UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Newton Alex Sander

Um *Framework* para Construção de Aplicações *Web* com Dados Espaciais e Espaço-Temporais

Florianópolis
2013

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Newton Alex Sander

Um *Framework* para Construção de Aplicações *Web* com Dados Espaciais e Espaço-Temporais

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de bacharel em Sistemas de Informação pela Universidade Federal de Santa Catarina.
Orientadora: Profª. Dra. Vania Bogorny

Florianópolis
2013

Newton Alex Sander

UM *FRAMEWORK* PARA CONSTRUÇÃO DE APLICAÇÕES *WEB* COM DADOS ESPACIAIS E ESPAÇO-TEMPORAIS

Trabalho de graduação aprovado como requisito parcial para a conclusão do Curso de Sistemas de Informação, do Departamento de Informática e Estatística do Centro Tecnológico, da Universidade Federal de Santa Catarina.

Florianópolis, 30 de janeiro de 2013.

_____
Prof. Leandro José Komosinski, Dr.
Coordenador do Curso

Banca Examinadora:

_____
Prof.ª Vania Bogorny, Dr.ª
Orientadora
Universidade Federal de Santa Catarina

_____
Prof. João Bosco Mangueira Sobral, Dr.
Universidade Federal de Santa Catarina

_____
Prof.ª Patrícia Vilain, Dr.ª
Universidade Federal de Santa Catarina

RESUMO

O uso crescente de dispositivos que geram informações localizadas no tempo e no espaço, ou seja, dados espaço-temporais, aumentou também a necessidade de manipulá-las. Contudo, não há muitas ferramentas para realizar essa manipulação e que possam ser utilizadas no desenvolvimento de *softwares* capazes de executar tarefas genéricas, como a descoberta de um objeto móvel entrando em uma área geográfica na qual não tem permissão e de um objeto móvel ultrapassando a velocidade máxima permitida para determinada área etc. Assim, este trabalho tem como objetivo a criação de um *framework* que identifique eventos por meio da análise de dados espaço-temporais de objetos móveis e o desenvolvimento de uma aplicação *web* que utilize as funcionalidades disponibilizadas por esse *framework*. A aplicação *web* desenvolvida identifica os eventos relacionados a objetos móveis entrando ou saindo de áreas geográficas de acesso permitido ou proibido; ao período de permanência de um objeto móvel dentro ou fora de uma área de acesso permitido ou proibido; e à velocidade mínima ou máxima permitida de um objeto móvel. Para validar o funcionamento do *framework* foram desenvolvidas três avaliações, as quais mostram a viabilidade do uso comercial ou acadêmico do *framework*. Objetiva-se que o *framework* apresentado possa ser utilizado por desenvolvedores para facilitar a criação de *websites* envolvendo dados espaço-temporais.

Palavras-chave: Dados espaço-temporais. Dados espaciais. Análise de dados. *Framework.*

ABSTRACT

The increasing use of devices that generate data located in time and space, namely, spatio-temporal data, has increased the need to manipulate them. However, there are not many tools to accomplish this manipulation and to be used in the development of softwares capable to execute generic tasks, such as the discovery of a mobile object entering in a geographical area in which it is not allowed to enter or the discovery of a mobile object exceeding the maximum allowed speed for certain area, etc.

Thus, this work had as goal the creation of a framework capable of identifying events through the analysis of spatio-temporal data of mobile objects and the development of a web application that uses the features provided by this framework. The developed web application identifies the events related to mobile objects entering or leaving geographic areas with forbidden or allowed access; the period that a mobile object stayed inside or outside an area with forbidden or allowed access; and to the minimum and maximum allowed speed for a mobile object. To validate the operations of the framework, three evaluations were built, which show the viability of commercial and academic use of the framework. The goal is that the framework can be used by developers to facilitate the creation of websites with spatio-temporal-data.

Keywords: Spatio-temporal data. Spatial data. Data Analysis. Framework.

SUMÁRIO

1 INTRODUÇÃO E MOTIVAÇÃO

Devido à popularização de dispositivos móveis coletores de pontos localizados no tempo e no espaço (como os sistemas de posicionamento global – do inglês *Global Positioning System* (GPS) –, os telefones celulares, as redes de sensores, entre outros) e à grande quantidade de aplicações que podem se beneficiar com o uso dos dados coletados por tais dispositivos, existe uma demanda crescente por sistemas que manipulem dados espaço-temporais.

Algumas aplicações de manipulação de dados espaço-temporais são a análise de comportamento de presidiários em regime semiaberto, o monitoramento de animais, a análise de mobilidade humana, o monitoramento de veículos, etc. Um sistema para monitoramento de criminosos, por exemplo, poderia funcionar do seguinte modo: cada presidiário usa uma pulseira com um dispositivo coletor da localização geográfica, a qual é periodicamente enviada para um centro de processamento de dados, que verifica se o presidiário está dentro da área permitida. Caso ele ultrapasse os limites dessa área, um alarme deve soar.

Já em uma aplicação para o controle de frotas em empresas de transporte de mercadorias, cada veículo poderia ser equipado com um dispositivo móvel, que coletaria a sua localização geográfica e o tempo. Esses dados poderiam ser enviados em tempo real ou posteriormente para um centro de processamento de dados, registrando a trajetória do veículo. Desse modo, poder-se-ia analisar as trajetórias dos veículos em busca de desvio de rota, parada não planejada, direção agressiva ou, ainda, para o planejamento do horário de chegada ao destino.

As aplicações que manipulam dados espaço-temporais são mais complexas que as aplicações que utilizam dados convencionais, pois manipulam não só os dados não espaciais, mas também os dados geográficos e a dimensão tempo. Além disso, tais aplicações fazem uso de consultas mais complexas envolvendo o tempo, o espaço e as informações semânticas, por exemplo: buscar um aumento excessivo da velocidade de um objeto móvel (CARBONI, 2011); buscar trajetórias que se interseccionam em um certo intervalo de tempo; ou identificar determinados padrões de comportamento, como perseguição (SIQUEIRA; BOGORNY, 2011) ou desvio (ALVARES, L. O; LOY, A.M.; RENSO, C.; BOGORNY, V., 2011).

Com esses dois exemplos de aplicação já fica clara a complexidade das aplicações de manipulação e de análise de dados espaço-temporais, bem como a importância de realizar

essas análises em tempo real em aplicações *web*, possibilitando o acesso remoto às informações.

Para realizar essas análises em sistemas *web*, considera-se necessário o desenvolvimento de *websites* de conteúdo dinâmico, tarefa que até hoje pode ser considerada complicada. Apesar da não tão recente abordagem de *websites* de conteúdo dinâmico ter facilitado muito a vida do desenvolvedor, existe agora a similaridade entre os projetos de *websites* dinâmicos e as grandes possibilidades de reúso dos seus códigos-fonte, que não são aproveitadas, por exemplo, os códigos de procedimentos de autenticação, de controle de sessão e de comunicação com o banco de dados.

O reúso de códigos-fonte no desenvolvimento de *websites* de conteúdo dinâmico explicitado acima foi quase sanado com o advento de *web frameworks full stack*, os quais proveem as funções básicas que uma aplicação *web* necessita disponibilizar e que até então deveriam ser programadas para cada sistema. Entre essas funções estão o serviço de aplicação, um mecanismo de armazenamento de dados, um sistema de *templates*, um gerenciador de requisições, um módulo de autenticação e um *kit* para Asynchronous JavaScript and XML (AJAX).

Podemos citar como exemplo de *framework full stack*, o *web framework* Django, o qual é composto por vários

> elementos fortemente acoplados que foram desenvolvidos desde o início e trabalham muito bem juntos. Ele inclui um ORM bastante poderoso e simples de usar; possui uma interface de administração on-line que torna possível editar os dados no banco de dados a partir de um navegador; e possui um motor de *template* baseado em texto desenvolvido para ser utilizado por *webdesigners* que não conhecem Python e que suporta herança de *templates* e filtros. (KUBICA, 2012)

Desse modo, Django "é um *web framework fullstack* no sentido de que ele provê todos os componentes para uma aplicação *web*: acesso ao banco de dados, *request framework*, lógica de aplicação, motor de *templates* etc." (FORCIER; BISSEX; CHUN, 2009).

Com o uso de *web frameworks full stack* o desenvolvedor pode dedicar-se apenas ao desenvolvimento do conteúdo da aplicação, pois geralmente a modificação das funções essenciais desta ocorre apenas quando os requisitos do projeto do *website* possuem especificidades. Esse é o caso, por exemplo, do funcionamento do sistema de *templates*, os quais são modificados apenas quando necessita-se de uma alteração específica. Isso porque, conforme descreve Oracle (2012), "os *templates* são modelos para impor um leiaute-padrão e

o estilo visual em diversas páginas ou nas regiões de conteúdo. Assim, ao alterar um *template*, todas as páginas ou regiões baseadas nele são automaticamente modificadas.".

O desenvolvimento de aplicações *web* que utilizam dados espaço-temporais possui necessidades específicas. Entre os componentes úteis para essas aplicações estão um banco de dados com suporte a dados geográficos; uma biblioteca de *object-relational mapping* (ORM); um servidor *hypertext transfer protocol* (HTTP) para recebimento dos dados geográficos emitidos pelos dispositivos coletores; um servidor *web*; um motor de *templates*; e *templates web* para a visualização dos dados geográficos ou dos eventos gerados por eles.

As aplicações *web* que utilizam dados geográficos podem fazer um amplo uso de *web frameworks full stack* genéricos, porém, ainda existe muito retrabalho para o desenvolvedor que pode ser extirpado por meio da agregação de um módulo de ORM para dados geográficos; da adoção de *plug-ins* que adicionem o suporte de dados geográficos a sistemas gerenciadores de bancos de dados (SGBDs); e de funções espaço-temporais que tornem transparentes ao desenvolvedor as consultas com objetos geográficos.

Com base nos problemas mencionados, encontrou-se a motivação para realizar este trabalho, que visa à criação de um *web framework* específico para a manipulação de dados espaço-temporais. Esse *framework* proporcionará ao desenvolvedor, além dos serviços essenciais (sistema de *templates*, servidor de aplicação e gerenciador de requisições), um módulo ORM com suporte a dados geográficos; um servidor para o recebimento e o armazenamento dos dados geográficos; uma interface para a visualização dos dados localizados no tempo e no espaço; e, por fim, um pacote de funções espaço-temporais, que fornecerão uma maior abstração na manipulação de dados geográficos, não sendo necessária a utilização direta das operações espaciais disponíveis no banco de dados geográfico.

O *web framework* proposto poderá ser utilizado em aplicações nas quais é necessário fazer análise on-line de dados geográficos, os quais podem ser inseridos no banco de dados por meio do serviço de recebimento de dados provido pelo próprio *framework*. Além disso, também poderá facilitar o desenvolvimento e os testes de novos algoritmos para manipular e analisar dados geográficos que poderão ser rapidamente incorporados por meio da extensão do *framework*.

Assim, ao desenvolvedor da aplicação caberá apenas: a criação dos algoritmos de análise de dados espaço-temporais necessários para o domínio da aplicação que não estiverem implementados no *framework*; a definição dos fatos que geram eventos (e.g. o objeto entra em área proibida); o desenvolvimento de tratamentos para o evento gerado (e.g. soar um alarme);

e, por fim, a criação da interface específica ao domínio da aplicação que não tem relação com a manipulação dos dados geográficos (e.g. uma página para cadastro de presidiários contendo número de cadastro de pessoa física (CPF), a foto e o nome completo dos indivíduos).

No exemplo da aplicação *web* de controle de frotas, o desenvolvedor que utilizar o *framework* possuirá à sua disposição os algoritmos que geram os eventos quando o caminhão com um dispositivo móvel estiver fora da área geográfica para a qual é planejado o seu tráfego; delongar-se trafegando por uma área específica; ou, ainda, quando deslocar-se com velocidade muito elevada ou muito baixa, saindo da velocidade exigida para o tráfego na área.

As demais partes do trabalho estão organizadas da seguinte forma: o capítulo 2 contém os conceitos necessários para o entendimento do domínio ao qual o trabalho se refere. No capítulo 3 é feita a descrição da estrutura, do desenvolvimento e do funcionamento do *framework*. No capítulo 4 são descritas aplicações que estendem o *framework*. Por fim, o capítulo 5 apresenta a conclusão do trabalho.

2 CONCEITOS

Neste capítulo são apresentados os conceitos necessários para o entendimento do domínio que motivou o desenvolvimento deste trabalho.

2.1 *FRAMEWORK*

Uma grande quantidade de aplicações possui uma considerável quantia de código-fonte que pode ser reutilizada por aplicações semelhantes. As linguagens de programação orientadas a objeto proporcionaram um reúso mais aprimorado do que os pacotes de funções forneciam, isto é, propiciam o reúso de *design*, que é a utilização de uma arquitetura definida para a solução do problema. Esse tipo de código utilizado para reúso é chamado de *framework*.

Diferentemente de bibliotecas, que fornecem ao desenvolvedor apenas partes de código reutilizáveis, os *frameworks* oferecem, além do código implementado, uma arquitetura que o desenvolvedor deve utilizar e um espaço bem-definido para o desenvolvimento do código customizado da aplicação.

Segundo Johnson e Foote (1988), em tradução livre, "uma importante característica de um *framework* é que os métodos definidos pelo usuário que utiliza o *framework* serão frequentemente chamados pelo próprio *framework*, e não pela aplicação do usuário". Os autores também definem que o *framework* geralmente é responsável pela coordenação e pelo sequenciamento da atividade da aplicação.

Johnson e Foote (1988) ainda descreveram com concisão os benefícios, as metodologias de desenvolvimento e a classificação de *frameworks*. Essa classificação nos diz que os *frameworks* podem ser *white-box* ou *black-box*, conforme as definições explicadas sucintamente a seguir.

Framework white-box: é aquele no qual o usuário necessita adicionar métodos em subclasses em uma ou mais classes do *framework*, e cada método adicionado para a subclasse necessita manter as convenções internas das classes superiores. Para utilizar um *framework white-box* é necessário conhecer sua implementação, por isso são chamados de "caixa-branca".

Framework black-box: a utilização deste tipo de *framework* é feita por meio de componentes. O usuário necessita conhecer apenas a interface destes para utilizar o *framework*, por esse motivo é chamado de "caixa-preta". *Frameworks black-box* são mais difíceis de desenvolver, porém, mais fáceis de utilizar, e são considerados por Johnson e Foote (1988) uma evolução ideal para *frameworks white-box*.

Segundo Riehle (2000), os desenvolvedores que utilizam um *framework*, reutilizam a sua arquitetura e implementação para resolver um problema por meio de uma aplicação que se encaixa no domínio modelado pelo *framework*. Ao reutilizar a arquitetura, que deve ser bem entendida, customiza-se a arquitetura de um novo *software* para solucionar um problema específico. Já ao reutilizar a implementação, os programadores desenvolvem as aplicações mais rapidamente.

Aplicações *web* possuem uma grande quantidade de código-fonte que pode ser reutilizado e que gera uma grande complexidade quando este não é fundamentado por uma boa arquitetura de *software*.

