, 3 e 4.

1. Introdução

Este documento apresenta os requisitos de usuário da ferramenta *Resgate* – Despacho de ambulâncias automatizado e está organizado da seguinte forma: a seção 2 contém uma descrição do propósito do sistema; a seção 3 contém uma descrição do minimundo apresentando o problema; e a seção 4 apresenta a lista de requisitos de usuário levantados junto ao cliente.

2. Descrição do Propósito do Sistema

A ferramenta Resgate tem como objetivo fornecer uma forma de controlar o despacho de ambulâncias em situações de urgência e emergência, levando em consideração os recursos físicos e humanos disponíveis no momento do incidente.

3. Descrição do Minimundo

O serviço de emergência é um serviço provido pelas autoridades de uma região específica para cidadãos que consiste no despacho de ambulâncias, seus funcionários e equipamentos para determinados incidentes.

Um cidadão faz uma ligação para a central de emergências provida pelas autoridades. De uma chamada deseja-se saber: número, hora de início, descrição e nome do cidadão. A partir de uma chamada pode ser incluído no sistema um incidente. Um incidente é algum tipo de acidente ou emergência em que seja necessário a assistência de uma ou mais ambulâncias, por exemplo, um acidente de carro que tem pessoas feridas. Não existe uma predefinição do

que é ou o que não é um incidente, portanto todas as chamadas recebidas são analisadas por funcionários do serviço de emergência, podendo ser considerada como não emergencial. De um incidente deseja-se saber: endereço, nível de prioridade, configurações necessárias de ambulâncias e o estado do incidente. Além disso, o incidente deve ser associado a uma ou mais chamadas (chamadas múltiplas retratando o mesmo incidente devem ser também identificadas pelos atendentes) e o hospital para onde serão levadas as vítimas. Hospitais são centros médicos que possuem alas de emergência capaz de receber pessoas trazidas por ambulâncias. De um hospital deseja-se saber: endereço e nome.

Ambulâncias e funcionários ficam à espera de instruções para despacho em estações. De uma estação deseja-se saber: endereço, ambulâncias, nome, funcionários, equipamentos e estado da estação. Ambulância é qualquer veículo usado para serviço de emergência que auxilia cidadãos em caso de incidentes. De uma ambulância deseja-se saber: placa identificadora e o estado da ambulância. Equipamento é qualquer peça que seja útil para o atendimento a um incidente e é composto por um tipo e um identificador único.

Funcionários são recursos humanos que trabalham em ambulâncias ou na central de emergência e proveem ajuda a cidadãos em caso de incidentes. De um funcionário deseja-se saber: nome e função (atendente, motorista, paramédico ou administrador).

Uma ambulância só pode ser despachada para um incidente se ela atender os requisitos necessários para o mesmo, ou seja, a configuração da ambulância tem que ser a mesma configuração requerida pelo incidente. Ela se refere ao tipo de veículo, o papel dos funcionários e os equipamentos presentes na ambulância. Quando o funcionário autoriza o despacho da ambulância, a data e hora de ocorrência do evento são armazenados. Ao finalizar o atendimento a um incidente, a ambulância deve retornar à sua estação deixando os funcionários, equipamentos e a própria ambulância disponíveis para novos despachos.

4. Requisitos de Usuário

Tomando por base o contexto do sistema, foram identificados os seguintes requisitos de usuário:

Requisitos Funcionais

Identificador	Descrição	Prioridade	Depende de
RF01	O sistema deve permitir ao usuário registrar chamadas recebidas por cidadãos.	Alta	RN02, RN03
RF02	O sistema deve, sempre que possível, detectar a localização de origem da chamada e associar ao registro de chamadas.	Alta	RF01
RF03	O sistema deve permitir ao usuário rejeitar chamadas não emergenciais.	Alta	RN03
RF04	O sistema deve auxiliar o usuário a identificar, através das informações obtidas da ligação, se o chamado se refere a um incidente já aberto no sistema.	Alta	RF01, RN04
RF05	O sistema deve permitir ao usuário associar chamadas duplicadas aos incidentes abertos no sistema ou criar novos incidentes.	Alta	RF01, RF04
RF06	O sistema deve permitir ao usuário indicar o número de ambulâncias necessárias e as suas respectivas configurações (i.e., ambulância com paramédicos, e caminhões de bombeiro com bombeiros).	Alta	
RF07	O sistema deve permitir ao usuário confirmar informações a respeito de novos incidentes, liberando o despacho de ambulâncias pelo sistema.	Alta	
RF08	O sistema deve, através da confirmação do incidente, determinar a melhor ambulância para ser enviada ao local do incidente, dadas as configurações requeridas e permitir ao funcionário confirmar as informações.	Alta	RF06, RF07, RN07
RF09	O sistema deve informar às estações de despacho de ambulâncias novos incidentes com as configurações de despacho, se a ambulância estiver na estação, ou informar para a própria ambulância caso contrário.	Alta	RF08, RN01, RN05, RN06
RF10	O sistema deve fechar incidentes quando todos os recursos alocados forem liberados.	Alta	RF07

Identificador	Descrição	Prioridade	Depende de
RF11	O sistema deve, no caso de uma desativação de uma ambulância em uso, determinar a melhor ambulância a ser despachada no lugar da que foi desativada, dadas as configurações necessárias.	Alta	RF08, RF09, RN05
RF12	O sistema deve monitorar eventos relacionados a ambulâncias e manter os status de cada ambulância atualizados, incluindo as configurações das ambulâncias.	Alta	RF01, RF12
RF13	O sistema deve gerar mensagens sempre que ambulâncias chegarem nos locais de incidentes, deixando os locais de incidentes (parar ir ao hospital), e quando elas forem liberadas (incidente resolvido).	Média	
RF14	O sistema deve permitir ao funcionário gerenciar incidentes.	Alta	
RF15	O sistema deve permitir ao administrador cadastrar estações.	Alta	
RF16	O sistema deve permitir ao administrador cadastrar hospitais.	Alta	
RF17	O sistema deve permitir ao administrador cadastrar ambulâncias.	Alta	
RF18	O sistema deve permitir ao administrador cadastrar equipamentos.	Alta	
RF19	O sistema deve permitir ao administrador cadastrar funcionários.	Alta	

Regras de Negócio

Identificador	Descrição	Prioridade	Depende de
RN01	Uma ambulância não pode ser despachada para incidentes sem a configuração e equipamentos corretos.	Alta	
RN02	Para todas as chamadas feitas por cidadãos devem ser registrados o número e a hora de início da chamada.	Alta	

Identificador	Descrição	Prioridade	Depende de
RN03	Funcionários recebem ligações feitas por cidadãos e avaliam se trata-se de uma emergência ou não.	Alta	
RN04	Incidentes são associados com uma ou mais chamadas feitas por cidadãos.	Alta	
RN05	Uma ambulância não pode ser despachada para um incidente enquanto estiver em atendimento.	Alta	
RN06	Funcionários podem despachar ambulâncias que estejam disponíveis tanto nas estações quanto fora delas.	Média	
RN07	Apenas funcionários livres podem ser lotados em ambulâncias para despacho.	Alta	

Requisitos Não Funcionais

Identificador	Descrição	Categoria	Escopo	Prioridade	Depende de
RNF01	A ferramenta deve possuir um controle de nível de usuário, diferenciando atendentes e administradores.	Segurança	Sistema	Alta	
RNF02	A ferramenta deve reutilizar componentes e *frameworks* existentes.	Reusabilidade	Sistema	Média	
RNF03	A ferramenta deve ser capaz de tratar qualquer possível erro de execução e informar ao usuário.	Confiabilidade	Sistema	Alta	

Identificador	Descrição	Categoria	Escopo	Prioridade	Depende de
RNF04	O sistema deve ser preciso e atualizado com relação às informações de incidentes abertos, incluindo seus status, configurações e posições das ambulâncias alocadas.	Confiabilidade	Sistema	Alta	

Documento de Especificação de Requisitos

Projeto: **Resgate** – Despacho de ambulâncias automatizado

Registro de Alterações:

Versão	Responsáveis	Data	Alterações
1.0	Thiago Martinho da Costa	15/04/2015	Inclusão da seção 1
1.1	Thiago Martinho da Costa	02/05/2015	Inclusão das seções 2 e 3
1.2	Thiago Martinho da Costa	26/10/2015	Inclusão das seções 4, 5 e 6

1. Introdução

Este documento apresenta a especificação dos requisitos da ferramenta Resgate. 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 da seguinte forma: a seção 2 apresenta os subsistemas identificados, mostrando suas dependências na forma de um diagrama de pacotes; a seção 3 apresenta o modelo de casos de uso, incluindo descrições de atores, os diagramas de casos de uso e descrições de casos de uso; a seção 4 apresenta o modelo conceitual estrutural do sistema, na forma de diagramas de classes; a seção 5 apresenta o modelo comportamental dinâmico do sistema, na forma de diagramas de estado; finalmente, a seção 6 apresenta o glossário do projeto, contendo as definições das classes identificadas.

2. Identificação de Subsistemas

A Figura 1 mostra os subsistemas identificados no contexto do presente

projeto, os quais são descritos na tabela abaixo.

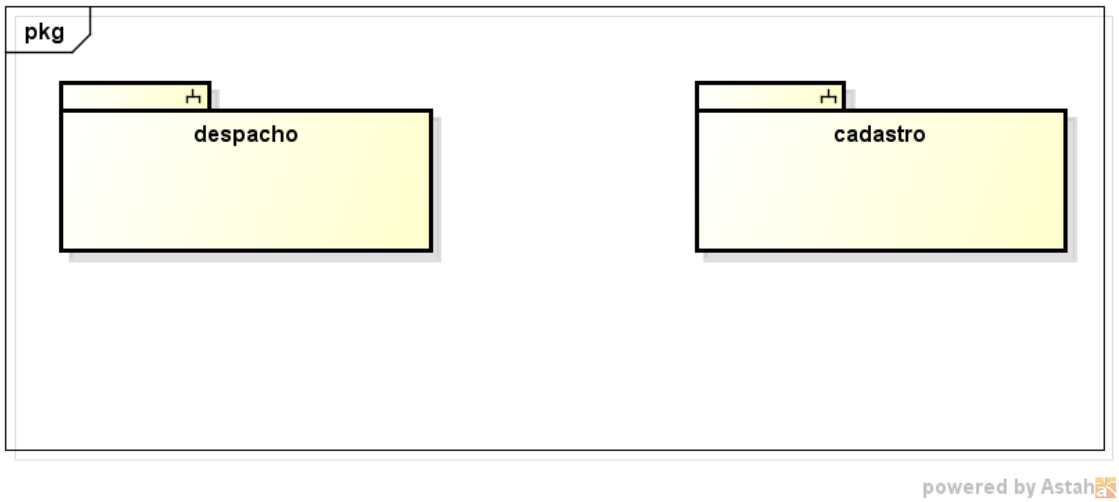

Figura 1 – Diagrama de Pacotes e os Subsistemas Identificados.

Tabela 1 – Subsistemas

Subsistema	Descrição
cadastro	Envolve toda a funcionalidade relacionada a cadastros do sistema. Engloba o cadastro de estações, hospitais, ambulâncias, equipamentos, funcionários e configuração de ambulâncias.
despacho	Envolve a funcionalidade relacionada despacho de ambulâncias, gerenciamento de incidentes e de chamadas.