Uma ferramenta criada para acessar o banco de dados – provendo consultas, inserções e outras funcionalidades –, se bem desenvolvida, pode ser utilizada por qualquer aplicação *web* que utilize banco de dados. O mesmo ocorre para a autenticação; para o processo de tradução de *templates* em código *HyperText Markup Language* (HTML); o controle e o despacho de requisições HTTP; o controle de sessão; o mapeamento de *uniform resource locator* (URL); ou, ainda, para uso de AJAX.

Assim, o objetivo de um *framework web* é prover uma boa arquitetura de *software* e, acoplada a ela, as funções que possuem alta possibilidade de reúso e que possam ser estendidas pelo desenvolvedor, incumbindo a ele a codificação do conteúdo específico da aplicação que está sendo criada.

O *framework* desenvolvido neste trabalho estende o *framework* para aplicações *web* CherryPy. Ambos são do tipo *white-box*, e, por consequência, seus códigos-fonte precisam ser bem entendidos para que possam ser estendidos.

2.2 DADOS GEOGRÁFICOS E DADOS ESPAÇO-TEMPORAIS

Dados geográficos são dados que possuem uma localização com relação à superfície terrestre (CÂMARA, G.; CASANOVA, M. A.; HEMERLY, A. S.; MAGALHÃES, G. C.; MEDEIROS, C. M. B., 1996).

Segundo Braz e Bogorny (2012), dados georreferenciados são comumente caracterizados a partir de três componentes fundamentais, quais sejam: as características não espaciais, que descrevem o fenômeno sendo estudado (por exemplo, o nome de um município, a população); as características espaciais, que descrevem a localização espacial do objeto ou fenômeno, ou seja, sua geometria; e os relacionamentos espaciais, que representam o relacionamento de vizinhança entre objetos geográficos (e.g. intercepta, cruza, sobrepõe etc.).

Um objeto móvel é o objeto que se desloca no tempo e no espaço. Uma pessoa caminhando e um satélite orbitando são exemplos deste tipo de objeto. Já como dispositivos móveis consideramos os telefones celulares, os aparelhos de GPS e outros dispositivos capazes de coletar e enviar a localização espaço-temporal dos objetos móveis.

Quando um objeto móvel carrega um dispositivo que captura a localização e o tempo, o dispositivo gera um dado espaço-temporal. Dados espaço-temporais são dados de localização espacial que também possuem a informação temporal da coleta. E um objeto móvel com um dispositivo que coleta os dados de espaço e de tempo estará registrando sua *trajetória*.

Com esse tipo de dado, diferentemente de um dado somente espacial, é possível realizar cálculos para chegarmos a informações como a velocidade de deslocamento, a aceleração, a localização em tempo real de um objeto móvel, entre muitas outras.

2.3 OPERAÇÕES ESPACIAIS DE UM BANCO DE DADOS GEOGRÁFICO

Os bancos de dados geográficos requerem operações específicas para manipular dados geográficos. Essas operações são suportadas pela maioria dos Sistemas Gerenciadores de Banco de Dados Geográficos e estão disponíveis na linguagem SQL espacial. Entre elas, estão as funções de teste de relacionamentos espaciais, como toca, atravessa, contém, intersecciona etc.

Neste trabalho, as operações espaciais são operações implementadas pelo Sistema Gerenciador de Banco de Dados (SGBD), enquanto as funções espaço-temporais (detalhadas a seguir) são funções desenvolvidas neste trabalho e inseridas no *framework* para que as aplicações possam fazer análises on-line de dados espaço-temporais.

2.4 FUNÇÕES ESPAÇO-TEMPORAIS

Função espaço-temporal é um termo definido neste trabalho para identificar o método aplicado aos dados espaço-temporais com o objetivo de identificar eventos semânticos.

As funções espaço-temporais fazem a busca e a identificação do evento desejado analisando os dados espaço-temporais provenientes de dispositivos móveis. Uma função espaço-temporal pode, por exemplo, fazer a identificação de ocorrências de velocidade excessiva nos dados enviados por um aparelho celular acoplado a um caminhão.

No exemplo anterior, de uma função geográfica que busca ocorrências de velocidade excessiva, é perceptível a necessidade de parametrização do método de identificação dos fatos, pois não seria viável manter um limite de velocidade máxima igual para objetos de diferentes tipos (ciclistas, carros, caminhões etc). Por esse motivo, as funções espaço-temporais de cada aplicação possuem uma parametrização individual para cada dispositivo emissor de dados espaço-temporais.

As funções espaço-temporais desenvolvidas e prontas para uso neste *framework* são SPEED, INTERSECTS_AREA e NOT_INTERSECTS_AREA, as quais identificam, respectivamente, eventos relacionados à velocidade e a objetos móveis interseccionando e não interseccionando áreas. A função SPEED recebe parâmetros de velocidades máxima ou mínima para gerar evento. As funções de intersecção de área recebem parâmetros referentes à velocidade máxima e mínima dentro da área e, ainda, de tempo máximo dentro e fora da área.

2.5 EVENTO SEMÂNTICO EM DADOS ESPAÇO-TEMPORAIS

Neste trabalho, o termo evento semântico é definido como a identificação de um fato desejado nos dados espaço-temporais cuja ocorrência possui significado no contexto da aplicação.

Por exemplo, a Figura 1 exibe um objeto móvel que representa um detento localizado em uma região exterior à área do presídio, definindo o evento semântico fuga, o qual ocorre quando se identifica o fato do dado espaço-temporal do objeto móvel estar localizado fora da área permitida. Já um objeto móvel que representa um funcionário do presídio localizado fora da área da penitenciária não constitui um evento semântico de fuga, por isso a análise dos dados não gera esse evento.

Figura 1 – Evento semântico de tipo fuga
Fonte: elaborada por Vinicius Rosa (2012).
Nota: figura elaborada para este trabalho.

Um evento, no contexto deste trabalho, possui o carimbo de tempo do momento de seu início e, caso tenha terminado, de seu fim. Também possui um indicador que define se o evento está ativo ou não.

2.6 *OBJECT-RELATIONAL MAPPING* (ORM) (MAPEAMENTO OBJETO-RELACIONAL)

Apesar do desenvolvimento de SGBDs orientados a objeto ter começado há mais de duas décadas, seu uso continua restrito se comparado ao dos sistemas relacionais. Até mesmo quando poderia haver um aumento significativo de produtividade, visto que a maioria das linguagens de programação utilizadas atualmente são orientadas a objeto, opta-se pelo uso de SGBDs relacionais.

Esse paradigma gera uma dificuldade explícita de manipulação de dados do modelo relacional utilizando o modelo orientado a objetos, como a que ocorre ao lidarmos com granularidade, herança, associações e navegação de dados.

Um componente de mapeamento objeto-relacional serve para abstrair essa divergência dos paradigmas relacional e orientado a objetos, criando uma camada que deixa transparente para o desenvolvedor a estrutura relacional do banco de dados. Dessa forma, o desenvolvedor acessa uma base de dados orientada a objetos virtual e o componente ORM fica responsável por associar as modificações feitas nesses objetos com as modificações dos dados relacionais, ou seja, é responsável pelo mapeamento. Muitas consultas SQL também tornam-se transparentes para o desenvolvedor, pois os componentes proveem métodos para o acesso simplificado aos objetos.

Neste trabalho foi empregada a biblioteca SQLAlchemy para o mapeamento objeto-relacional, pois ela possui uma forte comunidade de usuários e sua documentação é abundante. Utilizamos também a biblioteca GeoAlchemy a fim de suprir a falta de suporte a dados geográficos apresentada pela SQLAlchemy. Com o uso dessas duas bibliotecas é possível manipular os dados espaço-temporais provenientes do banco de dados sem a necessidade de redigir consultas em SQL.

2.7 *SOFTWARES*

A seguir são listados os mais importantes *softwares* e bibliotecas que são utilizados pelo *framework* para que este suporte todas as funcionalidades exigidas.

2.7.1 Python

Python é uma linguagem de programação de código aberto utilizada tanto em corporações como na academia para uma grande variedade de propósitos, sendo considerada uma ótima linguagem para desenvolvimento *web*.

Segundo Martelli e Ascher (2002), "Python é uma excelente linguagem para desenvolvimento web e [...] vem com a maioria dos módulos de que você necessitará." Além disso, Python possui diversas bibliotecas, o que, conforme Huffman (2009) (um dos fundadores do agregador social Reddit), é um dos principais benefícios dessa linguagem.

Ainda conforme Martelli e Ascher (2002),

> comparado a outras linguagens de programação, Python parece ter um número bastante elevado de servidores de aplicação e linguagens de *templating*. Embora seja fácil de desenvolver qualquer coisa para a *web* em Python, seria peculiar fazê-lo sem antes olhar os servidores de aplicação disponíveis. Em vez de recriar continuamente páginas dinâmicas e scripts, a comunidade Python assumiu a tarefa de construir esses servidores de aplicação para permitir que outros usuários criem conteúdo de forma fácil.

Para completar o suporte ao desenvolvimento, há uma vasta documentação oficial. Mas esses não são os únicos pontos fortes de Python. Outro grande benefício dessa linguagem, além da facilidade de aprendê-la e utilizá-la, é a organização do código, que é altamente legível. Segundo Huffman (2009), "[...]um bom código Python tem uma estrutura bastante óbvia. […] É extremamente expressivo, extremamente legível e extremamente fácil de escrever. E isso deixa a vida mais suave."

Esses são alguns dos motivos pelos quais organizações mundialmente reconhecidas em diversas áreas – de games à ciência, e até mesmo o governo norte-americano – utilizam Python, por exemplo, Google™, Yahoo® Maps, Walt Disney Feature Animation®, Jasc® Software, National Aeronautics and Space Administration (NASA) e muitos outros.

2.7.2 PostGis

O Postgis adiciona suporte a tipos espaciais ao SGBD PostgreSQL, provendo os tipos de dados, os índices e as operações para dados geográficos, bem como as funcionalidades essenciais a dados não geográficos que devem ser fornecidas pelo SGBD, como a integridade

transacional, o controle de conflitos, a replicação, as cópias de segurança e a formulação de consultas por meio da linguagem SQL.

Esse *software* tem suporte certificado aos tipos e às funções para dados geográficos especificadas pelo Open Geospatial Consortium (OGC), organização que define padrões para dados espaciais. Entre os tipos suportados pelo Postgis estão o ponto, a linha, o polígono, o multiponto, o multipolígono e a multilinha. Além disso, suporta funções de relacionamentos e medidas espaciais, como área, azimute, distância, intersecção, atravessa, toca, centroide, entre muitas outras.

2.7.3 SQLAlchemy

SQLAlchemy é uma biblioteca Python que provê o mapeamento objeto-relacional e um pacote de funções que adicionam uma camada maior de abstração entre a comunicação do *software* com o SGBD, permitindo, por exemplo, executar consultas no banco de dados utilizando expressões em Python.

Essa biblioteca provê também um sistema de representação do esquema dos dados que pode tanto emitir instruções em *data definition language* (DDL) como gerar esquemas por introspecção utilizando a estrutura existente do banco de dados. Também provê o mapeamento dos tipos dos dados do banco de dados com os tipos da linguagem Python, podendo utilizar dialetos de SQLite, Postgresql, MySQL, Oracle, MS-SQL, Firebird, Sybase entre outros.

A biblioteca SQLALchemy é utilizada por grandes corporações em *softwares* amplamente utilizados, como o repositório Dropbox®, o navegador Mozilla Firefox®, o agregador social Reddit®, a distribuição Linux Fedora™, o *software* de *cloud computing* Openstack™, o *site* para desenvolvimento de pesquisas SurveyMonkey™, o repositório de códigos abertos SourceForge®, entre outros. (SQLALCHEMY, 2012)

2.7.4 GeoAlchemy

A GeoAlchemy é uma biblioteca Python desenvolvida devido à ausência de suporte a bancos de dados espaciais na biblioteca SQLAlchemy e a estende fornecendo suporte para os SGBDs Postgres com PostGIS, Spatialite, MySQL, Oracle® e MS SQL Server 2008.

A GeoAlchemy possibilita ao desenvolvedor utilizar os tipos geométricos fornecidos pelo banco de dados em seu *software*, permitindo, inclusive, usar o esquema do banco de dados codificado para gerar instruções DDL. Provê, ainda, o acesso às operações espaciais específicas do SGBD, como toca e intersecciona ou a distância e a comparação entre os objetos geográficos. A operação é executada pelo SGBD, mas codificada pelo desenvolvedor com uma camada de abstração e expressões em Python em vez de em SQL, o que seria mais comum.

Possui também a funcionalidade de suporte ao *Spatial Reference System Identifier* (SRID) – em português, sistema identificador de referência espacial –, um identificador que serve para diferenciar os dados geográficos gerados em sistemas de coordenadas diferentes. Dessa forma, o dado geográfico pode ser transferido de um banco de dados para outro, com sistema de coordenadas diferente, sem que haja perda de valor ou significado.

O suporte ao SRID facilita ao desenvolvedor a utilização de dados de diversos sistemas de coordenadas geográficas, pois a transformação dos dados de um sistema de coordenadas para outro é feito de forma quase transparente.

2.7.5 Genshi

A biblioteca Genshi é responsável pela geração dinâmica das páginas requisitadas pelo navegador *web* populando e transformando estruturas *extensible markup language* (XML), que possuem variáveis e *tags* Python, em páginas HTML.

De acordo com a sua página oficial, Genshi é "uma biblioteca Python que provê um conjunto integrado de componentes para análise, geração e processamento de HTML, XML ou de outro conteúdo textual para geração de saídas para *web*." (EDGEWALL SOFTWARE, 2012).

Sua principal característica é a linguagem de marcação inteligente para criação de *templates*. Isso significa que, diferentemente das linguagens convencionais, a Genshi entende

as diferenças entre *tags*, atributos e nós de textos, o que trata-se de uma grande vantagem. (EDGEWALL SOFTWARE, 2012)

Ademais, modelos XHTML simples fazem a Genshi ser fácil de usar mesmo para *webdesigners* que não conhecem Python. Essa facilidade é apoiada, ainda, pela documentação abrangente, que engloba tanto a documentação da aplicação como um guia de usuário.

2.7.6 CherryPy

CherryPy é um *framework web* orientado a objetos utilizado em aplicações *web* Python. Oferece ao desenvolvedor os componentes essenciais de aplicações *web*, como o controle de sessão, de arquivos estáticos, de *cookies* e de *cache*, a codificação de caracteres, a autorização e a compressão.

Devido ao fato do *framework* CherryPy não ser intrusivo, não afeta a escolha de seus usuários, fornecendo um conjunto de ferramentas para qualquer desenvolvedor sem fazer qualquer suposição sobre o modo que ele ou ela optará para usá-las. (HELLEGOUARCH, 2007) Dessa forma, deixa a cargo do desenvolvedor o uso e a escolha de ferramentas para a tradução de *templates*, para o ORM, para a codificação de formulários e também a utilização de bibliotecas Javascript. No entanto, apesar de não prover tais ferramentas, existe uma vasta gama de opções para esses componentes e sua adição à estrutura do *framework* é bastante simples.