3. 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 abaixo.

Tabela 2 – Atores.

Ator	Descrição
Administrador	Funcionário do órgão que presta serviço de emergência e responsável pela manutenção dos cadastros do sistema.
Funcionário	Funcionário do órgão que presta serviço de emergência responsável pelo atendimento ao cidadão e despacho de ambulância ou atendimento ao incidente.
MDT	Sistema presente nas ambulâncias responsável por receber as instruções de despacho e informar o estado atualizado do despacho.
Google Maps	Sistema de mapa que oferece o cálculo de distância entre posições geográficas.

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

3.1. Subsistema *cadastro*

A Figura 2 apresenta o diagrama de casos de uso do subsistema *cadastro*.

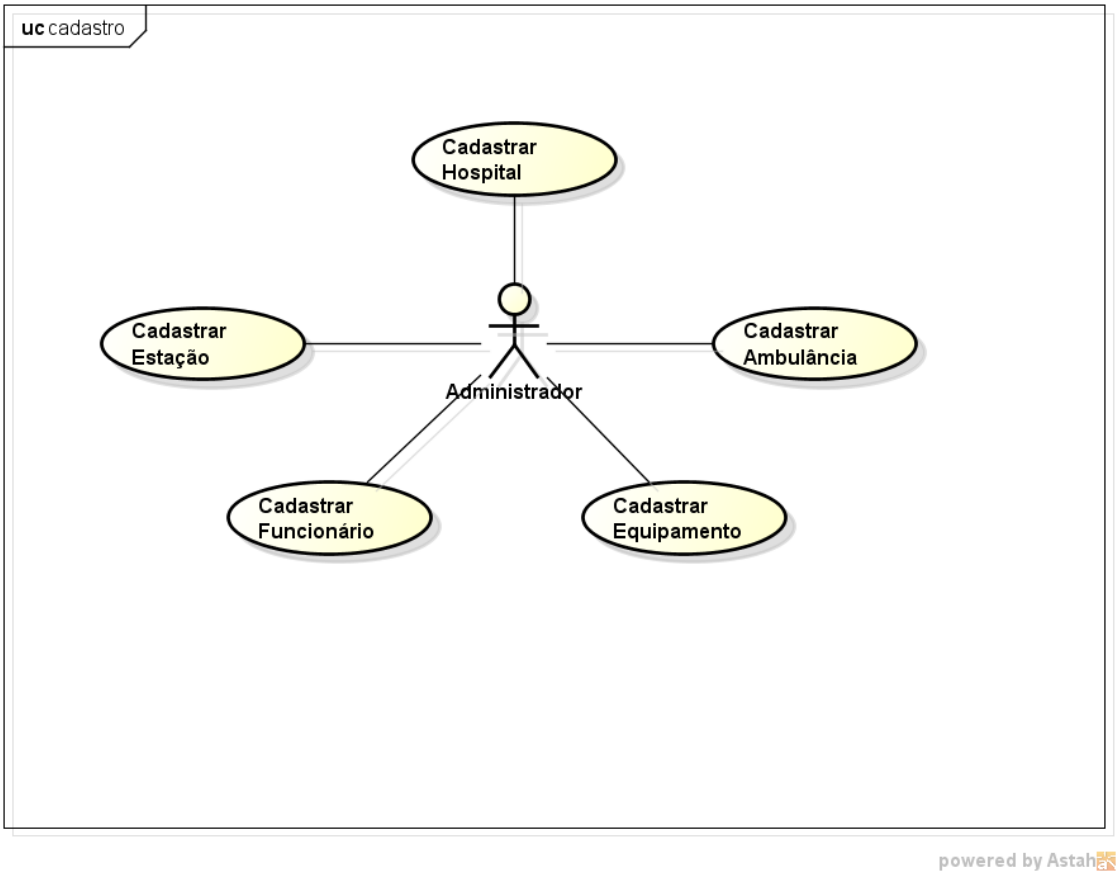

Figura 2 – Diagrama de Casos de Uso do Subsistema *despacho*.

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 abaixo, segundo o padrão da organização.

Tabela 3 – **Casos de Uso Cadastrais**

Subsistema	**cadastro**				
Identificador	**Caso de Uso**	**Ações Possíveis**	**Observações**	**Requisitos**	**Classes**
UC1	**Cadastrar Hospital**	I, A, C, E	[I] informar: endereço e nome. [A] não é permitido inativar um hospital associado a um incidente aberto. [E] não é permitido excluir um hospital associado a um incidente aberto.	RF16	Hospital
UC2	**Cadastrar**	I, A, C, E	[I] informar: endereço e nome.	RF15	Station

	Estação		[A] não é permitida a inativar estações que possuam ambulâncias associadas. [E] não é permitida a exclusão de estações que possuam ambulâncias associadas.		
UC3	**Cadastrar Equipamento**	I, A, C, E	[I] informar: tipo, identificador único e estação. [E] não é permitida a exclusão de equipamentos que possuam ambulâncias associadas.	RF18	Equipment
UC4	**Cadastrar Funcionário**	I, A, C, E	[I] informar: nome, função e estação. [E] não é permitida a exclusão de funcionários que estejam lotados em ambulâncias.	RF19	Employee
UC5	**Cadastrar Ambulância**	I, A, C, E	[I] informar: placa identificadora e estação. [E] não é permitida a exclusão de ambulâncias em atendimento. [E] ao excluir uma ambulância deve-se desassociar os funcionários e equipamentos lotados na ambulância.	RF17	Ambulance, Station, Employee, Equipment

3.2. Subsistema despacho

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

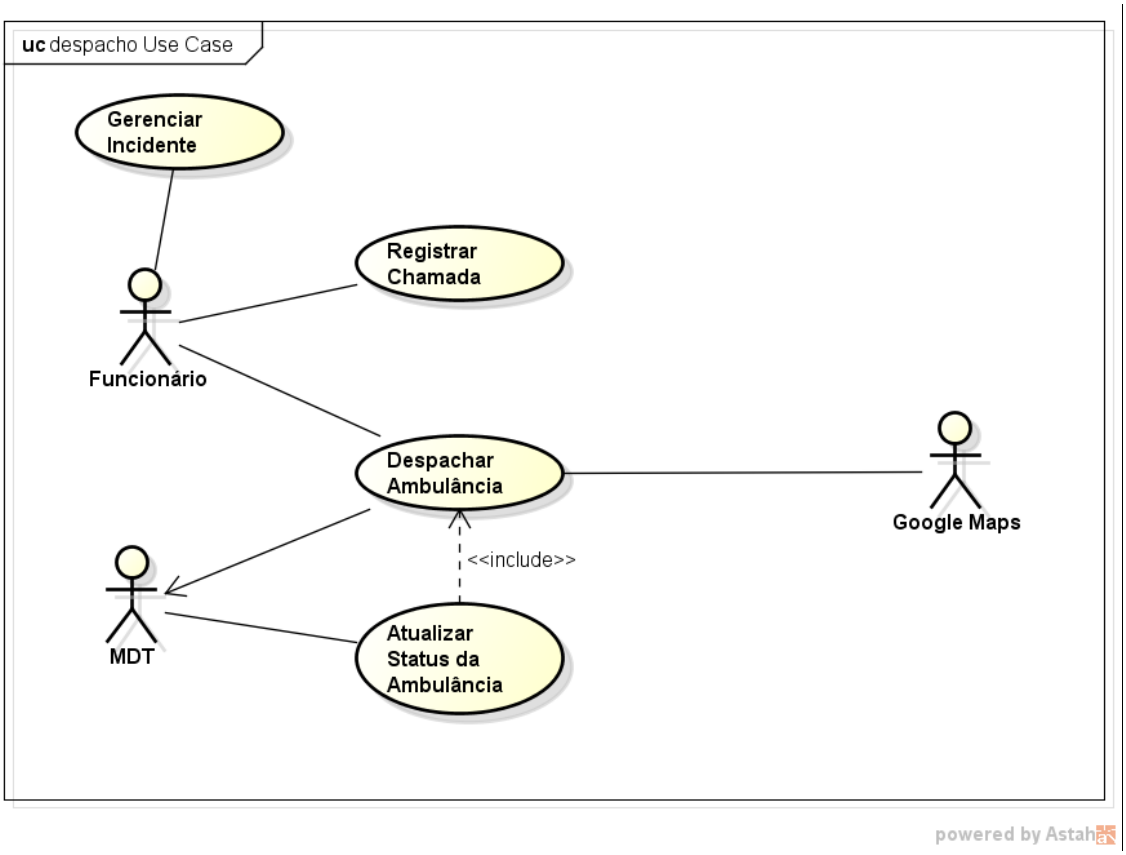

Figura 3 – Diagrama de Casos de Uso do Subsistema *despacho*.

A seguir, são apresentadas as descrições de cada um dos casos de uso identificados.

Descrição de Caso de Uso

Identificador do Caso de Uso: Registrar Chamada
Caso de Uso: UC6

Descrição Sucinta: Este caso de uso é responsável pelo registro de chamadas e inclusão de incidentes no sistema.

Fluxos de Eventos Normais

Nome do Fluxo de Eventos Normal	Precondição	Descrição
Incluir Nova Chamada		1 – O funcionário seleciona a opção de incluir nova chamada no sistema; 2 – O sistema exibe uma tela de inclusão dos dados para o funcionário e insere a data e hora em que a ligação foi recebida. 3 - O funcionário informa a descrição da chamada, número de telefone de origem e nome do cidadão. 4 – O funcionário solicita que o sistema faça busca por chamadas similares levando em consideração a descrição da chamada, número de origem da chamada ou nome do cidadão. 5 – O sistema exibe o resultado da busca com as possíveis chamadas similares. 6 – O funcionário indica ao sistema que as chamadas encontradas não são similares à nova chamada. 7 - O funcionário informa: endereço do incidente, o nível de prioridade, configurações de ambulâncias necessárias (tipo de funcionários de equipamentos requeridos), e o hospital para onde as vítimas serão levadas. 8 – O funcionário confirma os dados inseridos. 9 - A nova chamada é registrada no sistema. 10 - Um novo incidente contendo os dados obtidos através da ligação, a saber: endereço do incidente, o nível de prioridade, configurações de ambulâncias necessárias e o hospital, é criado e a chamada é associada ao novo incidente pelo sistema. 11 – O sistema libera o despacho de ambulâncias para o incidente.
Consultar Chamada		1 - O funcionário informa a chamada que deseja consultar; 2 - Os dados são exibidos para o funcionário.

Fluxos de Eventos Variantes

Nome do Fluxo de Eventos Normal Relacionado	Variante	Descrição
Incluir Nova Chamada	5 – O sistema não encontra nenhuma chamada similar.	5 – O sistema informa ao usuário através de uma mensagem que não foi possível localizar nenhuma chamada similar e o funcionário avança para o passo 7.
Incluir Nova Chamada	6 – O funcionário verifica que as dentre	6 – O funcionário solicita que o sistema recupere o incidente que esteja associado a chamada similar

	as chamadas encontradas pelo sistema, existem similares.	encontrada. 7 – O sistema exibe o incidente para o funcionário. 8 – O funcionário associa a nova chamada ao incidente e solicita ao sistema que o salve. 9 – O sistema salva o incidente.
Incluir Nova Chamada	8 – O funcionário não confirma os dados inseridos.	8 – O sistema retorna ao passo 3, com os campos já preenchidos e permite ao funcionário editá-los.