O ponto crucial para o uso do CherryPy neste trabalho é o fato de este ser um *framework* simples, independente e não intrusivo. Conforme Hellegouarch (2007),

> um dos principais objetivos sempre foi manter o CherryPy tão simples quanto possível, com o objetivo de evitar que a biblioteca cubra a engenharia do projeto. Graças ao âmbito restrito coberto pela biblioteca, os desenvolvedores têm sido capazes de se concentrar na aplicação e no retorno à comunidade.

Além disso, o núcleo do CherryPy, isto é, suas funções mais importantes, não necessitam de pacotes Python de terceiros para trabalhar, mas apenas de pacotes da biblioteca-padrão de Python, o que mostra a sua autonomia. (HELLEGOUARCH, 2007)

3 UM *FRAMEWORK* PARA APLICAÇÕES GEOGRÁFICAS

Neste capítulo são explicitadas a abrangência e o potencial das aplicações geográficas, bem como o *framework* proposto.

3.1 APLICAÇÕES GEOGRÁFICAS

São definidas como aplicações geográficas, neste trabalho, as aplicações que suportam o armazenamento, a manipulação e a visualização de dados geográficos ou espaço-temporais.

Aplicações que oferecem ao usuário informações geográficas sempre foram de grande importância. Começaram a ser desenvolvidas para computadores na década de 1960 pelo governo do Canadá e desde então sua utilização só aumentou. As aplicações geográficas foram empregadas no mapeamento criminal, em sistemas de geografia histórica, hidrologia, sensoriamento remoto, para realizar rastreamento, entre outros.

Com a popularização da internet e, mais recentemente, com a *web* 2.0, as aplicações geográficas também foram popularizadas, tornando seu uso bastante comum. Elas têm sido utilizadas em sistemas de recomendação nos quais a localização é decisiva – como no caso de busca de hotéis e restaurantes próximos do usuário em redes sociais baseadas em localização – ou, ainda, para localização geográfica de lugares importantes, como em um mapa inserido em um *site* empresarial que apresenta a localização da matriz e das filiais da empresa.

Para se ter uma ideia da dimensão dessa popularização, basta reparar na quantidade de usuários da rede social baseada em localização chamada Foursquare™, que é de 25 milhões; já o Wikimapia, um *site* de mapeamento colaborativo, tem mais de 11 milhões de lugares marcados.

Excluindo esses *websites* com um número muito grande de acessos, como o Foursquare™ e o Wikimapia, podemos analisar, ainda no âmbito da *web*, uma quantidade enorme de aplicações geográficas para fins específicos, mas que operam em uma vasta gama de áreas, como é o caso de aplicações para o controle de frotas ou para o rastreamento de objetos, de animais ou de pessoas. Essas aplicações fazem intenso uso de análise de comportamento, de consultas de dados espaço-temporais e de manipulação em tempo real de dados espaço-temporais de objetos móveis.

Um exemplo de uma aplicação espaço-temporal pode ser o monitoramento de veículos

que transportam cargas frágeis, como frutas e verduras, em busca da identificação de aumentos excessivos de velocidade (CARBONI, 2011). Essa aplicação pode ter a finalidade de, após a definição de regiões onde ocorrem acelerações bruscas, avisar o motorista para que tome cuidado ao transitar pelas regiões definidas, prevenindo, assim, aumentos bruscos de velocidade que poderiam provocar danos no produto transportado e até mesmo acidentes de trânsito. Ou poderia, ainda, avisar o supervisor do motorista sobre seu mau comportamento.

A partir dos dados espaço-temporais fornecidos pelos dispositivos móveis nos veículos é possível visualizar, em tempo real, a localização do objeto móvel e o trajeto percorrido ou aplicar o algoritmo que identifica as movimentações bruscas do veículo.

Outro exemplo de aplicação é a de monitoramento de delinquentes em regime semiaberto. Um dispositivo móvel preso ao tornozelo do sujeito envia a localização e o tempo deste para um centro de processamento de dados. Em regime semiaberto há uma área na qual é permitido que o recluso esteja, portanto, quando ele sai da área permitida está quebrando a lei e deve ser punido. A aplicação geográfica é responsável por analisar os dados espaço-temporais da trajetória do delinquente a fim de definir se ele está em área não permitida para, assim, soar um alarme informativo caso o sujeito esteja burlando a lei.

Há ainda várias aplicações de funcionamento similar que se aplicam a parques fabris, aeroportos e outros grandes empreendimentos, como hidrelétricas e mineradoras. Nesse tipo de aplicação, é necessário saber a localização, em tempo real, de produtos, máquinas, veículos e pessoas. Em aeroportos, por exemplo, quando um avião pousa, são necessários pelo menos três veículos para efetuar o desembarque de carga e passageiros: um para o reboque do avião, um para unir a escada à aeronave e um para o transporte de bagagens. Caso cada um dos veículos do aeroporto possua um dispositivo móvel, é possível saber, usando uma aplicação geográfica, onde está o veículo mais próximo do avião que deve ser auxiliado.

Com os poucos exemplos citados anteriormente já é possível constatar a importância das aplicações geográficas. No entanto, mesmo com a crescente necessidade e com a importância dessas aplicações, há uma demanda que não tem sido atendida por ferramentas para desenvolvimento de aplicações espaço-temporais para *web* que suportem o recebimento, a manipulação e a visualização de dados espaço-temporais.

Para atender essa demanda, este trabalho tem por objetivo o desenvolvimento de um *framework* que suporte as funcionalidades necessárias para o desenvolvimento de aplicações como as anteriormente descritas.

3.2 *FRAMEWORK* PROPOSTO

A partir do levantamento de requisitos necessários para o desenvolvimento de aplicações *web* que façam a análise de dados espaço-temporais em tempo real, foram definidas as funcionalidades que o *framework* suportaria. São elas: (1) o recebimento de dados geográficos por requisições HTTP ou de carregamento destes a partir de um arquivo XML; (2) a análise dos dados em busca de eventos semânticos; (3) a possibilidade de uso de funções espaço-temporais que possam ser associadas aos dispositivos móveis, podendo ser parametrizadas e assim gerar os eventos específicos para os objetivos de projeto; (4) a disponibilização de interface para a visualização dos dados espaço-temporais utilizando Google™ Maps; e (5) a possibilidade de uso do esquema de banco de dados definido para suportar essas funcionalidades. Definidas as funcionalidades, desenvolveu-se o *framework web* proposto, chamado WebGeoFramework, utilizando uma junção de *frameworks* e bibliotecas Python.

O diagrama de módulos apresentado na Figura 2, a seguir, demonstra a organização conceitual, em camadas lógicas, da estrutura utilizada para o funcionamento do *framework* e da aplicação geográfica disponibilizada por ele. As camadas lógicas demonstradas são *Ferramentas*, *WebGeoFramework* e *AppFramework*, as quais representam, respectivamente: as ferramentas utilizadas pelo *framework*, o *framework* desenvolvido neste trabalho e a aplicação geográfica disponibilizada pelo *framework*.

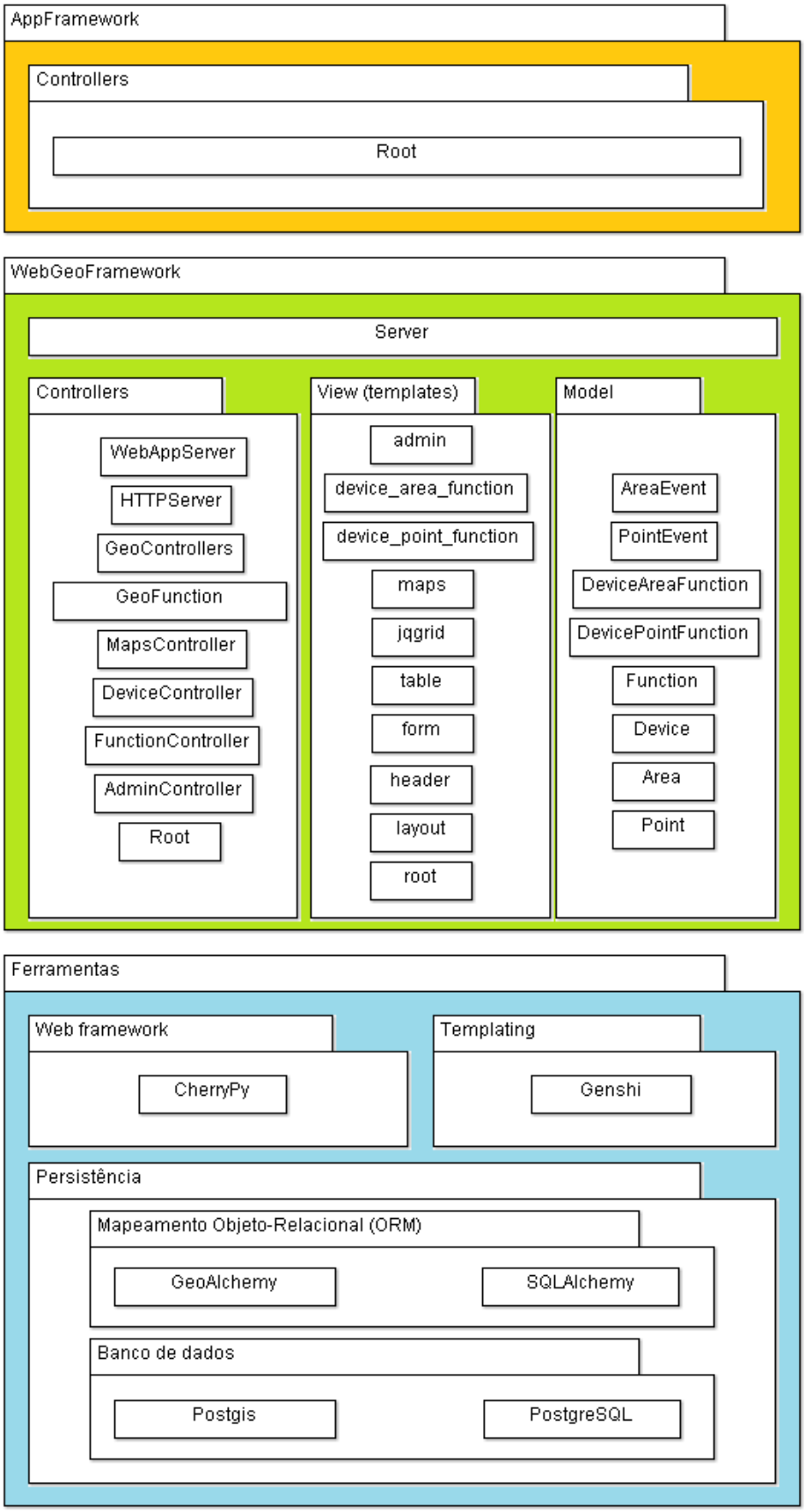

Figura 2 – Diagrama de módulos da estrutura do *framework*
Fonte: elaborada pelo autor (2012).

3.2.1 Ferramentas

Na camada *Ferramentas*, na parte inferior da Figura 2, estão localizados os *softwares* de persistência e de *templating* e o *web framework* utilizados, os quais são descritos a seguir.

3.2.1.1 Persistência

Para a persistência, isto é, o armazenamento dos dados, o banco de dados com suporte a dados geográficos utilizado foi o PostgreSQL com o *plugin* PostGIS, que provê índice, tipos e operações para manipular dados geográficos. Esses *softwares* foram escolhidos devido ao amplo uso comercial e acadêmico que possuem, o que proporciona grande quantidade de documentação. Outro ponto crucial para a escolha é o fato de ambos possuírem código-fonte aberto e livre.

Ainda na camada lógica de persistência, o componente de mapeamento objeto-relacional utilizado foi o SQLAlchemy, uma biblioteca Python que provê, além do mapeamento objeto-relacional, um pacote de funções que adiciona uma camada maior de abstração na comunicação entre o *software* e o SGBD, permitindo, por exemplo, executar consultas no banco de dados utilizando expressões em Python.

Havia ainda a necessidade de que a biblioteca de mapeamento objeto-relacional suportasse dados geográficos, por isso, optou-se pelo uso da GeoAlchemy, desenvolvida por causa da ausência de suporte a bancos de dados espaciais na biblioteca SQLAlchemy.

3.2.1.2 *Web framework*

O *web framework* utilizado foi o CherryPy, sendo este o responsável por todo o controle de requisições e o despacho HTTP e também pelo controle de sessão do usuário na aplicação. O CherryPy, diferentemente dos *web frameworks fullstack* TurboGears e Django, não provê ferramentas para a tradução de *templates*, para o ORM ou para a codificação de formulários, mas essas ferramentas podem ser facilmente adicionadas à sua estrutura.

Um dos principais motivos para a escolha do *web framework* CherryPy neste trabalho é o fato de ele ser um *framework* não intrusivo, no sentido que não afeta a escolha de seus

usuários, fornecendo um conjunto de ferramentas para qualquer desenvolvedor sem fazer qualquer suposição sobre o modo que ele ou ela optará para usá-las. (HELLEGOUARCH, 2007).

Além disso, o CherryPy possui um servidor de aplicação pronto para a implantação do *website* desenvolvido, simplificando ainda mais a criação ágil de aplicações *web*. Também coube ao CherryPy prover o serviço HTTP utilizado pelo *framework* para receber os dados espaço-temporais. Dessa forma, tem-se o desacoplamento da funcionalidade de recebimento de dados de localização com as funções do servidor *web*, como o tratamento de requisições de navegadores, a geração e o envio de páginas HTML e a análise de dados espaço-temporais.

3.2.1.3 *Templating*

Para o *templating* foi utilizada a biblioteca Genshi. Ela é responsável pela geração dinâmica das páginas requisitadas pelo navegador *web*, populando e transformando as estruturas XML, que possuem variáveis e *tags* Python, em páginas HTML. Os *templates*, que são considerados moldes de páginas *web*, separam a lógica computacional da interface visualizada pelo usuário e possuem grande possibilidade de reúso. Um exemplo de *template* pode ser visto no Apêndice D.

3.2.2 *WebGeoFramework*

Na parte central do diagrama é apresentada a camada lógica que representa o *framework* desenvolvido neste trabalho. O seu desenvolvimento priorizou uma arquitetura otimizada e inteligível, tendo como base o consagrado modelo *model-view-controller* (MVC), que separa a aplicação em três partes, quais sejam, como o próprio nome explicita: *model* (gerencia o domínio de informação), *view* (interface) e *controller* (lógica de funcionamento). No caso de aplicações *web*, a *view* é formada principalmente pelos *templates* que irão gerar as páginas HTML exibidas para o usuário.

3.2.2.1 *Model*

O *model* é formado pelas classes persistidas pelo *framework* com o uso do mapeamento objeto-relacional. Essas classes podem ser visualizadas no diagrama apresentado na Figura 2, sendo elas: *Point*, *Area*, *Device*, *Function*, *DevicePointFunction*, *DeviceAreaFunction*, *PointEvent* e *AreaEvent*. E as tabelas geradas por essas classes são, respectivamente: *areas*, *devices*, *functions*, *device_point_functions*, *device_area_functions*, *point_events* e *area_events*. Já um exemplo de classe persistida pode ser visto no Apêndice C.

O relacionamento entre as tabelas do banco de dados ocorre do mesmo modo que entre as classes persistidas. Assim sendo, o esquema lógico do banco de dados, apresentado na Figura 3, pode facilitar a compreensão do funcionamento das classes de mapeamento.

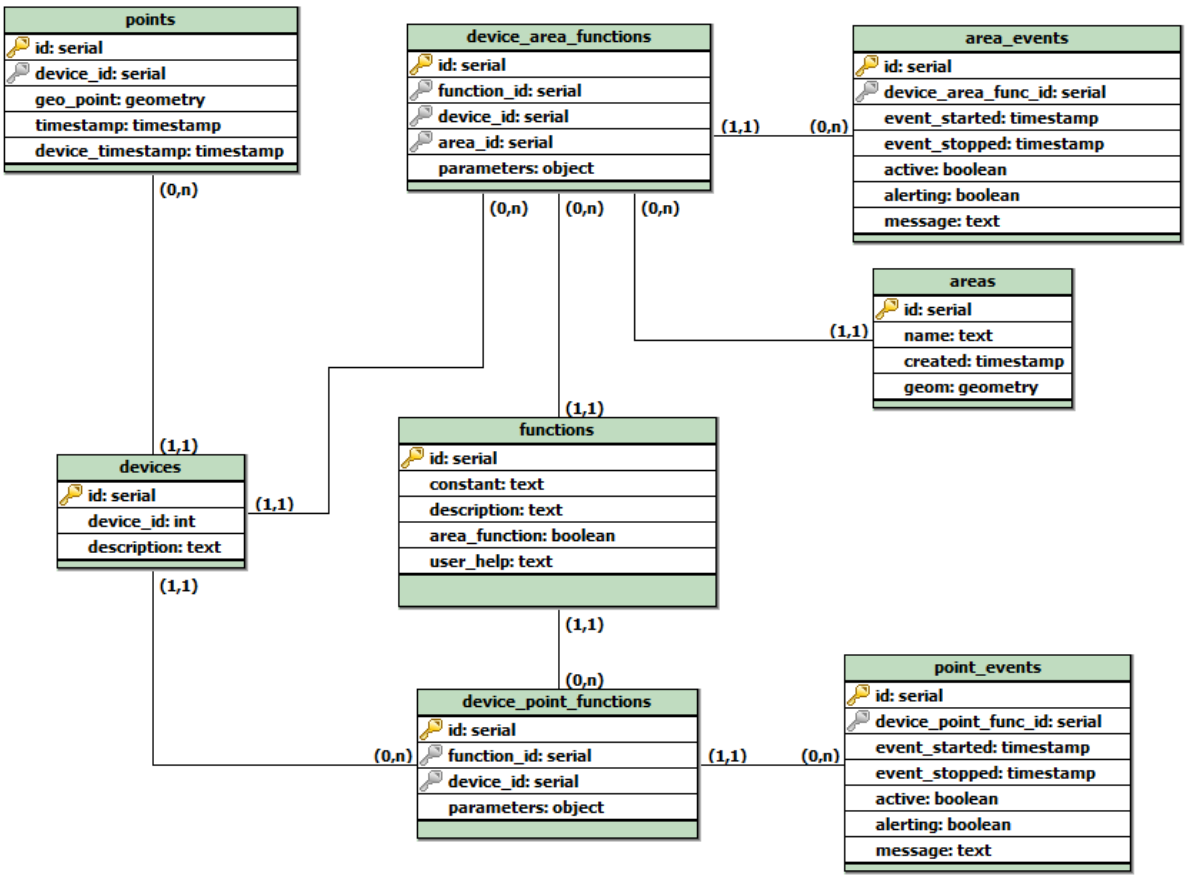

Figura 3 – Esquema lógico do modelo de dados persistido pelo *framework*
Fonte: elaborada pelo autor (2012)

A tabela *points* possui cinco colunas: identificador único, ponteiro para o *Device* associado, carimbo de tempo da inserção no banco de dados, dado espacial e carimbo de tempo gerado pelo dispositivo de coleta. Esses atributos fazem com que o registro dessa

tabela seja georreferenciado e associado ao dispositivo que o gerou, bem como possua referências temporais do momento em que foi inserido no sistema e também do momento em que foi coletado. Pode-se dizer, portanto, que um registro da tabela *points* é um dado espaço-temporal.

Os registros da tabela *areas*, diferentemente dos registros de *points*, não são criados por dispositivos de coleta, devendo ser inseridos manualmente pelo desenvolvedor. Seus atributos são: identificador único, nome da área, carimbo de tempo da criação e ponto geométrico de tipo polígono. Um registro dessa tabela representará áreas georreferenciadas que poderão ser utilizadas para a identificação de eventos semânticos.

Um registro da tabela *devices* representa o dispositivo de coleta de dados geográficos. Sua importância está no fato de ele prover a identificação do gerador do dado espaço-temporal. Possui os seguintes atributos: identificador, identificador do dispositivo (um valor único fornecido por este) e descrição do dispositivo.

A tabela *functions* representa a função espaço-temporal que busca e identifica eventos semânticos. Uma condição para que o *framework* possa aplicar as funções espaço-temporais é a criação de instâncias de *functions* que as representem. Devido ao fato de o *framework* possuir funções espaço-temporais prontas para o uso, ele disponibiliza também os registros da tabela *functions* que as representam, isto é, as instâncias de SPEED, INTERSECTS_AREA e NOT_INTERSECTS_AREA. As colunas dessa tabela são: um identificador único, o nome dado à função espaço-temporal (chamado de constante), a descrição da função, a ajuda ao usuário e um atributo para identificar se a função será aplicada a áreas ou não.

A tabela *device_point_functions* representa o relacionamento entre *devices* e *functions*, por isso, possui referências à função aplicada e ao *device*. Também possui uma coluna com os parâmetros que devem servir de entrada na função espaço-temporal. Por exemplo, no caso da função velocidade, os parâmetros poderiam ser "*max_speed=80;min_speed=10;*".

A tabela *device_area_functions* representa o relacionamento entre *devices* e *functions* e também se relaciona com a tabela *areas*. Assim sendo, possui referências à função aplicada, ao *device* e à *area*. Ela também possui uma coluna com os parâmetros que devem servir de entrada para a função espaço-temporal.

As tabelas *point_events* e *area_events* armazenam os eventos semânticos identificados. Suas colunas são: identificador, carimbos de tempo de início e fim do evento, mensagem descritiva do evento e atributos para informar se o evento está ativo e se está emitindo alertas. Além disso, a tabela *point_events* tem uma referência para a tabela *device_point_functions* e,

do mesmo modo, a tabela *area_events* tem uma referência para a tabela *device_area_functions*.

3.2.2.2 *View*

A segunda parte do modelo MVC, no qual está baseada a arquitetura do *framework* proposto, é chamada *view* e está relacionada à interface. Consideramos, neste trabalho, que a *view* do *framework* encontra-se no lado do cliente e no lado do servidor. Por exemplo, quando o usuário faz uma requisição no navegador *web* para visualizar uma página com um mapa do Google™ Maps, a estrutura da página é criada no lado do servidor e o mapa é carregado do lado do cliente.

Os componentes da *view* do lado do servidor foram criados utilizando a linguagem de marcação HTML e a biblioteca Genshi, sendo que graças a esta foi possível criar *templates* de páginas que podem ser facilmente reutilizados, entre os quais estão as páginas para a criação de formulários e de tabelas e para visualização de mapa. Dessa forma, o desenvolvedor que estenderá o *framework* poderá criar as páginas da aplicação sem a necessidade de codificação de interface.

Os principais *templates* podem ser visualizados no diagrama da Figura 2, apresentada na página 25. São eles: (1) *admin*, responsável pela criação do menu superior de administração; (2) *device_area_function* e *device_point_function*, responsáveis pela criação das páginas de parametrização de funções espaço-temporais; (3) *maps*, responsável pela criação da página de visualização de mapa que contém o componente Google™ Maps; (4) *jqgrid*, trata-se de um *template* para criação de página com componente para visualização tabular de dados, como é o caso do histórico de registros de eventos semânticos encontrados pelo *framework*; (5) *form* e *table*, que oferecem um molde simplificado para a criação de formulários e tabelas; e (6) *header* e *layout*, que são utilizados em todos os outros *templates*, pois definem características estruturais e de estilo da exibição das páginas. A utilização dos *templates* e das páginas criadas por eles será explicitada no item 3.3, referente ao funcionamento do *framework*.

Muitos dos componentes que aparecem nas páginas carregadas pelo navegador – como menus, *links*, botões e caixas de seleção – são criados dinamicamente pelo *framework* a partir da leitura de um arquivo de configuração de fácil entendimento. Assim, para desabilitar a exibição de menus e de páginas referentes à manipulação de áreas geográficas, por exemplo,

basta editar uma linha do arquivo de configuração.

O *template* mais importante do *framework* é o *maps*, responsável pela exibição do mapa do Google™ Maps. Por meio desse *template* são exibidos, em uma página específica, os eventos semânticos encontrados, os dados espaço-temporais, as áreas geográficas e os dispositivos móveis que podem ser monitorados.

Do lado do servidor é feito o despacho dos pontos geográficos para o navegador *web* do cliente assim que eles são recebidos, e o mesmo ocorre com os eventos semânticos identificados. Já do lado do cliente, os dados recebidos do servidor são inseridos no mapa ou exibidos na tela como mensagens de alerta. No *template maps*, a tecnologia utilizada para o recebimento de dados provenientes do servidor é a de *Server-Sent Events*. Com o uso dessa tecnologia não há necessidade de o navegador fazer requisições AJAX a cada intervalo de tempo para verificar se o servidor possui mais dados para envio. Assim, os dados são enviados pelo servidor para o cliente no momento em que estão disponíveis. Esse comportamento mais ágil é esperado em aplicações de monitoramento em tempo real de objetos móveis.

3.2.2.2 *Controller*

Em relação à terceira parte do MVC, a responsável pela lógica de funcionamento, temos os *controllers* do *framework*, que são apresentados no pacote *Controllers* da camada *WebGeoFramework* da Figura 2.

Eles podem ser divididos quanto ao domínio de suas funcionalidades em: (1) controladores responsáveis pelo recebimento de requisições *web* e despacho de páginas HTML, como *AdminController, FunctionController, DeviceController* e *MapsController*, os quais geram as páginas para cadastros, parametrização e visualização de funções espaço-temporais e também a página para visualização dos dados espaço-temporais em mapa; (2) controladores para a manipulação e a análise de dados espaço-temporais, que são os responsáveis pela aplicação das funções espaço-temporais e pela geração de eventos semânticos, como *GeoControllers* e *GeoFunction*; (3) controladores responsáveis pelo recebimento e tratamento das requisições *web* e dos pontos espaço-temporais recebidos, chamados de *WebAppServer* e *HTTPServer*; e (4) controlador-raiz da aplicação *web*, no qual são associados os controladores-filhos de exibição de páginas HTML, como é o caso de

FunctionController. Esse controlador-raiz é disponibilizado nas aplicações que estendem o *framework* para que elas possuam sua própria hierarquia de páginas quando não for necessário utilizar os controladores de despacho de páginas *web* disponibilizados pelo *framework*.

Para explicar o funcionamento do *framework* em relação à análise de dados espaço-temporais, é necessário descrever o código de alguns controladores. Porém, devido à dificuldade de traduzir o código-fonte para a linguagem natural sem produzir ambiguidade e à facilidade de delongar-se exageradamente nessa explicação, optamos por exprimir o funcionamento exemplificando, nos parágrafos seguintes, uma aplicação que estenda o *framework* mesclando vários níveis de abstração.

Quando um dispositivo móvel envia dados espaço-temporais para a aplicação, esses dados são recebidos pelo controlador *HTTPServer*, cuja responsabilidade é criar uma instância da classe *Point* e armazená-la no banco de dados. Também é responsabilidade desse controlador enviar a instância para um controlador *GeoController,* que verifica se alguma função espaço-temporal deve ser aplicada, e para o controlador que exibe o ponto geográfico no navegador *web*.

O controlador que recebe o objeto *Point* verifica, no banco de dados, se deve executar alguma função espaço-temporal para o ponto recebido. Caso isso seja necessário, ele aplicará a função passando os parâmetros armazenados nas instâncias de *DevicePointFunction* ou de *DeviceAreaFunction*. É responsabilidade desse controlador armazenar no banco de dados os eventos semânticos identificados durante as execuções das funções espaço-temporais e enviá-los para o controlador *MapsController*, que exibe os eventos no navegador *web*.

As funções espaço-temporais desenvolvidas nas extensões do *framework* deverão implementar o controlador *GeoFunction* desenvolvendo o código para o método *apply_function*, o qual receberá os parâmetros cadastrados que devem nortear a busca e a identificação de eventos semânticos. No caso da função já implementada SPEED, por exemplo, poderá receber os parâmetros "*max_speed"* e *"min_speed*;". Códigos-fonte de eventos semânticos podem ser visualizados no Apêndice A, enquanto códigos-fonte de funções espaço-temporais podem ser vistos no Apêndice B.

Funções espaço-temporais serão as mais recorrentes extensões nas aplicações que utilizarão o *framework* e é nelas que fica claro que o *framework* é o principal responsável pelo controle do fluxo de execução, pois é ele que chamará a função espaço-temporal implementada.

Para que o *framework* encontre o controlador implementado, é necessário que este seja

cadastrado no banco de dados como uma instância de *Function* associada a um *Device* por meio de uma instância de *DevicePointFunction* ou *DeviceAreaFunction*. Também é necessário importar a função em um arquivo criado pelo *framework* na aplicação que o estende. Quando a função espaço-temporal identificar um evento, deve gerar uma exceção de um tipo suportado pelo *framework*.

O controlador *MapsController*, que exibe pontos e eventos para o usuário por meio de um navegador *web*, utiliza uma funcionalidade do HTML5 chamada *Server-Sent Events*. Essa funcionalidade, suportada pela maioria dos navegadores mais utilizados, faz com que o servidor possa mandar dados para o usuário quando estes estiverem prontos para o envio, e não somente quando o navegador os requisitar. Dessa forma, é possível enviar pontos geográficos assim que eles forem recebidos pelo servidor, fornecendo para o usuário uma visualização em tempo real dos dispositivos móveis e dos eventos semânticos.

3.2.3 AppFramework

A aplicação *web* fornecida pelo *framework* pode ser visualizada na camada lógica *AppFramework*, localizada na parte superior da Figura 2. Quando o *framework* é instalado, a aplicação *AppFramework* está pronta para uso, provendo a configuração de *Devices* e de funções espaço-temporais e também a visualização de dados espaço-temporais e eventos semânticos.

Essa aplicação foi desenvolvida para criar uma base genérica que pudesse ser utilizada por aplicações que manipulam dados espaço-temporais, focando principalmente na configuração dos componentes necessários para a análise de dados espaço-temporais. Para isso, utilizou-se uma arquitetura que suporta a extração do *framework* de seu código e que mantém a possibilidade de usar a aplicação *AppFramework* em aplicações que estenderão o *framework* desenvolvido neste trabalho.