Requisitos Relacionados: RF01, RF02, RF03, RF04 e RF05.
Classes Relacionadas: EmergencyCall e Incident.

Descrição de Caso de Uso

Identificador do Caso de Uso: Gerenciar Incidente
Caso de Uso: UC7

Descrição Sucinta: Este caso de uso é responsável pela alteração e consulta de incidentes.

Fluxos de Eventos Normais

Nome do Fluxo de Eventos Normal	Precondição	Descrição
Alterar Incidente		1 - O funcionário informa o incidente que deseja alterar; 2 – O sistema exibe o incidente solicitado pelo funcionário permitindo a alteração apenas do estado do incidente (vide Figura 7, Seção 5 para descrição dos estados possíveis), nível de prioridade e hospital para onde serão levadas as vítimas. 3 – O funcionário altera os campos que desejar e solicita ao sistema que salve as alterações. 4 – O sistema então salva o incidente com os novos dados inseridos pelo funcionário.
Consultar Dados de Incidente		1 - O funcionário informa o incidente que deseja consultar; 2 - Os dados do incidente são apresentados.

Fluxos de Eventos de Exceção

Nome do Fluxo de Eventos Normal Relacionado	Condição de Exceção	Descrição
Alterar Dados de Incidente	3 – Dados inválidos	3 – Uma mensagem de erro é exibida, e o sistema retorna para a mesma tela permitindo ao funcionário editar os dados novamente.

Requisitos Relacionados: RF06, RF07 e RF14.
Classes Relacionadas: Incident, EmergencyCall e Hospital.

Descrição de Caso de Uso

Identificador do Caso de Uso: Despachar Ambulância
Caso de Uso: UC8

Descrição Sucinta: Este caso de uso é responsável pelo despacho da(s) ambulância(s).

Fluxos de Eventos Normais

Nome do Fluxo de Eventos Normal	Precondição	Descrição
Despachar Ambulância		1 – O funcionário solicita ao sistema que liste todos os incidentes pendentes de despacho. 2 – O sistema exibe todos os incidentes que estão abertos e não foram despachados. 3 – O funcionário seleciona um incidente e solicita ao sistema que verifique quais são as melhores ambulâncias para serem despachadas. 4 – O sistema envia para o Google Maps as posições das ambulâncias livres e do incidente e solicita que seja calculado a distância entre cada ambulância e o local do incidente. 5 – O Google Maps informa ao sistema a distância entre cada ambulância e o local do incidente. 6 – O sistema lista as ambulâncias em ordem da mais perto para a mais distante do local do incidente. 7 – O funcionário indica ao sistema a ambulância que deverá atender ao incidente. 8 – O sistema envia as instruções de despacho, a saber: endereço, configurações de ambulâncias, descrição, hospital e ambulância, ao MDT da estação que a ambulância pertence. 9 – O sistema realiza uma busca por funcionários, que estejam associados a estação da ambulância a ser despachada, que atendam as configurações de ambulância do incidente. 10 – O sistema aloca os funcionários encontrados à ambulância selecionada para atender ao incidente. 11 – O sistema realiza uma busca por equipamentos, que estejam associados a estação da ambulância a ser despachada, que atendam as configurações de ambulância do incidente. 12 – O sistema aloca os equipamentos encontrados à ambulância selecionada para atender ao incidente. 13 – O sistema salva a data e hora do despacho.

Obs.: para incidentes que exigem múltiplas ambulâncias, o cenário descrito acima deve ser realizado tantas vezes quanto for necessário. Enquanto não forem despachadas todas as ambulâncias o sistema deve considerar o incidente como aberto.

Fluxos de Eventos Variantes

Nome do Fluxo de Eventos Normal Relacionado	Variante	Descrição
Despachar Ambulância	2 – O sistema não encontra nenhum incidente aberto e a pendente de despacho.	2 - O sistema exibe uma mensagem ao funcionário informando que não existem incidentes abertos e pendentes de despacho.
Despachar Ambulância	4 – O sistema não encontra ambulâncias disponíveis para serem despachadas	4 – O sistema exibe uma mensagem ao funcionário informando que não existem ambulâncias disponíveis para despacho.
Despachar Ambulância	8 – A ambulância selecionada não se encontra na estação	8 – O sistema envia as instruções de despacho, a saber: endereço, configurações de ambulâncias, descrição, hospital e ambulância, ao MDT da ambulância selecionada. 9 – O sistema avança para o passo 8.
Despachar Ambulância	9 – O sistema verifica que a estação não dispõe de funcionários o suficiente para atender ao incidente.	9 - O sistema exibe uma mensagem ao funcionário informando que a estação da ambulância selecionada para o despacho não dispõe de funcionários o suficiente para atender as configurações de ambulância do incidente. 10 – O sistema verifica quais as ambulâncias estão mais próximas ao local do incidente que estejam disponíveis e que atendam as configurações de ambulância necessárias para o incidente desconsiderando todas as ambulâncias da estação que contém a ambulância selecionada pelo usuário como passíveis de serem selecionadas para despacho do incidente em questão. 11 – O sistema retorna para o passo 5.
Despachar Ambulância	11 – O sistema verifica que a estação não dispõe de equipamentos o suficiente para atender ao incidente.	11 - O sistema exibe uma mensagem ao funcionário informando que não a estação da ambulância selecionada para o despacho não dispõe de equipamentos o suficiente para atender as configurações de ambulância do incidente. 12 – O sistema verifica quais as ambulâncias estão mais próximas ao local do incidente que estejam disponíveis e que atendam as configurações de ambulância necessárias para o incidente desconsiderando todas as ambulâncias da estação que contém a ambulância selecionada pelo usuário como passíveis de serem selecionadas para despacho do incidente em questão. 13 – O sistema retorna para o passo 5.

Requisitos Relacionados: RF08
Classes Relacionadas: Dispatch, Ambulance, Incident, Employee e Equipment.

Descrição de Caso de Uso

Identificador do Caso de Uso: Atualizar Status da Ambulância
Caso de Uso: UC9

Descrição Sucinta: Este caso de uso é responsável pela atualização do status do despacho pela ambulância.

Fluxos de Eventos Normais

Nome do Fluxo de Eventos Normal	Precondição	Descrição
Informar Posição Geográfica		1 - O MDT informa a cada 13 segundos a localização geográfica da ambulância. 2 – O sistema atualiza a posição geográfica de onde a ambulância se encontra.
Confirmar Atendimento ao Incidente		1 – O MDT informa que a ambulância está de acordo com requisitos do incidente e irá atende-lo. 2 – O sistema altera o estado do incidente indicando que está em atendimento. 3 – O sistema altera o estado da ambulância indicando que a mesma se encontra em atendimento a um incidente e não está disponível para novos despachos.
Informar Em Rota para Hospital		1 – O MDT informa que a ambulância já fez o atendimento a vítima e está a caminho do hospital. 2 – O sistema altera o estado do incidente indicando que está a caminho do hospital e o estado da ambulância.
Informar No Hospital		1 – O MDT informa que a ambulância se encontra no hospital. 2 – O sistema altera o estado do incidente indicando que os feridos foram deixados no hospital e altera também o estado da ambulância.
Informar Em Rota para Estação		1 – O MDT informa que a ambulância já deixou a vítima no hospital e está a caminho da estação. 2 – O sistema altera o estado do incidente indicando que o incidente foi concluído com sucesso e altera o estado da ambulância.
Informar Em Espera		1 – O MDT informa que a ambulância já está na estação aguardando novas instruções de despacho. 2 – O sistema altera o estado da ambulância, libera os funcionários e os equipamentos associados a ambulância.
Informar Problema com Ambulância		1 – O MDT informa que por um motivo de força maior a ambulância não poderá continuar o atendimento. 2 – O sistema exibe uma mensagem informando ao usuário que o despacho de ambulâncias não foi concluído com sucesso. 3 – O sistema altera o estado da ambulância, funcionários e equipamentos para inativo. 4 – O sistema altera o estado do incidente para aberto. 5 – O funcionário efetua um novo despacho para o incidente. Ver Despachar Ambulância.

Fluxos de Eventos Variantes

Nome do Fluxo de Eventos Normal Relacionado	Variante	Descrição
Informar Posição Geográfica	1 – O MDT fica um período de tempo sem informar a localização geográfica.	1 – O sistema exibe uma mensagem para o funcionário informando que a ambulância não tem reportado sua localização.
Confirmar Atendimento ao Incidente	1 – Nenhuma confirmação de atendimento é recebida após 13 segundos.	1 – O sistema exibe uma mensagem para o funcionário informando que nenhuma ambulância confirmou o atendimento ao incidente.

Requisitos Relacionados: RF10, RF11, RF12 e RF13

Classes Relacionadas: Ambulance, AmbulanceSettings e Employee.

4. 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 6 – Glossário de Projeto – são apresentadas as descrições das classes presentes nos diagramas apresentados nesta seção.

4.1 - Subsistema cadastro

A Figura 4 apresenta o diagrama de classes do subsistema cadastro.

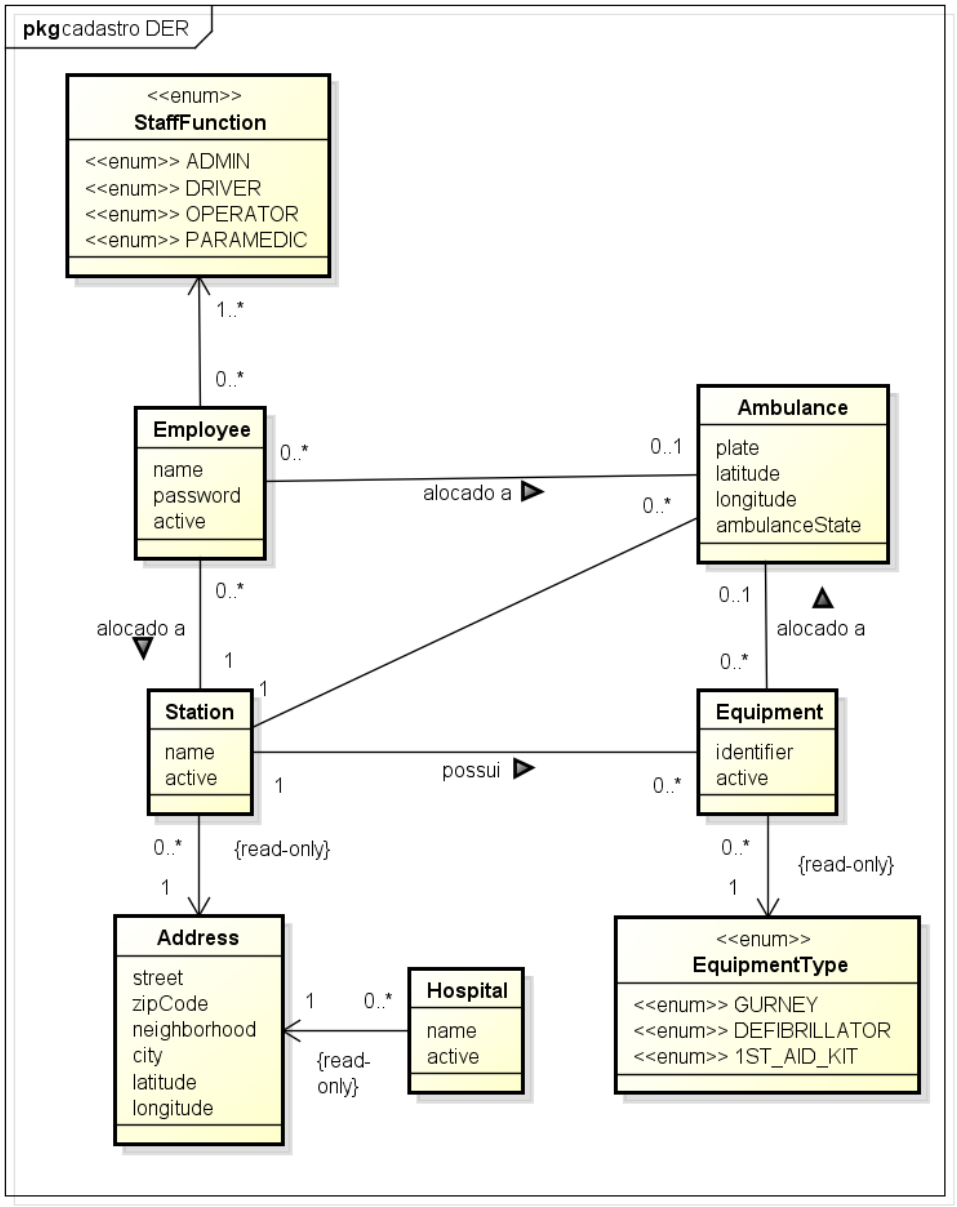

Figura 4 – Diagrama de Classes do Subsistema cadastro.

4.2 - Subsistema despacho

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

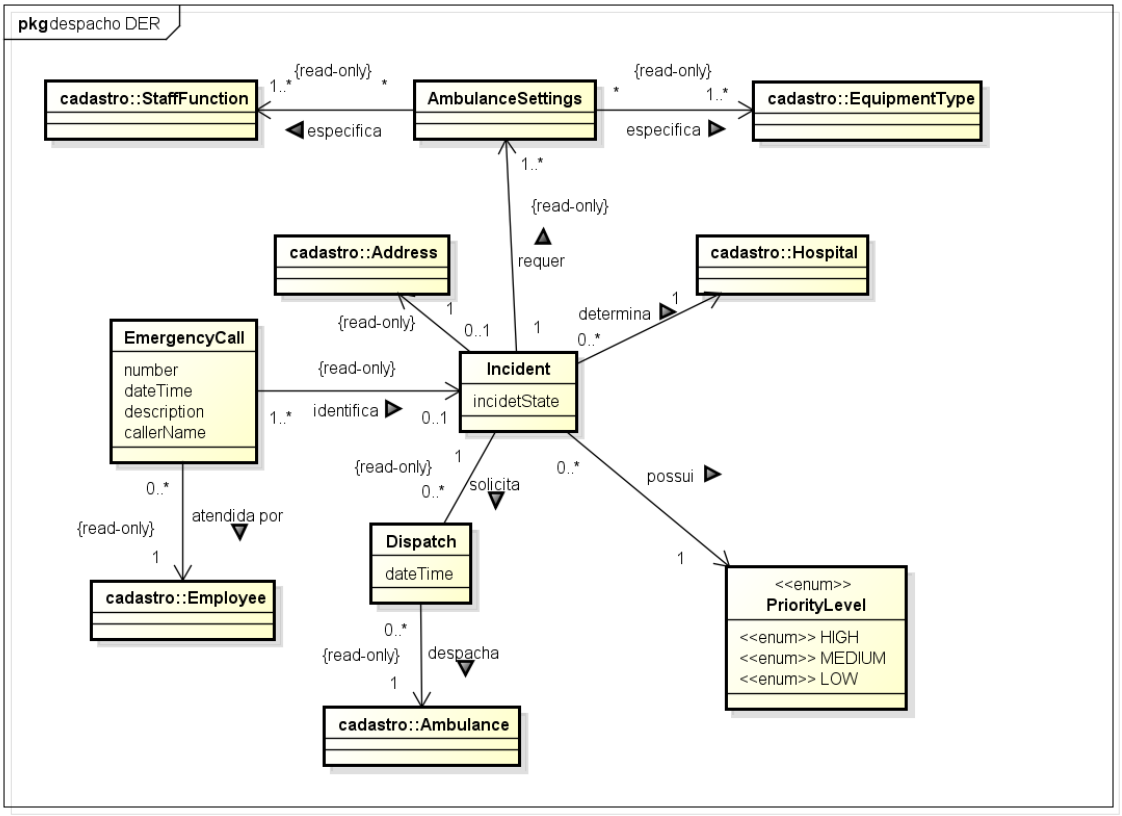

Figura 5 – Diagrama de Classes do Subsistema despacho.

5. Modelo Dinâmico

O modelo dinâmico visa capturar o comportamento dinâmico do sistema. A seguir, são apresentados os diagramas de estados elaborados no contexto deste projeto.

A Figura 6 apresenta o diagrama de estados da classe Ambulance do subsistema cadastro.

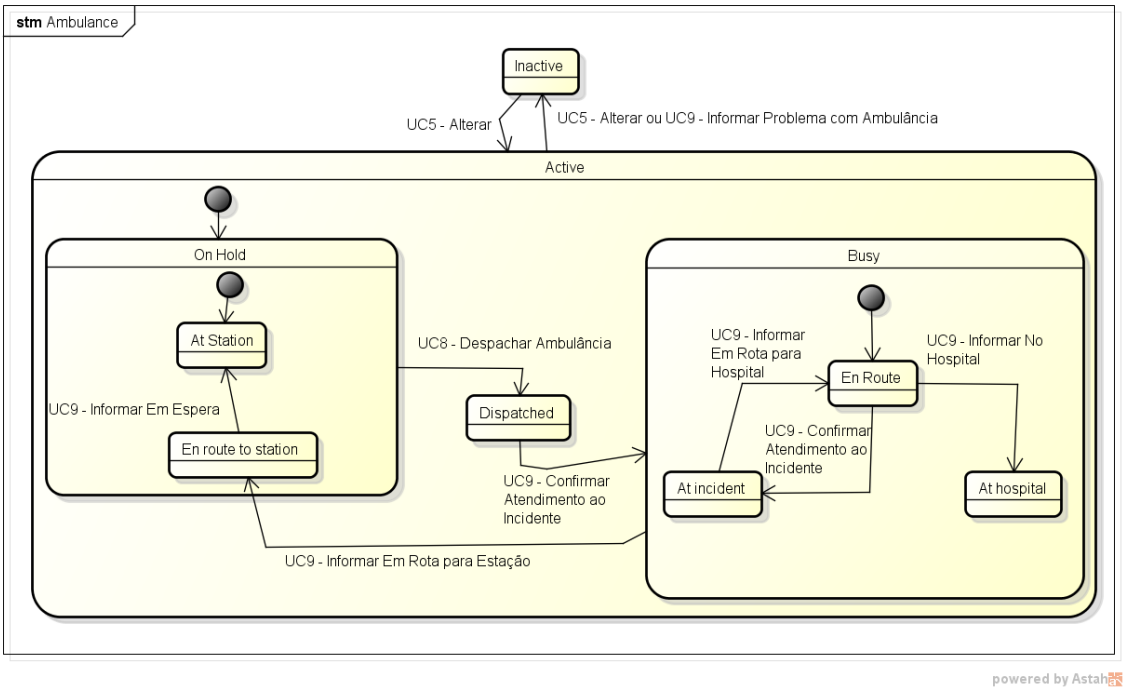

Figura 6 – Diagrama de Estados da Classe Ambulance.

A Figura 7 apresenta o diagrama de estados da classe Incidente do subsistema despacho.

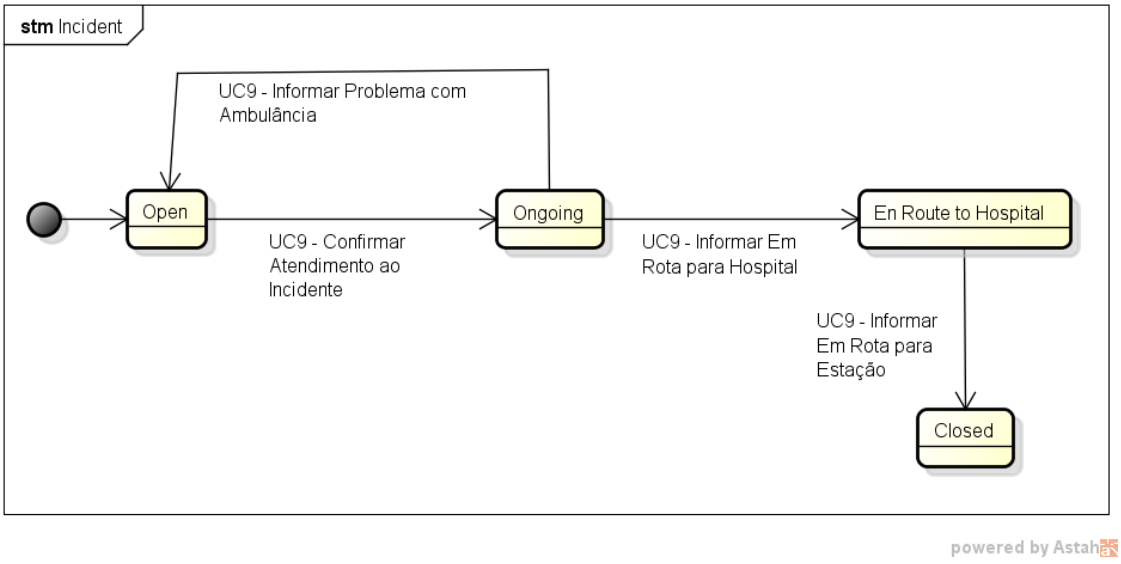

Figura 7 – Diagrama de Estados da Classe Incident.

6. Glossário de Projeto

Esta seção apresenta as definições dos principais conceitos envolvidos no projeto. Essas definições estão organizadas por subsistema.

6.1. Subsistema *cadastro*

- **Employee:** funcionários do serviço de urgência e emergência.
  - name: nome completo do funcionário
  - active: indicador de se o funcionário está ativo (V) ou não (F) no sistema. Funcionários inativos no sistema não podem atender chamadas ou participar de despacho de ambulâncias.
- **Station:** local onde as ambulâncias, funcionários e equipamentos ficam à espera de novos despachos.
  - name: nome da estação
  - active: indicador de se a estação está ativa (V) ou não (F) no sistema.
- **Hospital:** hospitais para onde poderão ser levados os feridos socorridos pelas ambulâncias.
  - name: nome do hospital
  - active: indicador de se o hospital está ativo (V) ou não (F) no sistema. Hospitais inativos no sistema não estão disponíveis para receber vítimas dos incidentes atendidos pelas ambulâncias.
- **Ambulance:** qualquer veículo usado para serviço de emergência que auxilia cidadãos em caso de incidentes.
  - plate: placa do veículo
  - ambulanceState: estado corrente da ambulância (ver diagrama de estado da classe Ambulance, Figura 6, e tipo de dados enumerado correspondente na Seção 6.3)
  - latitute: latitude da posição geográfica onde a ambulância se encontra (considerando a última vez que a posição foi informada ao sistema).
  - longitude: longitude da posição geográfica onde a ambulância se encontra (idem acima).
- **Equipment:** peça útil ao atendimento de um incidente (ex.: maca, desfibrilador, kit de primeiros socorros, etc.).
  - identifier: código que identifica unicamente o equipamento.
  - active: indicador de se o equipamento está ativo (V) ou não (F) no sistema. Equipamentos inativos no sistema não podem ser selecionados para integrar ambulâncias para despacho.