Esse *framework* engloba também a aplicação descrita acima, por esse motivo a extensão do *framework* costuma estender também a aplicação, visto que os *templates* e os controladores criados para ela podem ser utilizados na extensão.

3.3 FUNCIONAMENTO DO *FRAMEWORK*

A seguir são apresentadas as interfaces *web* para cadastro de dados necessários para o funcionamento do *framework*, bem como as interfaces comumente utilizadas depois de terminados os procedimentos de configuração e cadastro.

3.3.1 Cadastros e configuração

O *framework* desenvolvido é do tipo *white-box*, portanto, seu código-fonte pode ser analisado pelo desenvolvedor. Essa análise é, inclusive, recomendada, pois pode facilitar o desenvolvimento de aplicações que o estenderão.

Para estender o *framework*, desenvolvendo outra aplicação, pode-se realizar os procedimentos exibidos na Figura 4, isto é: o *framework* deve ser instalado (procedimento que instala também as bibliotecas necessárias); a aplicação deve ser criada utilizando o *script* fornecido pelo *framework*; e o arquivo de configuração da aplicação deve ser editado para que possua os valores corretos para o endereço, o nome, o usuário e a senha da conexão com o banco de dados.

Após a conclusão dos procedimentos anteriormente descritos, a aplicação estará pronta para ser utilizada e acessada pelo navegador *web*. O próximo passo é o desenvolvimento da função espaço-temporal e, se aplicável, o desenvolvimento do código que representará o evento identificado por essa função espaço-temporal. Depois de desenvolvida, a função pode ser cadastrada no banco de dados por meio da interface *web* na página específica para esse fim. Os dispositivos móveis também devem ser cadastrados e associados à função espaço-temporal recém-criada. As associações entre funções e dispositivos devem possuir a parametrização da função espaço-temporal para o dispositivo associado.

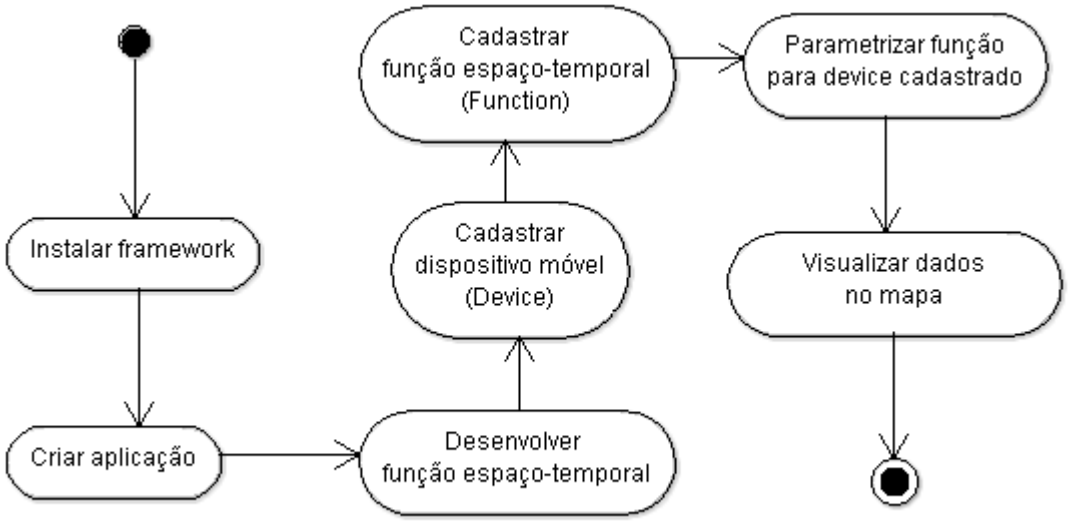

Figura 4 – Passo a passo para estender o *framework*
Fonte: elaborada pelo autor (2012)

Quando instalado, o próprio *framework* também pode ser utilizado como uma aplicação *web*, sem a necessidade de extensão. A Figura 5 exibe a página inicial dessa aplicação, criada a partir do *template root*, bem como a barra de menus superior, criada pelo *template admin* e que contém os *links* para os controles administrativos (como criação de funções espaço-temporais e de *devices*), para a visualização de eventos gerados por pontos ou áreas e para a visualização do mapa.

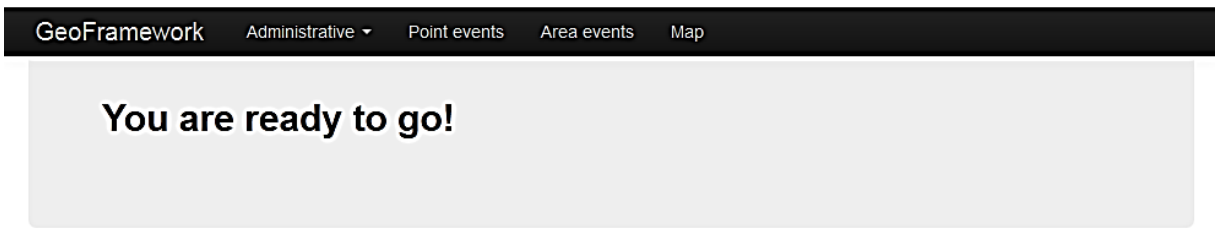

Figura 5 – Página inicial da aplicação *web* fornecida pelo *framework*
Fonte: elaborada pelo autor (2012)

Na Figura 6, a seguir, pode-se ver os itens do menu administrativo na interface *web*. No primeiro item, chamado *Function*, é possível visualizar, criar e editar funções espaço-temporais. Já clicando em *Device*, que é o segundo item desse menu, acessa-se uma página com todos os dispositivos móveis cadastrados e também é possível criar, editar e remover os dispositivos. Por fim, os itens seguintes, que são as páginas *Point function parameters* e *Area*

function parameters, são utilizados para cadastrar os parâmetros a serem fornecidos para as funções espaço-temporais.

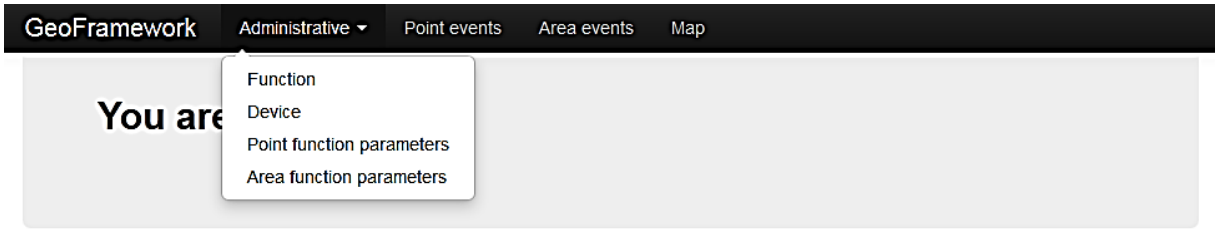

Figura 6 – Menu administrativo da aplicação *web* fornecida pelo *framework*
Fonte: elaborada pelo autor (2012)

O *framework* possui funções espaço-temporais já implementadas, como as funções de cálculo de velocidade e de intersecção de áreas. A página *Function*, exibida na Figura 7, é acessível a partir do menu administrativo e nela, as funções espaço-temporais, bem como os botões para criação, edição e deleção de funções são apresentados.

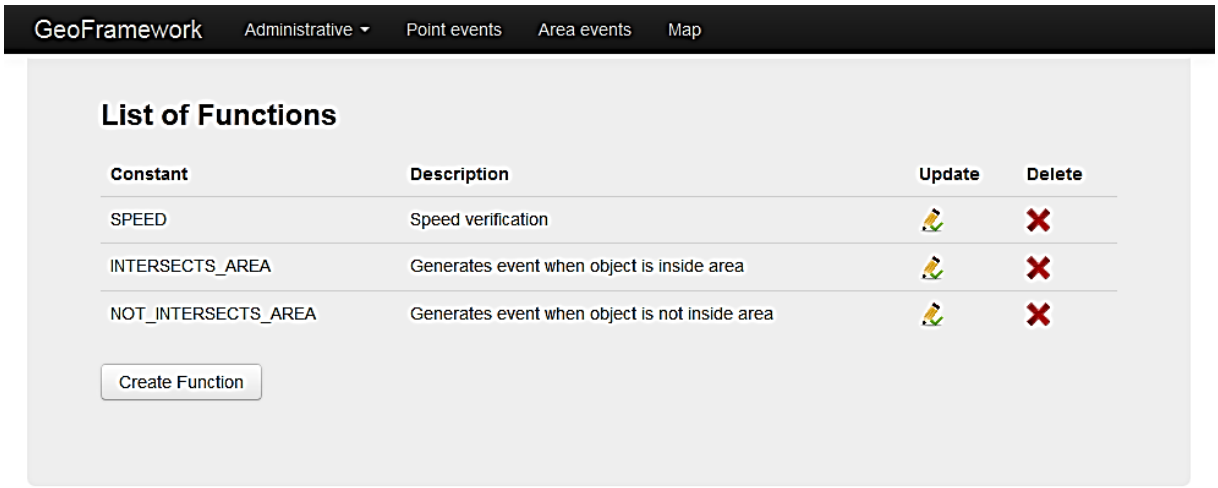

Figura 7 – Página da aplicação *web* fornecida pelo *framework* para visualização de funções espaço-temporais (*Functions*)
Fonte: elaborada pelo autor (2012)

A página para criar a função espaço-temporal no banco de dados é apresentada na Figura 8. A partir dessa página é esperado que o desenvolvedor de funções espaço-temporais cadastre, além dos atributos básicos (Constante e Descrição), um texto de ajuda ao usuário (Chaves de ajuda para o usuário), o qual cadastrará os parâmetros das funções. Esse texto deverá conter uma breve descrição dos parâmetros aceitos pela função espaço-temporal, de modo que o usuário não necessitará olhar o código-fonte da função espaço-temporal para efetuar cadastros *web*.

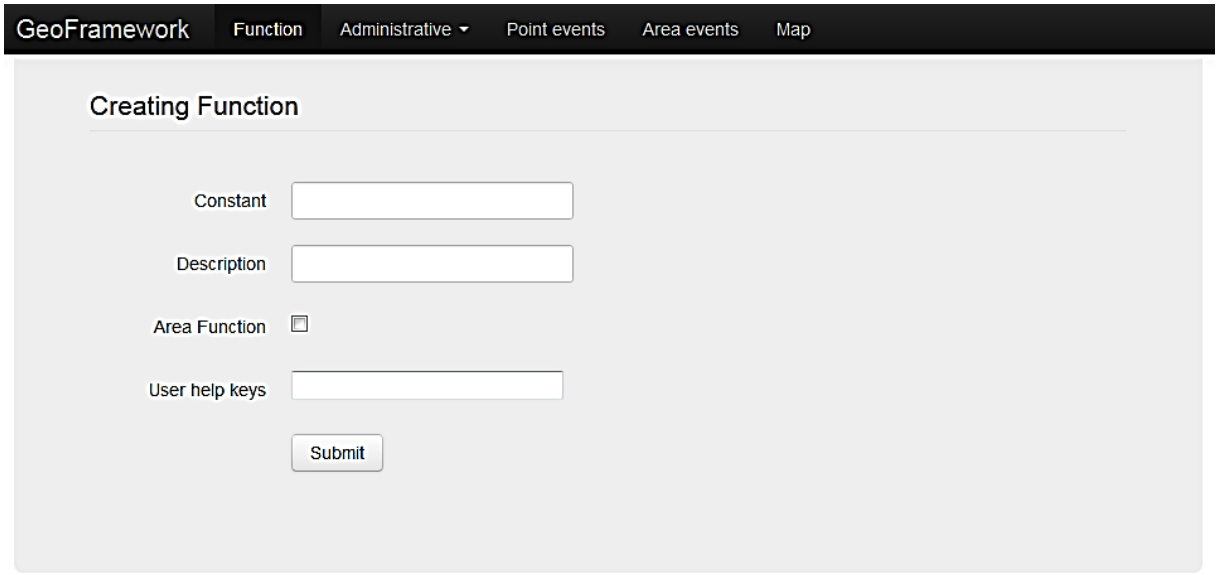

Figura 8 – Página da aplicação *web* fornecida pelo *framework* para criação de funções espaço-temporais (*Functions*)
Fonte: elaborada pelo autor (2012)

Para cadastrar o texto de ajuda ao usuário, a aplicação *web* fornece uma interface para texto em formato rico, isto é, permite que texto seja formatado com tabulação, cores, fontes e outros recursos de formatação. Essa interface, exibida na Figura 9, pode ser acessada pelo usuário a partir do campo de cadastro de ajuda ao usuário.

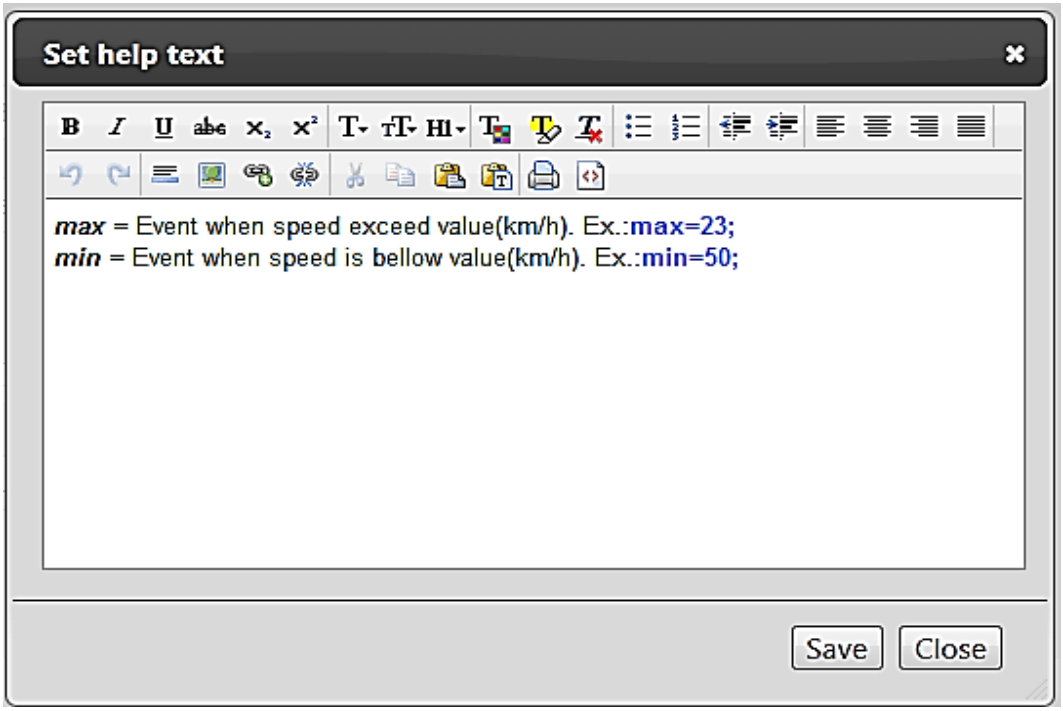

Figura 9 – Página da aplicação *web* fornecida pelo *framework* para cadastro de ajuda ao usuário
Fonte: elaborada pelo autor (2012)

A Figura 10, a seguir, demonstra a edição de uma função espaço-temporal. Essa página é acessível pela página administrativa *Function* por meio de um ícone indicativo no registro da função espaço-temporal. Nessa figura, podemos ver que o conteúdo do campo de ajuda ao usuário possui *tags* de formatação de texto, que foram geradas automaticamente pelo componente exibido na Figura 9, apresentada anteriormente.