- **Address:** endereços, conforme estrutura definida abaixo:
  - street: nome da rua.
  - zipCode: código postal do endereço.
  - neighborhood: bairro do endereço.
  - city: cidade do endereço.
  - latitude: a latitude do endereço.
  - longitude: a longitude do endereço.

6.2. Subsistema *despacho*

- **EmergencyCall:** chamadas recebidas pelo serviço de urgência e emergência.
  - number: número do telefone de origem da ligação
  - dateTime: data e hora do momento do recebimento da chamada
  - description: relato feito pelo cidadão do incidente ocorrido
  - callerName: nome do cidadão que efetuou a chamada
- **Incident:** situação de urgência ou emergência.
  - incidentState: estado corrente do incidente (ver diagrama de estado da classe Incident, Figura 7, e tipo de dados enumerado correspondente na Seção 6.3)
- **Dispatch:** registro de despachos realizados.
  - dateTime: data e hora em que foi realizado o despacho pelo sistema.
- **AmbulanceSettings:** registro dos tipos de funcionários e de equipamentos necessários para o atendimento a um incidente específico.

6.3. Tipos de Dados Específicos de Domínio

- **AmbulanceState:** estados pelos quais uma ambulância pode passar. Tipo enumerado que pode assumir os seguintes valores: INACTIVE, AT STATION, DISPATCHED, EN ROUTE, EN ROUTE TO STATION, AT HOSPITAL e AT INCIDENT. Ver diagrama de estado para a classe Ambulance, na Figura 6.
- **IncidentState:** estados pelos quais um incidente pode passar. Tipo enumerado que pode assumir os seguintes valores: OPEN, ONGOING, EN ROUTE TO HOSPITAL e CLOSED. Ver diagrama de estado para a classe Incident, na Figura 7.

- **StaffFunction:** funções que cada funcionário pode assumir. Tipo enumerado que pode assumir os seguintes valores: ADMIN, DRIVER, OPERATOR e PARAMEDIC.
- **EquipmentType:** tipo de equipamento que podem existir nas estações e ambulâncias. Tipo enumerado que pode assumir os seguintes valores: GURNEY, DEFIBRILLATOR e 1ST_AID_KIT. O sistema pode ser facilmente estendido futuramente para contemplar outros tipos de equipamento.
- **PriorityLevel:** nível de prioridade que um incidente pode ter. Tipo enumerado que pode assumir os seguintes valores: HIGH, MEDIUM e LOW.

Documento de Projeto de Sistema

Projeto: **Resgate** – Despacho de ambulâncias automatizado.

Registro de Alterações:

Versão	Responsáveis	Data	Alterações
1.0	Thiago Martinho da Costa	19/07/2015	Inclusão seção 1, 2, 3 e 4.
1.1	Thiago Martinho da Costa	26/10/2015	Inclusão da seção 5.

1. Introdução

Este documento apresenta o projeto (*design*) da ferramenta Resgate. Ele está organizado da seguinte forma: a seção 2 apresenta a plataforma de software utilizada na implementação da ferramenta; a seção 3 trata de táticas utilizadas para tratar requisitos não funcionais (atributos de qualidade); a seção 4 apresenta o projeto da arquitetura de software e o projeto dos componentes da arquitetura; por fim, a seção 5 contém as referências deste projeto.

2. Plataforma de Desenvolvimento

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

Tabela 1 – Plataforma de Desenvolvimento (Utilizadas em Runtime)

Tecnologia	Versão	Descrição	Propósito
Java EE	7	Linguagem de programação OO independente de plataforma.	Implementação em código.
Java Persistence API (JPA)	2.0	API para persistência de dados por meio de mapeamento objeto-relacional.	Mapeamento das classes de domínio para tabelas do banco de dados relacional (persistência).
JavaServer Faces (JSF)	2.2	API para a construção de interfaces de usuários baseada em componentes para aplicações web.	Criação das páginas Web utilizando componentes visuais pré-prontos e sua integração com o restante da aplicação.
Contexts and Dependency Injection for Java EE (CDI)	1.1	*Framework* gratuito para injeção de dependências.	Integração das diferentes camadas da arquitetura e serviços de transação.
MySQL	6.3	*Sistema Gerenciador de Banco de Dados Relacional gratuito.*	Persistência dos dados manipulados pela ferramenta.
Wildfly 8	8.1	Servidor de aplicação baseado na plataforma JEE e implementando na linguagem Java.	Prover um servidor para as páginas *web* e ser compatível com a especificação Java EE7.

Na Tabela 2 são listadas as tecnologias utilizadas na implementação da ferramenta, bem como o propósito de sua utilização.

Tabela 2 – Tecnologias Utilizadas Durante a Implementação

Tecnologia	Versão	Descrição	Propósito
NetBeans IDE for Java	8.0.2	Ambiente de desenvolvimento (IDE) para a linguagem Java.	Facilitar a atividade de implementação de software.
Astah Community	7.0	Ferramenta para modelagem em UML.	Ferramenta de modelagem UML (*Unified Modeling Language*).
Apache Maven	3.2.5	Ferramenta de gerência de projeto baseada em *project object model* (POM)	Simplificar o download das dependências 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, além das táticas a serem utilizadas para tratá-los.

Tabela 3 – Atributos de Qualidade e Táticas Utilizadas.

Categoria	Requisitos Não Funcionais Considerados	Condutor da Arquitetura	Tática
Segurança de Acesso	RNF01	Sim	Identificar usuários usando *login* e autenticá-los por meio de senha.
Reusabilidade	RNF02	Não	Reutilizar componentes e *frameworks* existentes. Quando não houver componentes disponíveis e houver potencial para reuso, devem-se desenvolver novos

			componentes para reuso.
Confiabilidade	RNF03, RNF04	Sim	Capturar todos os erros passíveis de identificação e informar ao usuário na ocorrência de um; Atualização em tempo real de incidentes e ambulâncias.

4. Arquitetura de Software

A arquitetura de software da ferramenta Resgate baseia-se na combinação de camadas e módulos. Cada um desses módulos, por sua vez, está organizada em três camadas seguindo o proposto pelo *FrameWeb,* a saber: Lógica de Apresentação, Lógica de Negócio e Lógica de Acesso a Dados. 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 e Controle, a segunda contém o de Domínio e o de Aplicação e a terceira somente o pacote de Persistência. Cada pacote será explicado melhor nas próximas seções onde serão descritos os dois módulos do Resgate (Cadastro e Despacho). A Figura 1 apresenta a visão geral das camadas e seus pacotes juntamente com o relacionamento que existe entre eles.

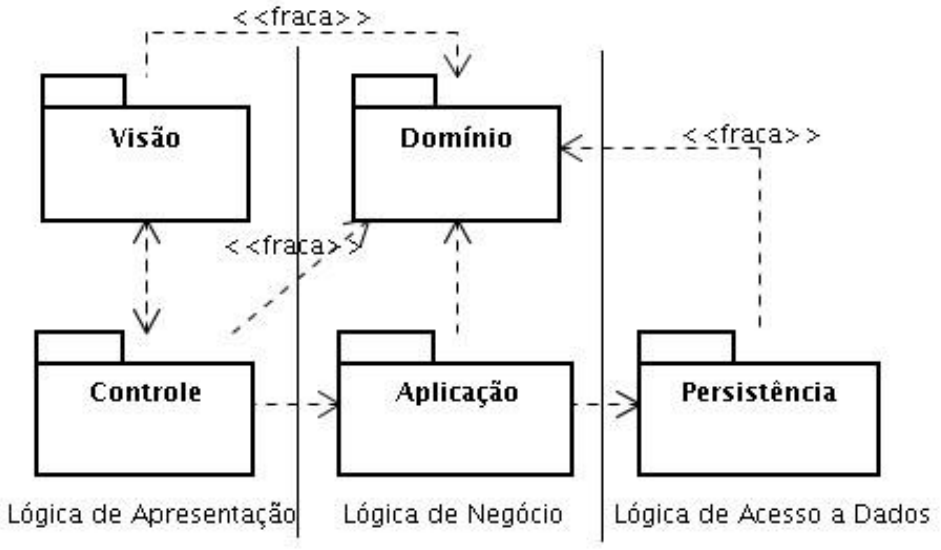

Figura 1 – Arquitetura padrão para WIS baseada no padrão arquitetônico Service Layer (FOWLER, 2002).

A figura 2 apresenta a subdivisão de cada módulo nas camadas descritas acima, a saber a camada de Lógica de Apresentação (*controller*), Lógica de Negócio, (*domain* e *application)* e Lógica de Acesso a Dados (*persistence).*

Figura 2 – Subdivisão dos módulos “Cadastro” e “Despacho” de acordo com a arquitetura de software.

4.1. Lógica de Apresentação

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

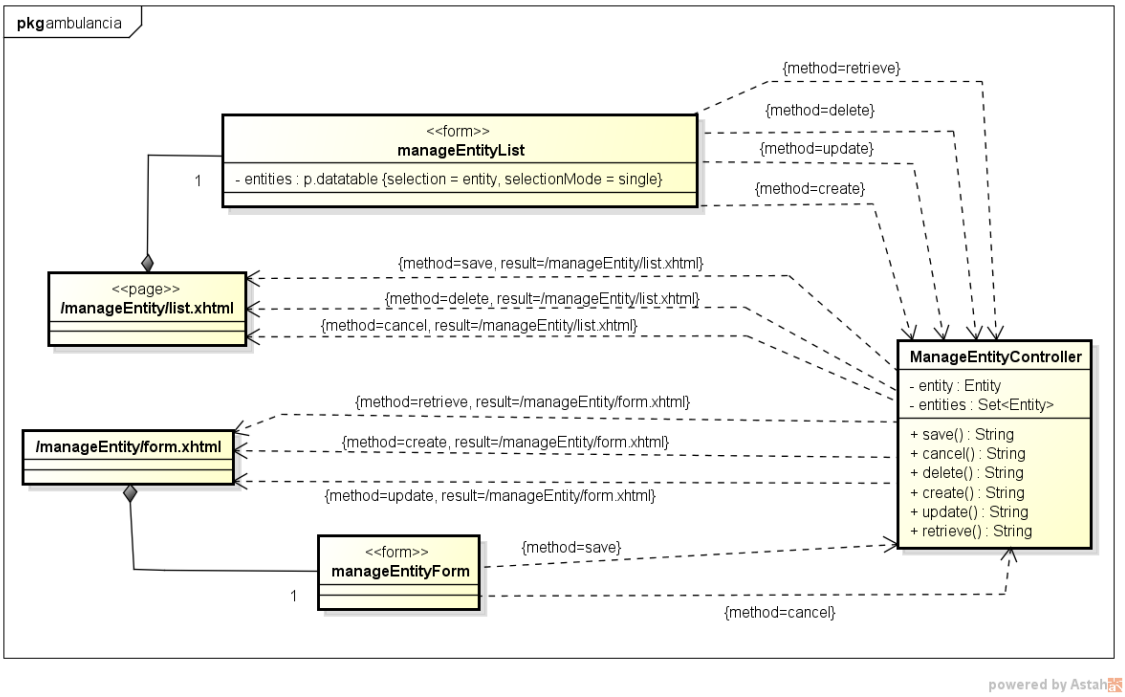

Figura 3 – Modelo de Navegação de um CRUD nemo-utils, usado como base para funcionalidades de cadastro do Resgate

Para os casos de uso que apresentam funções diferentes de apenas as básicas de cadastro, o modelo de navegação mostrado anteriormente não pode ser aplicado. Abaixo seguem os modelos de navegação para os casos de uso “Despachar Ambulância” e “Registar Chamada”.

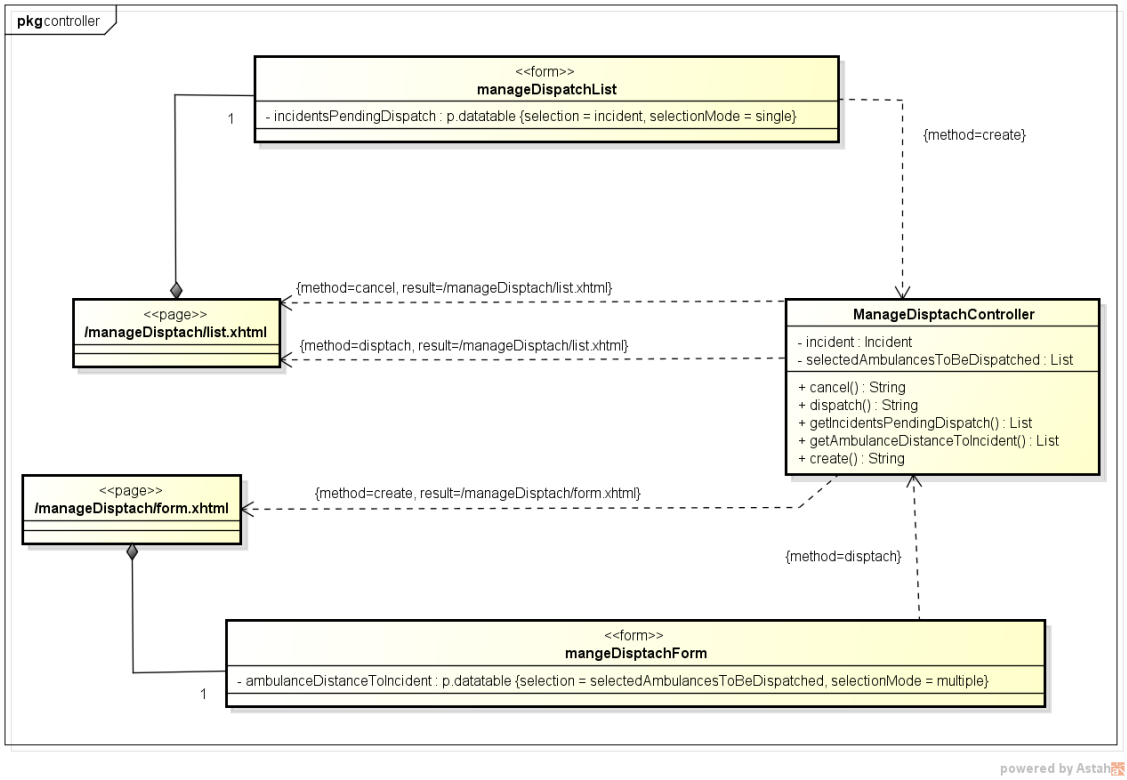

Figura 4 – Modelo de Navegação do caso de uso “Despachar Ambulância”

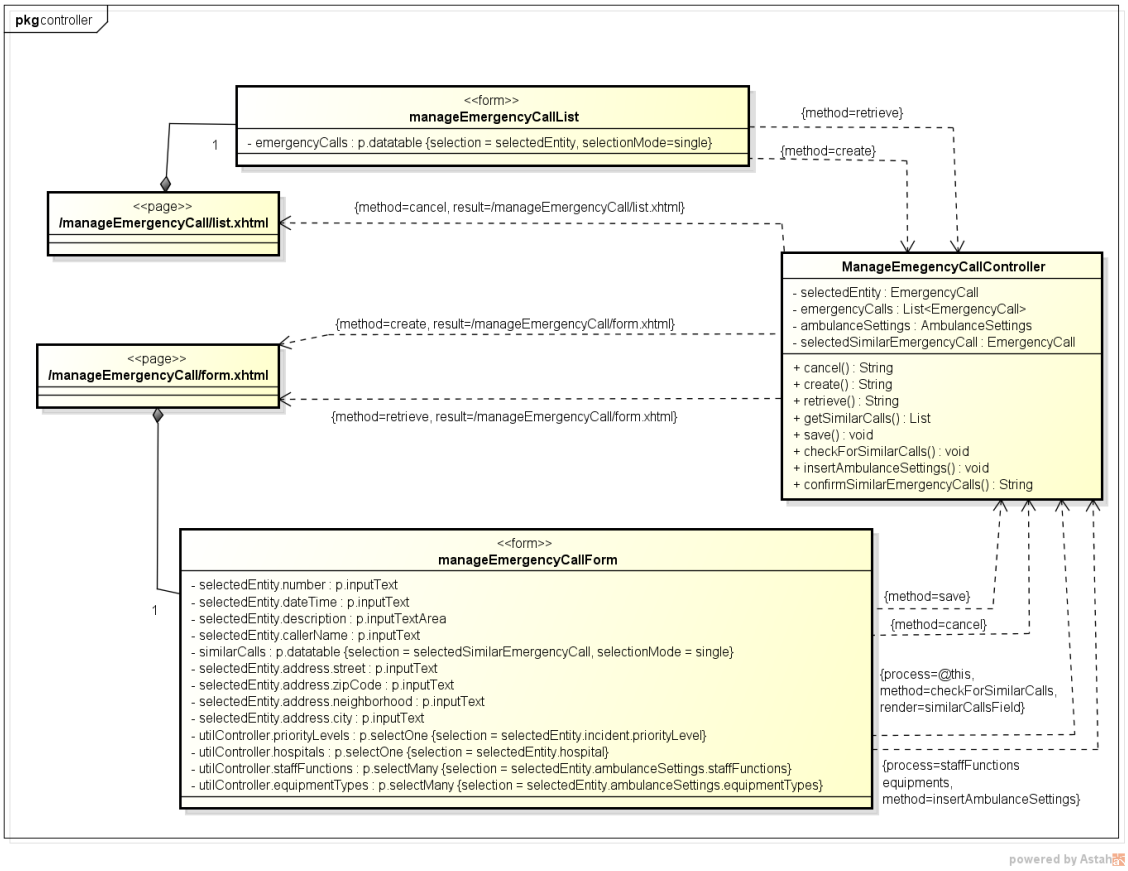

Figura 5 – Modelo de Navegação do caso de uso “Registrar Chamada”

4.2. Lógica de Negócio

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

Todas as classes de domínio estendem de *PersistentObjectSupport* do pacote *nemo-utils*, onde essa herança não é mostrada no diagrama acima com o intuito de não poluir o diagrama com várias associações.

Abaixo segue o modelo de domínio para os módulos “Cadastro” e “Despacho”.

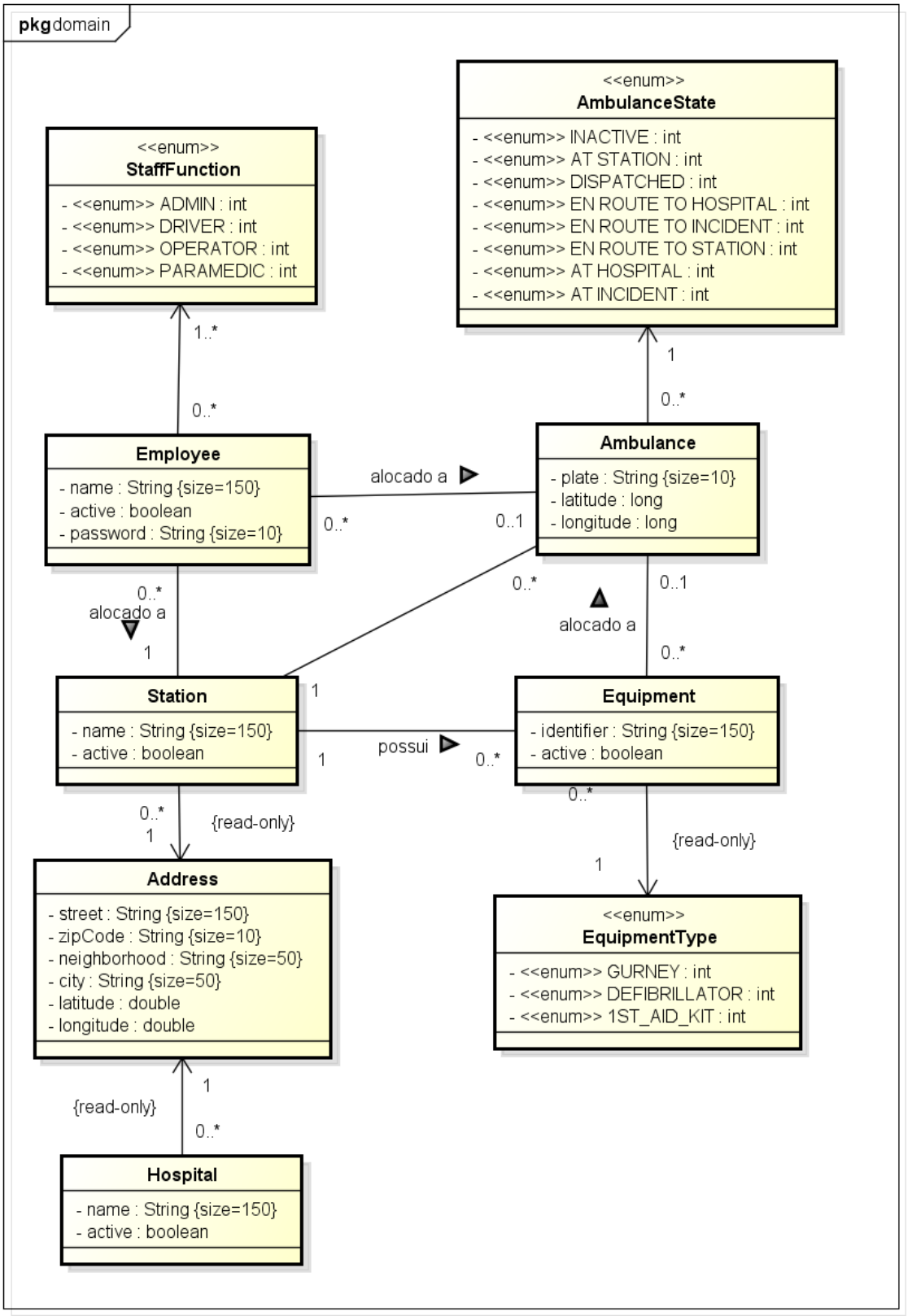

Figura 6 – Modelo de Domínio do Resgate para o módulo “Cadastro”