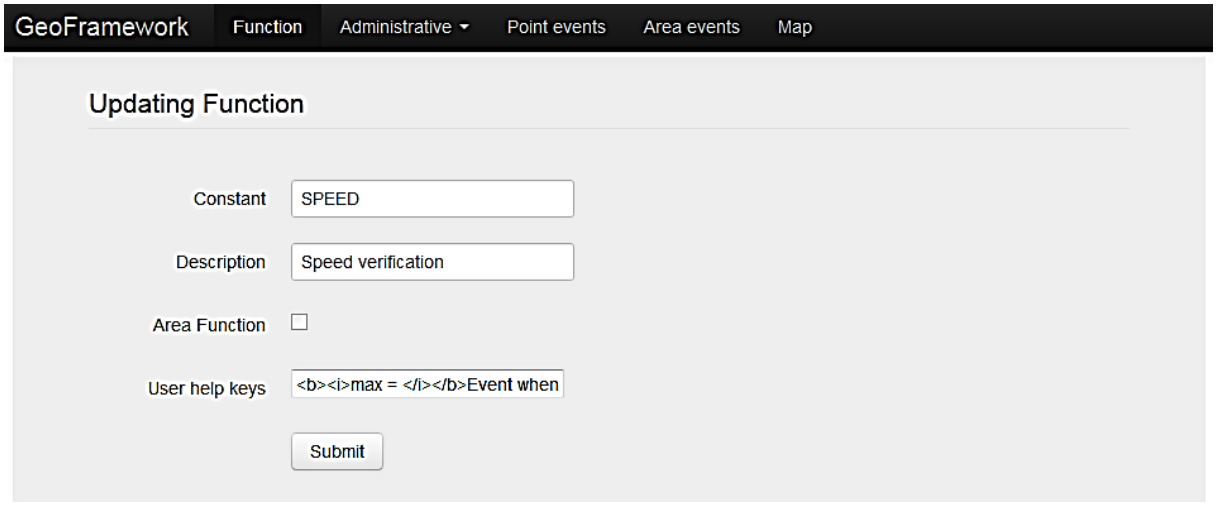

Figura 10 – Página da aplicação *web* fornecida pelo *framework* para edição de funções espaço-temporais
Fonte: elaborada pelo autor (2012)

Na Figura 11 é possível visualizar a página administrativa *Device*. Da mesma forma que a página *Function*, essa página apresenta a listagem dos dispositivos móveis já cadastrados, bem como serve de acesso para a criação, a edição e a remoção de dispositivos.

Essa página também lista as funções espaço-temporais associadas ao *Device*. Por exemplo, a análise da Figura 11 nos permite constatar que o dispositivo *Celular pedestre* possui a função espaço-temporal SPEED aplicada.

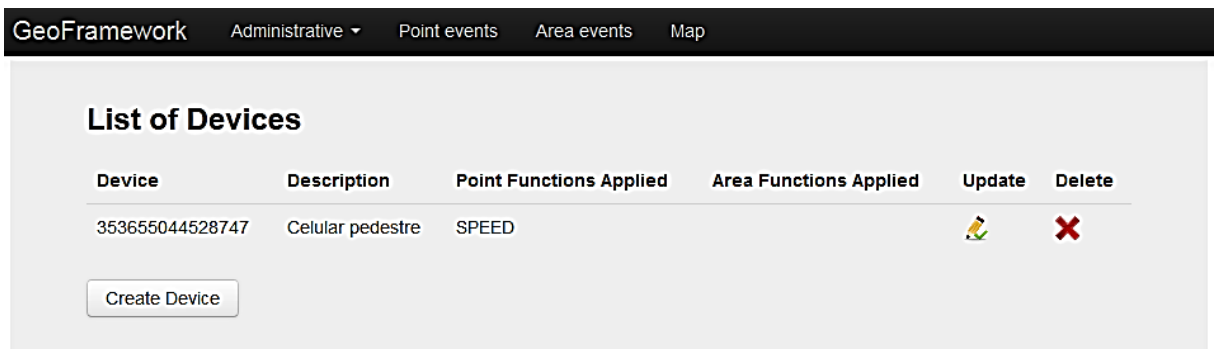

Figura 11 – Página da aplicação *web* fornecida pelo *framework* para exibição de dispositivos móveis (*Devices*)
Fonte: elaborada pelo autor (2012)

Para a criação de dispositivos móveis é disponibilizada a página *Creating Device*, exibida na Figura 12, que pode ser acessada a partir da página administrativa *Device*, como pode ser visto na Figura 11, já apresentada. Na página *Creating Device*, o atributo de nome *Device* é o identificador único do dispositivo, já o atributo de descrição – chamado *Description* – é utilizado para identificar o dispositivo no sistema sem restrição de unicidade.

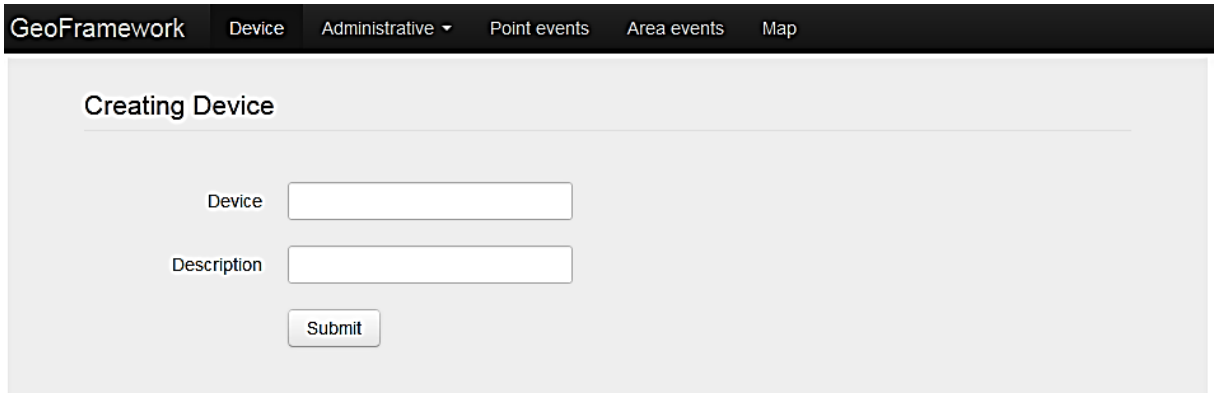

Figura 12 – Página da aplicação *web* fornecida pelo *framework* para criação de dispositivos móveis
Fonte: elaborada pelo autor (2012)

Quando executadas pelo *framework*, as funções espaço-temporais recebem parâmetros que norteiam seus funcionamentos. A página apresentada na Figura 13, que pode ser visualizada acessando o item *Point function parameters* no menu administrativo, demonstra a edição dos parâmetros que serão passados para a função espaço-temporal SPEED quando o servidor receber dados do dispositivo-móvel *Celular pedestre*.

É possível notar três ícones ao lado do campo de inserção de parâmetros: o primeiro guarda os dados cadastrados no banco de dados; o segundo abre a janela *Populating config parameters*, na qual é apresentada a ajuda ao usuário cadastrada na função espaço-temporal, como pode ser visto na Figura 9; e o último ícone remove a parametrização da função espaço-temporal.

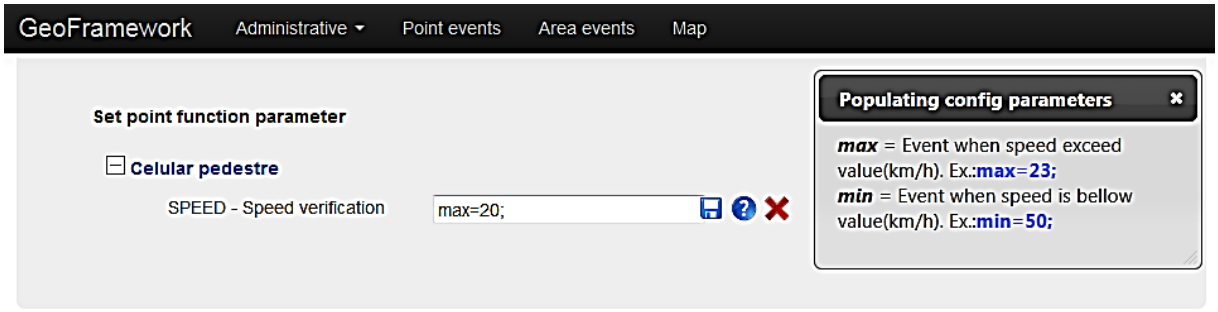

Figura 13 – Página da aplicação *web* fornecida pelo *framework* para parametrização de funções espaço-temporais
Fonte: elaborada pelo autor (2012)

3.3.2 Análise de dados e de resultados

Eventos semânticos gerados por funções espaço-temporais são salvos no banco de dados e podem ser visualizados nas páginas *Area events* e *Point events*, acessíveis a partir da barra de menus superior. A página *Point events*, que pode ser visualizada na Figura 14, a seguir, apresenta os últimos eventos semânticos encontrados, indicando o dispositivo móvel gerador do evento, o tipo do evento gerado, as datas de início e fim do evento, a mensagem gerada pelo evento e um indicador, que define se o evento está ativo ou não.

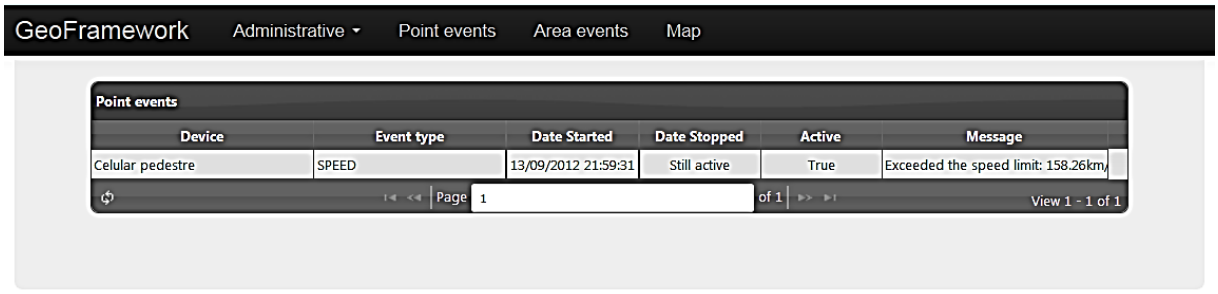

Figura 14 – Página da aplicação *web* fornecida pelo *framework* para visualização de eventos semânticos gerados
Fonte: elaborada pelo autor (2012)

Para o monitoramento de dispositivos móveis e dos eventos semânticos gerados por eles, a aplicação *web* fornecida pelo *framework* disponibiliza a página apresentada na Figura 15, acessível a partir do item *Map*, o qual está localizado na barra de menus superior. Nessa página é possível escolher quais dispositivos móveis deverão ser monitorados e quais áreas geográficas cadastradas deverão aparecer, enriquecendo a visualização para o usuário.

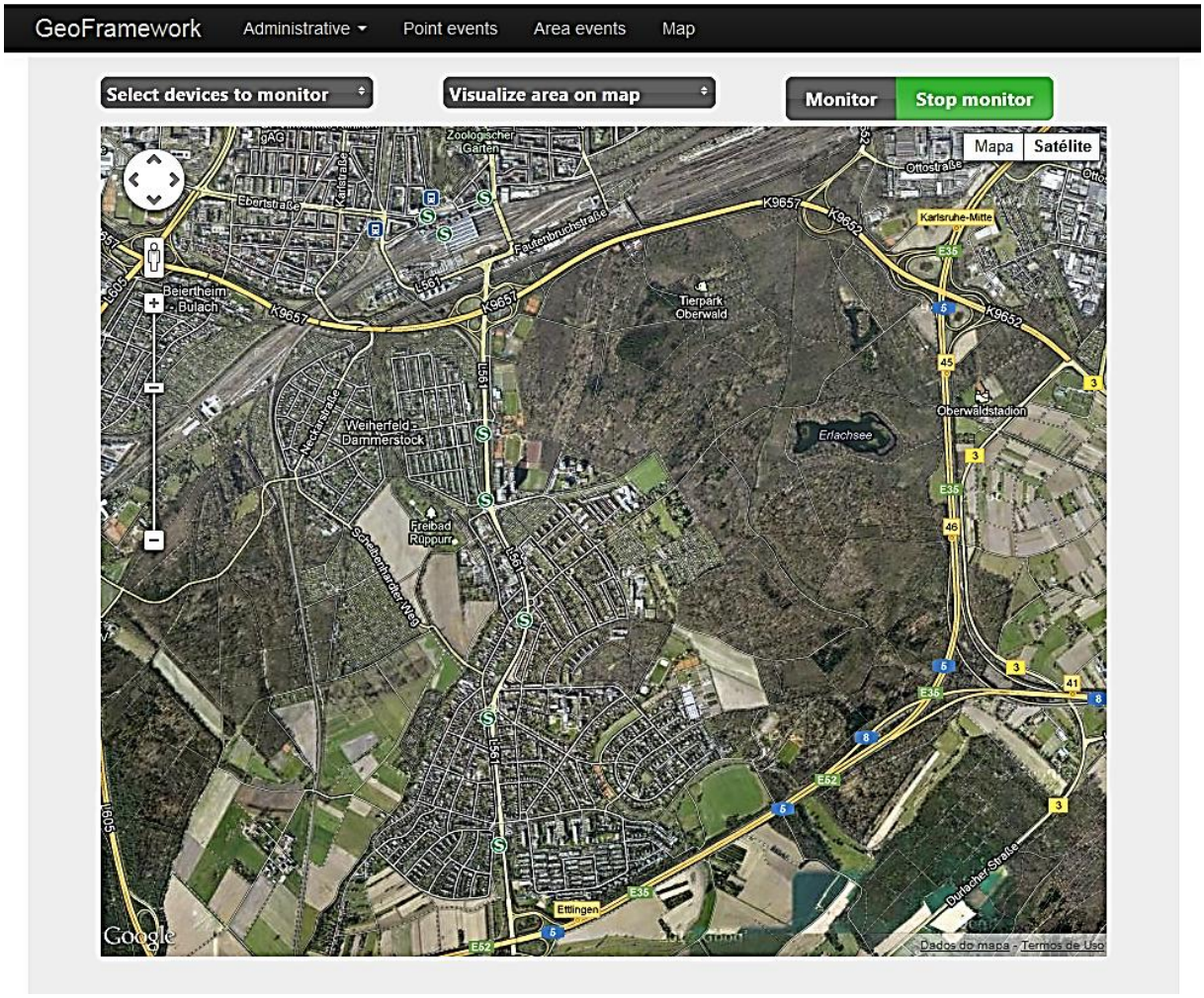

Figura 15 – Página da aplicação *web* fornecida pelo *framework* com mapa para monitoramento de dispositivos móveis e de eventos semânticos
Fonte: elaborada pelo autor (2012)

A aplicação *web* fornecida pelo *framework* foi desenvolvida para facilitar a configuração inicial do sistema para o desenvolvedor que for estender o *framework*. Dessa forma, o desenvolvedor poderá utilizar as funções do menu administrativo para o cadastro de funções, de dispositivos e de parâmetros e então retirar a permissão de acesso do usuário final ao menu de administração, permitindo o acesso deste somente às listas de eventos semânticos encontrados (Figura 14) e ao mapa para monitoramento de dispositivos (Figura 15).