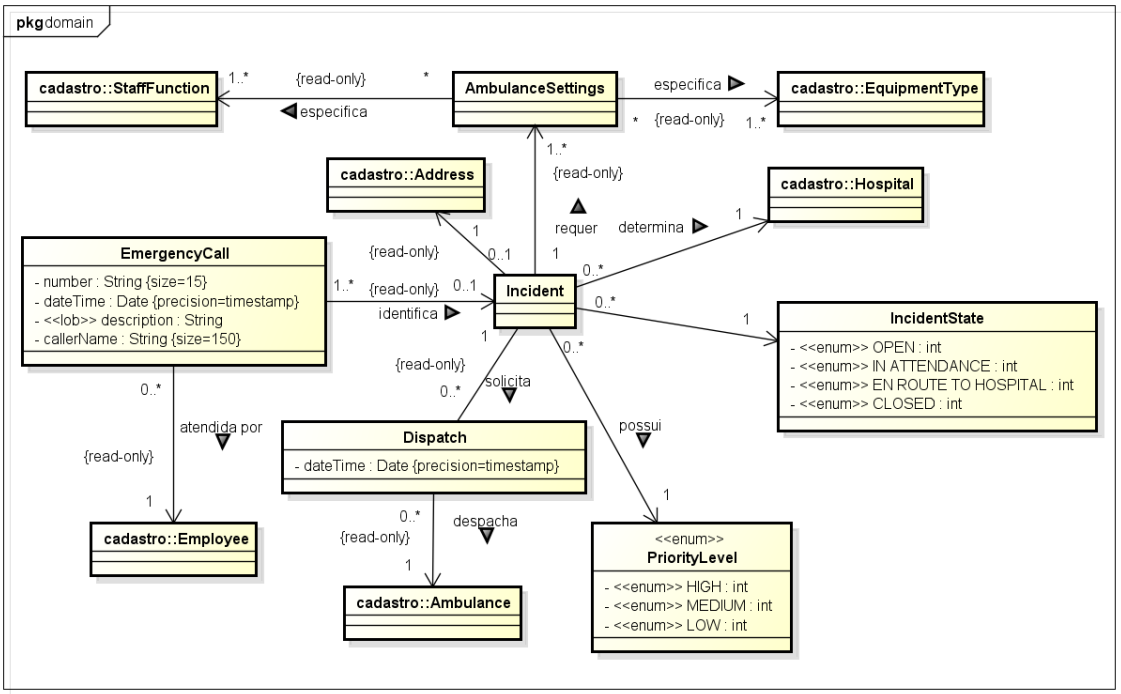

Figura 7 – Modelo de Domínio do Resgate para o módulo “Despacho”.

Todas as classes de aplicação estendem de *CrudServiceBean* do pacote *nemo-utils,* tal classe está representada abaixo de forma genérica. Da mesma forma dos diagramas anteriores essa herança não é mostrada no diagrama acima com o intuito de não poluir o diagrama com várias associações. Abaixo, segue o modelo de aplicação para os módulos “Cadastro” e “Despacho”.

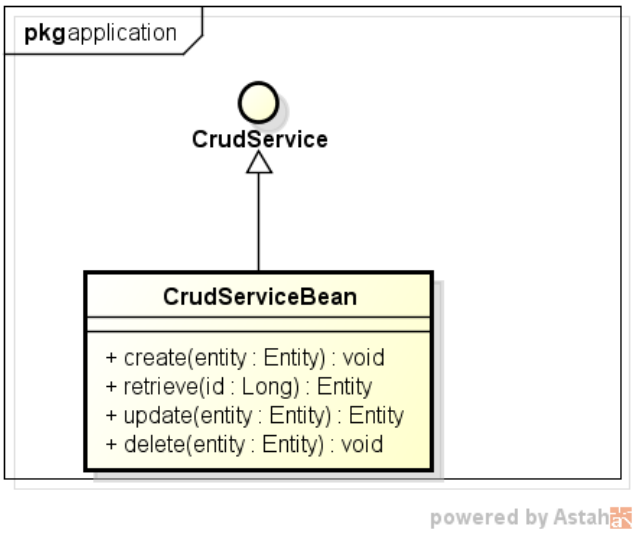

Figura 8 – Modelo de Aplicação genérica da ferramenta *nemo-utils*.

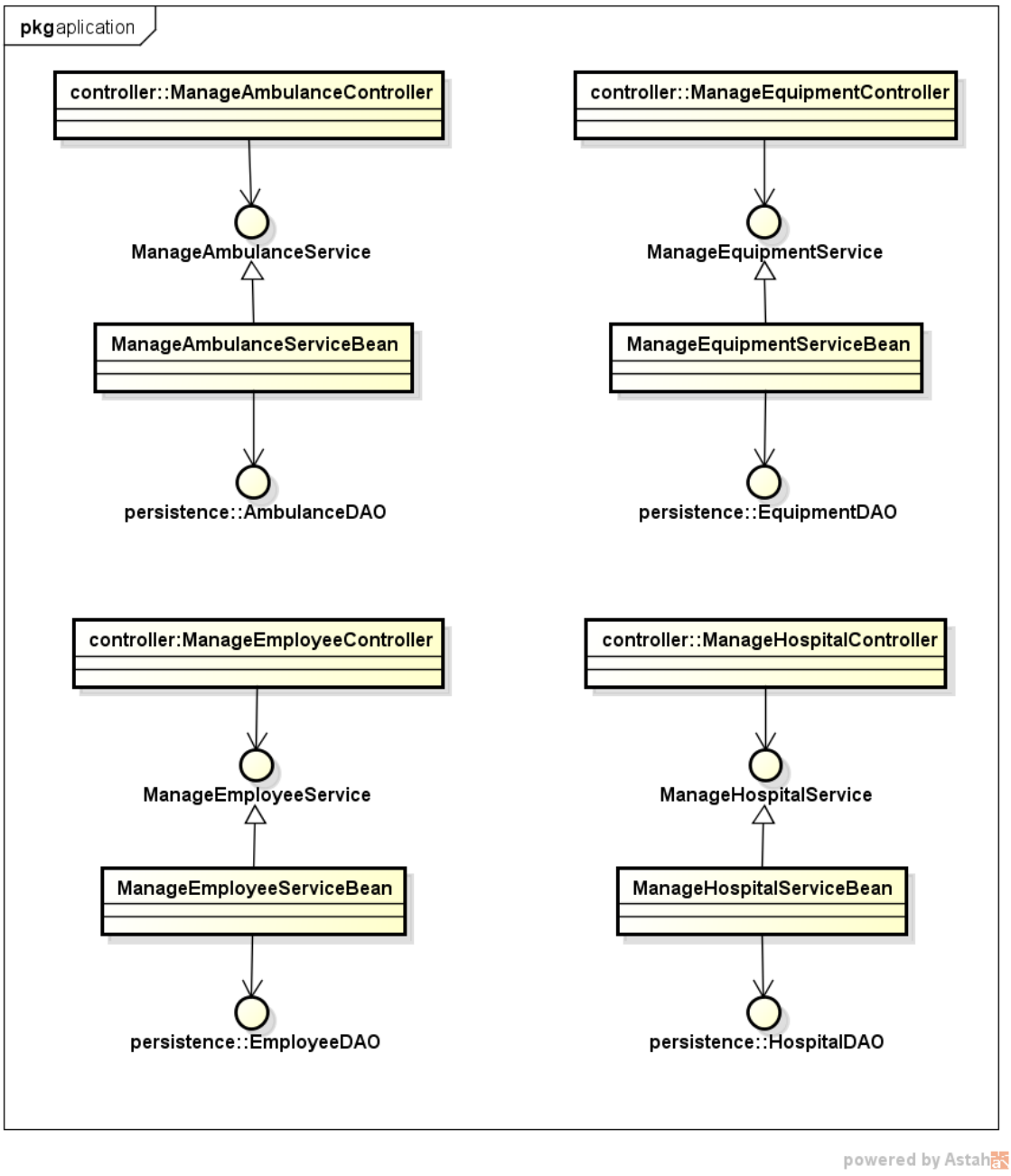

Figura 9 – Modelo de Aplicação do Resgate para o módulo “Cadastro”.

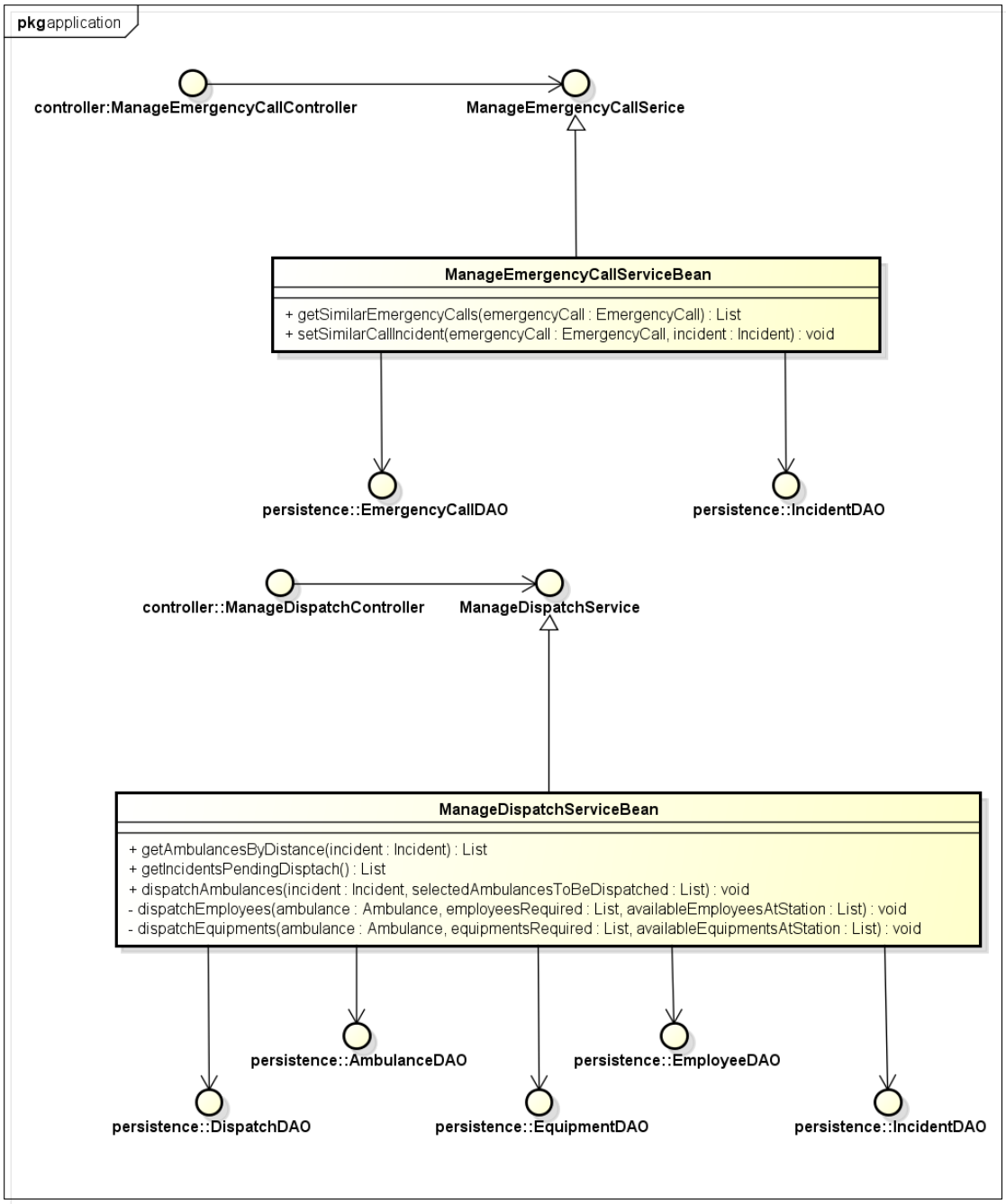

Figura 10 – Modelo de Aplicação do Resgate para o módulo “Despacho”.