Quando os procedimentos de cadastro e configuração do *framework* (descritos na seção 3.3.1) forem completados e a aplicação estiver em funcionamento, os dados espaço-temporais gerados pelos dispositivos móveis cadastrados serão enviados para as funções espaço-temporais associadas e poderão gerar eventos. Desse modo, o *framework* reconhece a função espaço-temporal criada na aplicação que o estendeu.

Por fim, os eventos gerados pelas funções espaço-temporais e os pontos recebidos podem ser vistos na página que possui o mapa (Figura 15), bastando selecionar os dispositivos que se deseja monitorar e, em seguida, clicar no botão que ativa o monitoramento dos dispositivos.

A Figura 16 apresenta um diagrama de atividades do procedimento de análise de dados espaço-temporais no *framework*. As atividades que compõem este diagrama serão brevemente descritas no texto seguinte com exemplos para facilitar o entendimento.

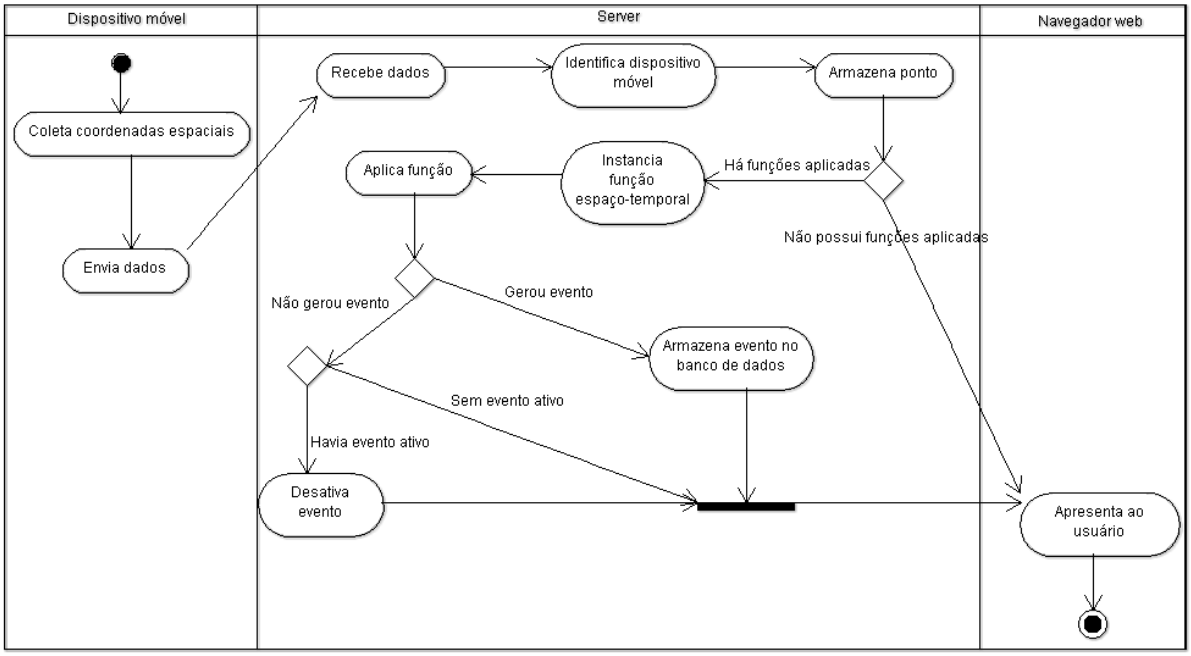

Figura 16 – Diagrama de atividades da análise de dados espaço-temporais
Fonte: elaborada pelo autor (2012)

Como ilustra a Figura 16, os dados espaço-temporais são coletados pelo dispositivo móvel, que os envia com um identificador do dispositivo e com um carimbo de tempo da coleta para a aplicação que estende o *framework*. Ao receber esses dados, a aplicação os armazena no banco de dados, chamando-os de ponto (*Point*). Após o armazenamento, é feita uma busca, no banco de dados, por parametrizações de funções espaço-temporais que estejam associadas ao dispositivo gerador do ponto. Toda parametrização encontrada é passada com o ponto para a função espaço-temporal, que então é aplicada.

A execução da função espaço-temporal objetiva a busca e a identificação de um fato desejado nos dados espaço-temporais. Quando o fato é identificado, ela deve gerar um evento

com as informações da ocorrência. Esse evento é armazenado no banco de dados e apresentado ao usuário *web* que estiver monitorando o dispositivo gerador do ponto.

A função espaço-temporal tem à sua disposição o acesso simplificado aos parâmetros que definem seu funcionamento. Além disso, pode acessar todos os dados e os operadores espaciais do banco de dados geográfico. Assim, a função espaço-temporal de SPEED (velocidade), que deve gerar eventos quando a velocidade máxima for ultrapassada, pode ser implementada da forma descrita a seguir.

Suponha-se que tenha ocorrido a parametrização da função SPEED associando-a com um dispositivo coletor e que o valor da velocidade máxima permitida tenha sido parametrizado como sendo 50km/h. Quando um ponto for recebido pela aplicação, esta aplicará a função espaço-temporal SPEED passando o valor máximo de velocidade permitido e os dois últimos pontos gerados pelo dispositivo móvel.

Na execução da função espaço-temporal SPEED é utilizada a operação espacial do banco de dados geográfico que calcula a distância entre os dados espaciais utilizando os dois últimos pontos gerados pelo dispositivo. Tendo a distância entre os dois últimos pontos, é necessário calcular a diferença de tempo entre eles, o que pode ser feito subtraindo os atributos de carimbo de tempo dos pontos. Com os valores de distância e de intervalo de tempo, é possível calcular a velocidade e então verificar se ela é maior que o máximo permitido, isto é, 50km/h. Caso seja, é necessário gerar um evento com as informações da ocorrência.

4 AVALIAÇÕES

Para validar os requisitos do projeto do *framework*, além da aplicação funcional contida nele, foram desenvolvidas três aplicações, que serão descritas a seguir.

4.1 – APLICAÇÃO 1 – IDENTIFICAÇÃO DE ACELERAÇÕES E MUDANÇAS DE DIREÇÃO BRUSCAS

Essa aplicação foi baseada nos métodos propostos por Carboni (2011) e buscou mostrar a viabilidade de uso do *framework* no meio acadêmico para a criação e o teste de algoritmos para a análise de comportamento de objetos móveis.

Os fins da identificação de acelerações bruscas, bem como da identificação de mudanças bruscas de direção, são, entre outros, reconhecer motoristas que dirigem de forma perigosa, podendo danificar ou colocar em risco a carga transportada; e identificar curvas perigosas, trechos sinuosos ou rodovias danificadas, que poderiam ter sinalização apropriada para prevenir acidentes.

Visto que o algoritmo proposto por Carboni (2011) utiliza apenas dados de pontos geográficos, a configuração da aplicação foi feita de forma a eliminar a interface para manipulação de áreas geográficas.

Como pode ser visto na Figura 17, na Aplicação 1 foram criados os algoritmos *SuddenDirectionChange* e *Acceleration*, que implementam o controlador *GeoFunction* do *framework*. Toda a interface utilizada nessa aplicação de avaliação é fornecida pela aplicação contida no *framework*, por isso o esforço de desenvolvimento se restringiu à criação dos algoritmos das funções espaço-temporais.

O controlador *Root*, apresentado no diagrama da Figura 17, é criado pelo *framework* e é responsável pela instanciação da hierarquia de páginas da aplicação. Devido ao fato da Aplicação não necessitar de modificações de interface, esse controlador permaneceu inalterado.

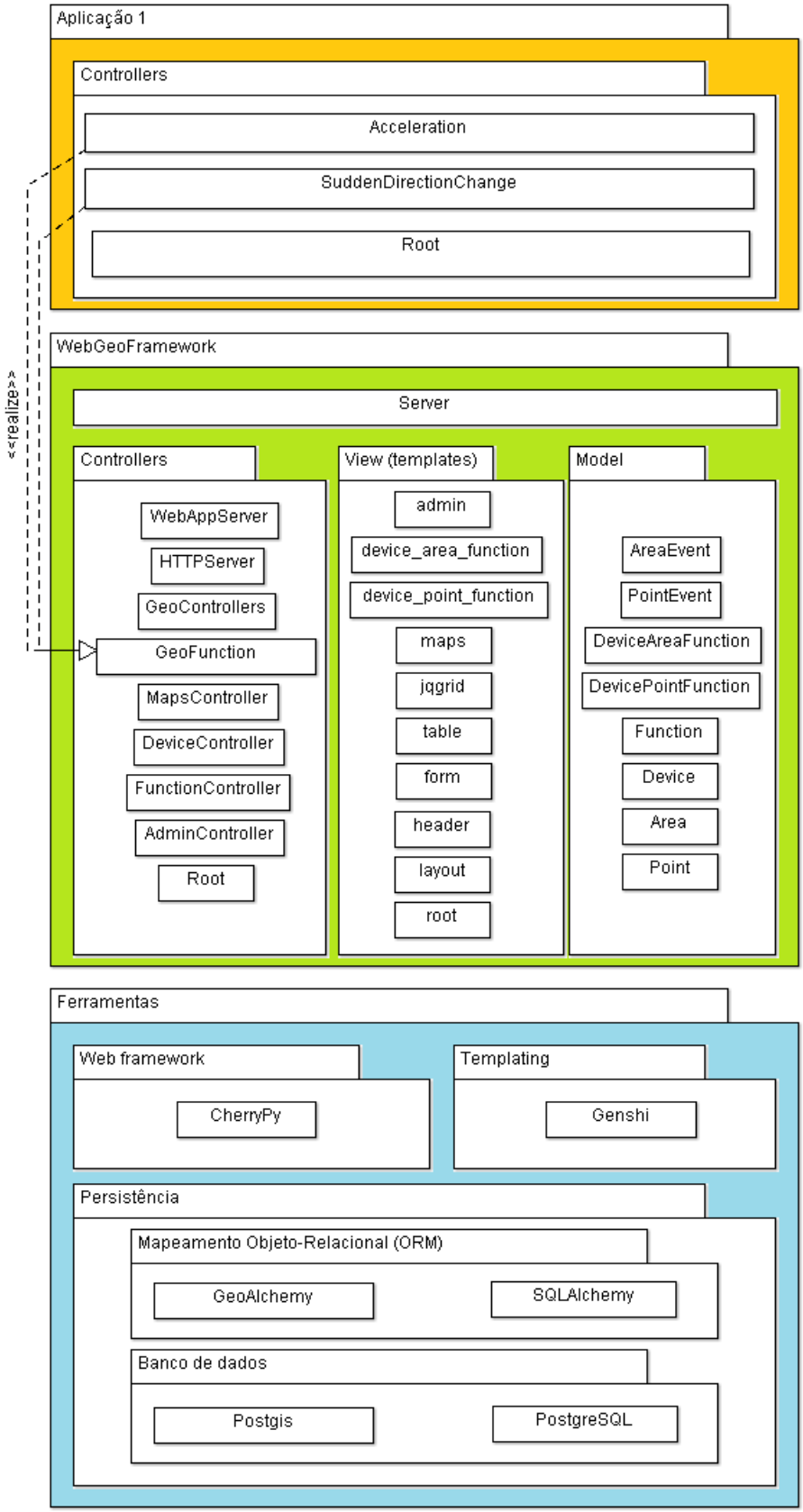

Figura 17 – Diagrama de módulos da Aplicação 1
Fonte: elaborada pelo autor (2012)

O algoritmo de identificação de aceleração e mudança de direção foi implementado em duas funções espaço-temporais, as quais utilizaram operadores espaciais do banco de dados para o cálculo de distância e de azimute. Com esses dados foi possível calcular a velocidade, essencial para o cálculo da aceleração, e a medida de abertura angular, necessária para o cálculo de mudança de direção. As funções implementadas são apresentadas na Figura 18, que exibe a página *web* de visualização de *Functions.*

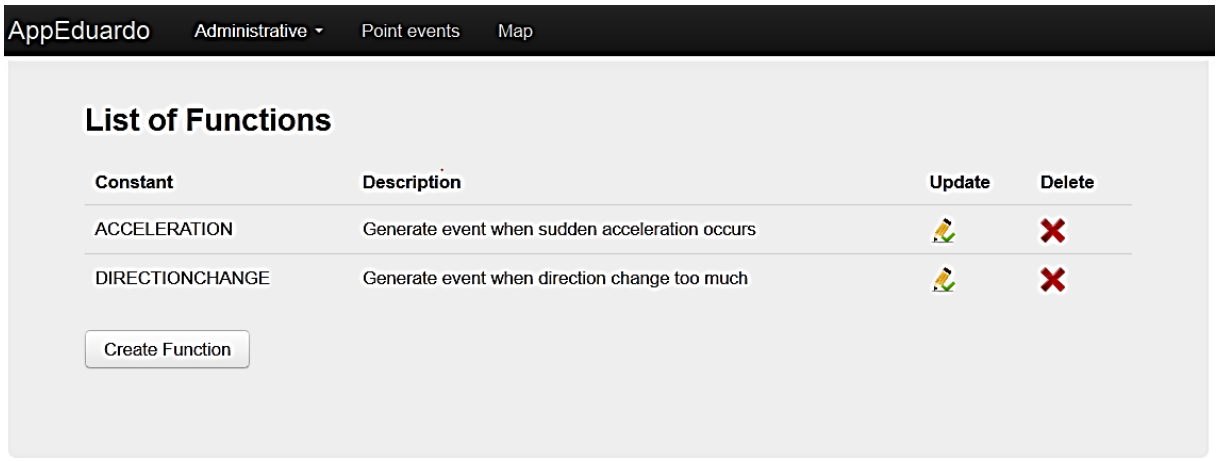

Figura 18 – Página de exibição de funções espaço-temporais da Aplicação 1
Fonte: elaborada pelo autor (2012)

O algoritmo previa parâmetros de entrada que definiam a identificação dos eventos: a aceleração e a mudança angular máxima aceitável. Esses parâmetros são facilmente inseridos na interface *web*, pois a arquitetura do *framework* prevê esse tipo de parametrização individual por dispositivo e função espaço-temporal.

Não foi necessário o desenvolvimento de novas interfaces *web*, visto que o acompanhamento em tempo real dos dispositivos móveis e dos eventos, gerados pelas funções espaço-temporais criadas, é suficiente para o uso proposto, que é o de validação de algoritmos de análise comportamental criados em ambiente acadêmico.