4.3. Lógica de Acesso a Dados

Nas figuras abaixo são apresentados os Modelos de Persistências desenvolvidos para o projeto de Resgate, os mesmos foram usados na implementação do pacote de persistência.

Vale notar que o nome das classes já indica qual tecnologia de

persistência foi utilizada, esse sistema de nomenclatura é mais uma sugestão do *FrameWeb* para simplificar o processo de software. Vale notar também que abaixo está representado o diagrama de persistência genérico provido pela ferramenta *nemo-utils* e o modelo para os módulos “Cadastro” e “Despacho”.

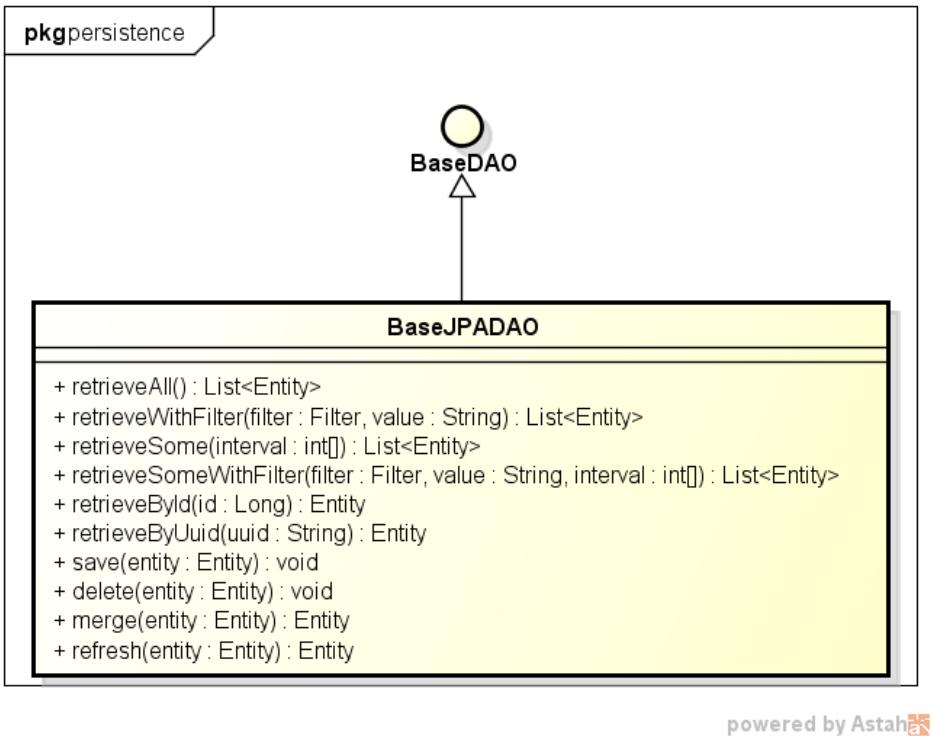

Figura 11 – Modelo de Persistência genérico da ferramenta *nemo-utils.*

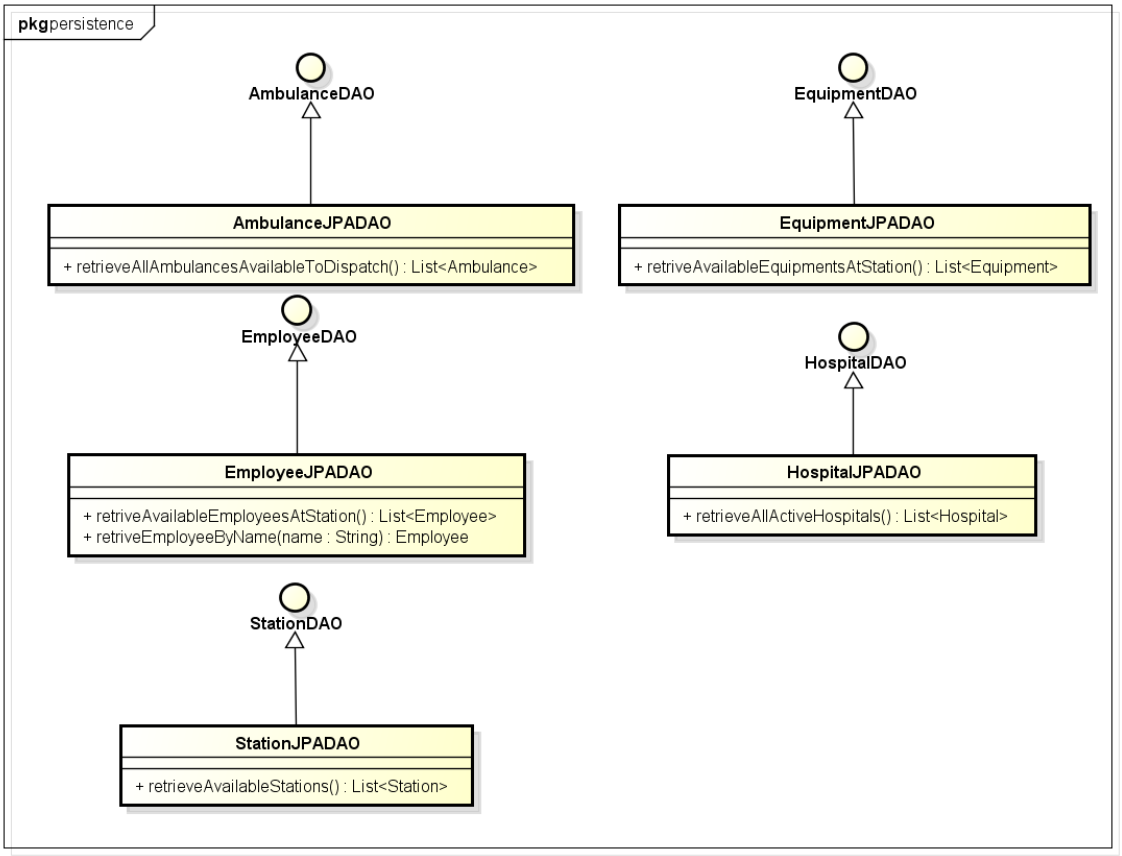

Figura 12 – Modelo de Persistência do Resgate para o módulo “Cadastro”.

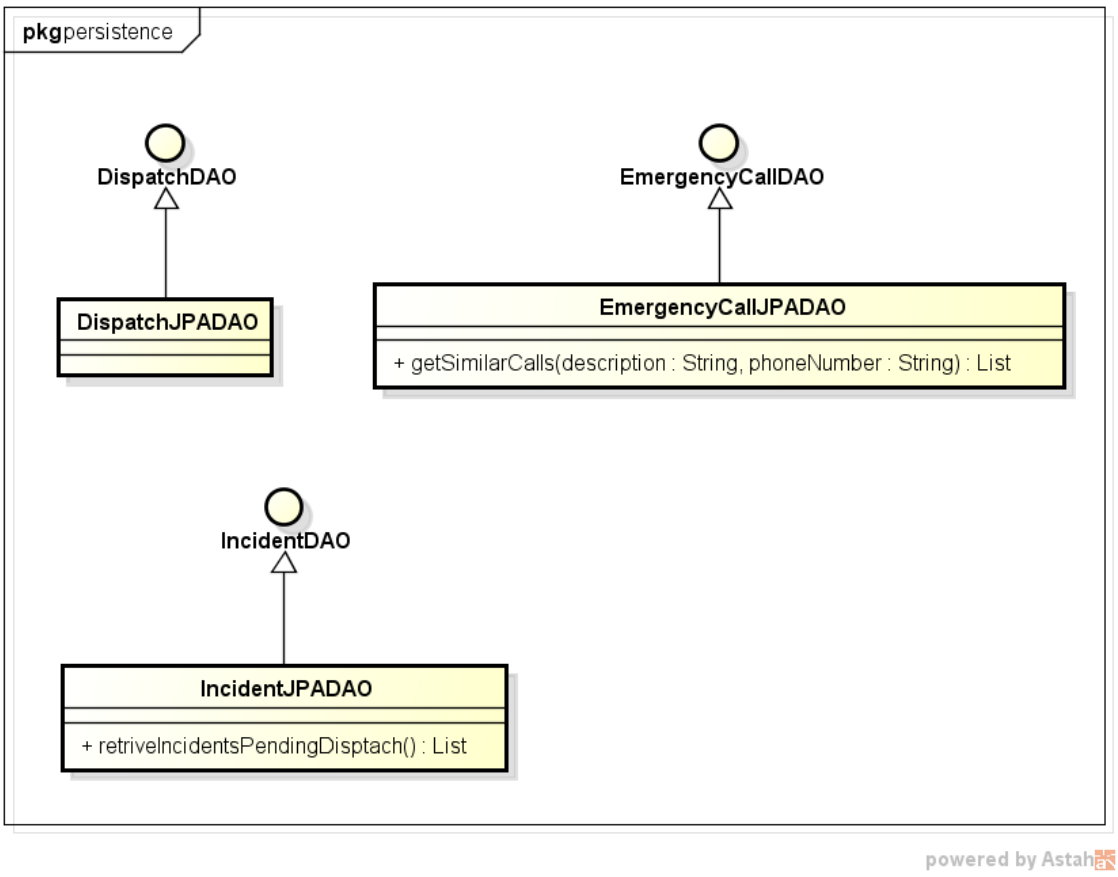

Figura 13 – Modelo de Persistência do Resgate para o módulo “Despacho.

Note que a relação de herança entre os DAOs específicos e o DAO base não é representada explicitamente nos diagramas para evitar poluição visual. Esta também é uma recomendação do *FrameWeb*, ficando, portanto, o desenvolvedor incumbido de derivar essa relação implicitamente ao analisar o modelo.

5. Referências

FOWLER, M. **Patterns of Enterprise Application Architecture**. Addison-Wesley, ISBN 0321127420, novembro 2002.

SOUZA, VÍTOR ESTÊVÃO SILVA. ***FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*.** Dissertação de Mestrado (Programa de Pós-Graduação em Informática)—Universidade Federal do Espírito Santo, 2007.