Na Figura 19, a seguir, é possível visualizar os eventos semânticos apresentados ao usuário da aplicação *web*. Esse usuário monitora um dispositivo-móvel que está efetuando acelerações e mudanças bruscas de direção. No lado direito da imagem é possível ver uma pilha de eventos semânticos gerados, sendo que a cor clara indica que o evento está inativo e a cor escura indica que o evento continua ativo.

Clicando em *Event Info* é possível visualizar uma janela como as apresentadas do lado esquerdo da figura, nas quais os dados dos eventos semânticos são apresentados.

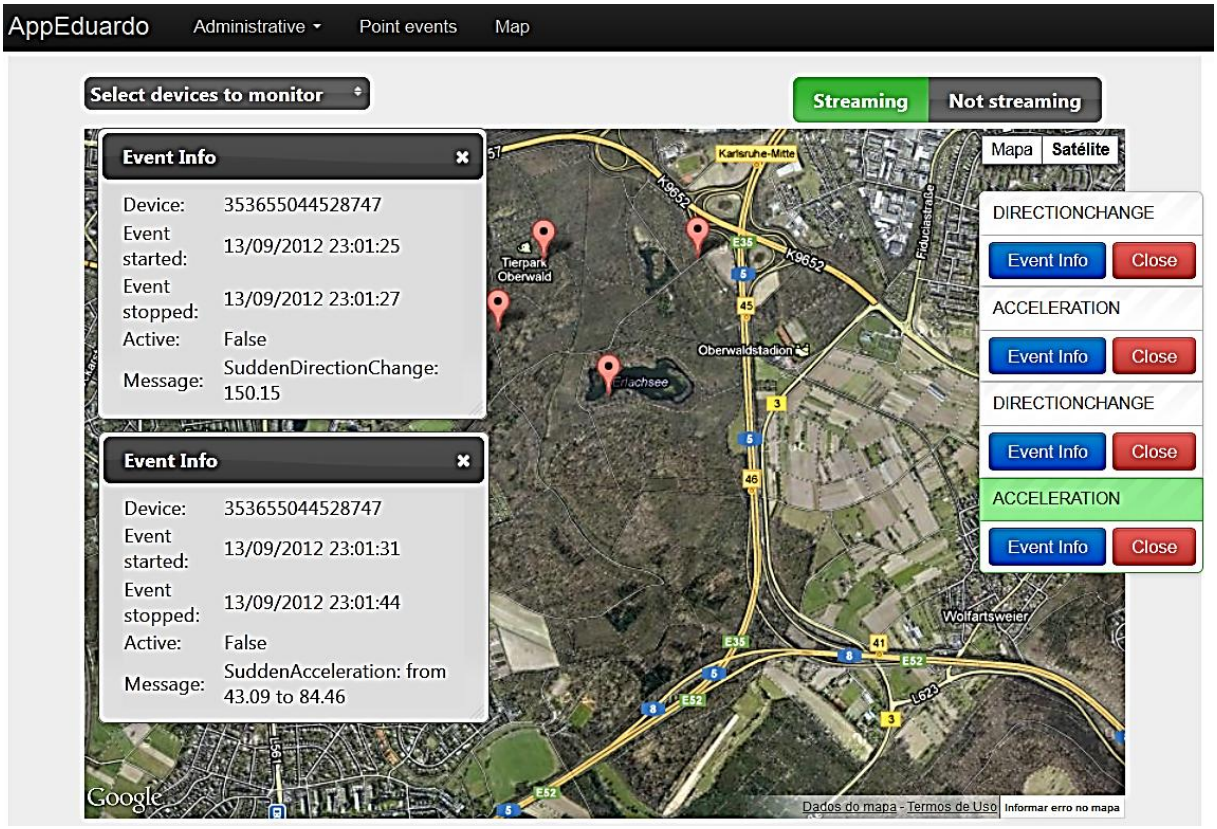

Figura 19 – Página com mapa utilizada pela Aplicação 1 para monitoramento de dispositivos móveis e de eventos semânticos
Fonte: elaborada pelo autor (2012)

Os eventos semânticos encontrados pela aplicação de avaliação podem ser visualizados na página *Point events*, como mostrado na Figura 20. A utilização dos dados apresentados nessa página para emissão de relatórios é simplificada, pois os mesmos são listados em estrutura tabular.

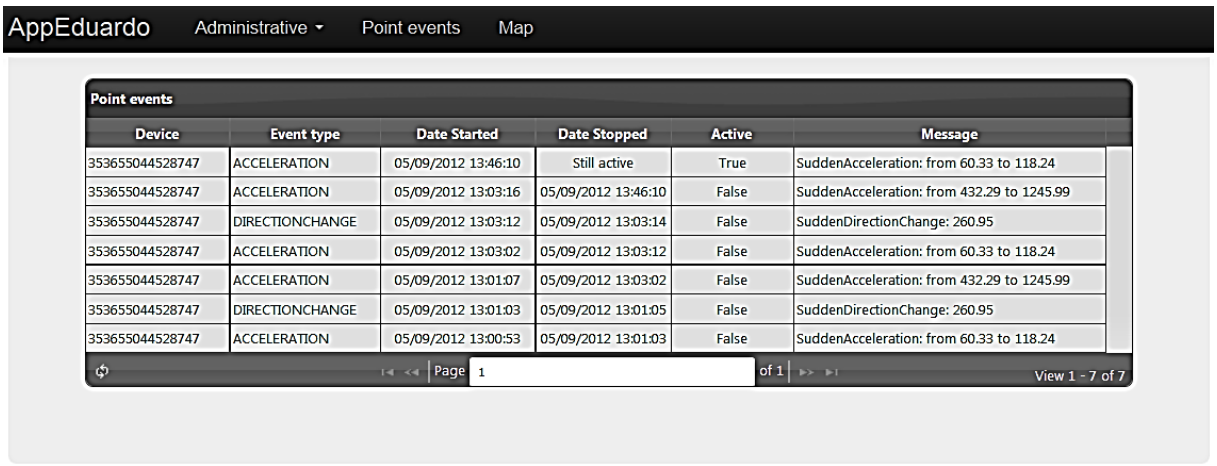

Figura 20 – Página de visualização dos eventos semânticos gerados na Aplicação 1
Fonte: elaborada pelo autor (2012)

Para o desenvolvimento da aplicação foram necessárias menos de 150 linhas de código, sendo que aproximadamente 110 foram utilizadas no desenvolvimento do algoritmo. Isso significa que o esforço para desenvolver a aplicação *web*, desconsiderando-se o esforço de implementação da função espaço-temporal, resumiu-se a 40 linhas de código.

Desta forma, comprova-se que as aplicações acadêmicas para a análise comportamental de objetos móveis podem utilizar o *framework*, tendo ganhos de produtividade e tempo de desenvolvimento.

4.2 – APLICAÇÃO 2 – CONTROLE DE FROTAS

Para validar o uso do *framework* por aplicações comerciais, em que comumente há a necessidade de interfaces com o usuário personalizadas, foi feita uma aplicação para o controle de frotas de caminhões. Essa aplicação oferece ao usuário uma página para cadastro de dados dos caminhões e respectivos motoristas e utiliza as funções espaço-temporais oferecidas pelo *framework*, que identificam os eventos de velocidade e de intersecção de áreas.

No diagrama de camadas apresentado na Figura 21, é possível visualizar quais foram as necessidades de desenvolvimento para a Aplicação 2. No pacote *Controllers* pode ser visualizada a utilização dos controladores do *framework MapsControllers* e *DeviceController* por meio de extensão. Ainda no pacote de controladores, houve pequena modificação do controlador *Root*, que agora instancia os controladores *MapsControllers* e *DeviceController* criados na Aplicação 2, e não os fornecidos pelo *framework.*

No pacote *Templates* houve maior esforço de desenvolvimento devido ao fato de a interface necessitar de várias modificações. Os *templates root, header* e *layout*, que são fornecidos pelo *framework* e definem as configurações genéricas de exibição para todos os *templates* tiveram de ser modificados. O *template device_form* foi criado para exibir as necessidades específicas dos dispositivos móveis dessa aplicação, isto é, os campos para cadastro de nome de motorista e da placa do caminhão.

Por fim, no pacote *Model* foi criada a classe de persistência *Truck*, que representa o dispositivo móvel caminhão, possibilitando o cadastro do nome do motorista e placa do veículo.

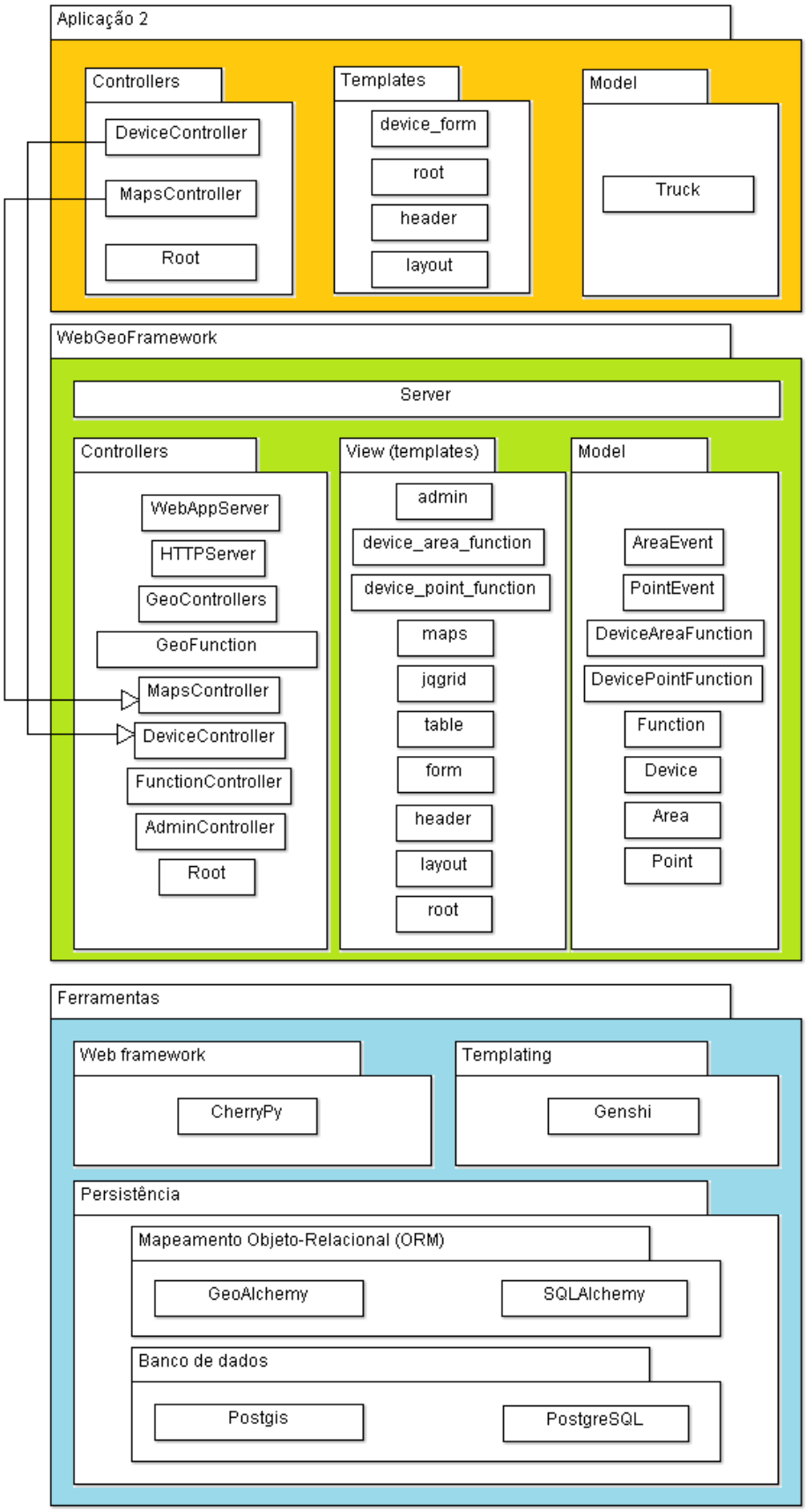

Figura 21 – Diagrama de módulos da Aplicação 2
Fonte: elaborada pelo autor (2012)

É importante ressaltar que os modelos de dados do *framework*, que são mapeados em tabelas e colunas no banco de dados, são criados no próprio *framework* e acessados nas aplicações que o estendem, porém, não há restrição à criação de novos modelos de dados pelas aplicações. Há, ainda, a possibilidade de associação de novos modelos criados pela aplicação com modelos já existentes no *framework*. Esse caso específico foi demonstrado nesta aplicação, em que a classe *Truck* possui um relacionamento 1 para 1 com a classe *Device*, criada pelo *framework*.

Além da criação de páginas para cadastro e visualização de caminhões, houve modificações no funcionamento da página de mapa, que agora exibe o evento mostrando qual caminhão o gerou.

Também foram alterados os *templates*, por meio da edição de menus e títulos da aplicação, demonstrando a facilidade de modificação de estilo das páginas da aplicação. A página inicial da aplicação, que pode ser vista na Figura 22, foi modificada para que apresente a lista dos caminhões cadastrados. Nessa figura também é possível visualizar que o menu superior possui rótulos diferentes para as páginas de visualização de eventos e de mapa.

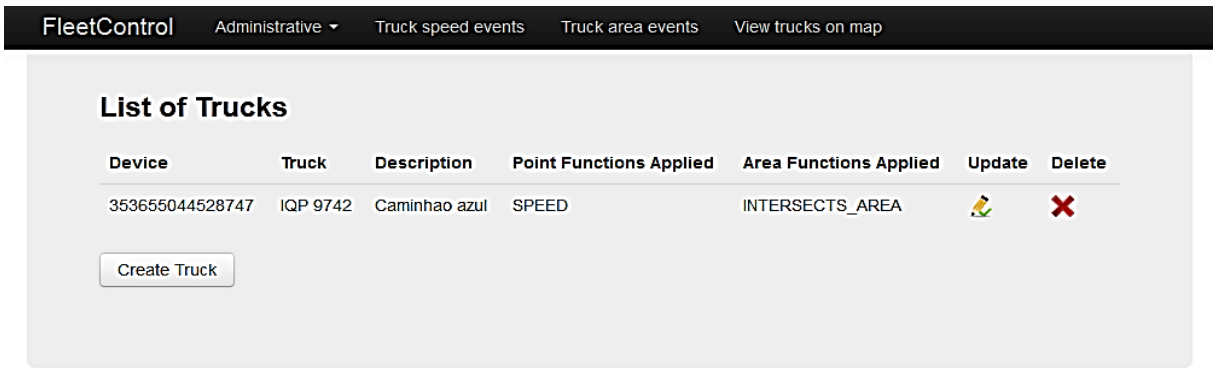

Figura 22 – Página inicial da aplicação de controle de frota de caminhões (Aplicação 2)
Fonte: elaborada pelo autor (2012)

Para o cadastro de caminhão, a página de cadastro *Device* foi modificada, apresentando agora campos para cadastro da placa do caminhão e do motorista, como pode ser visto na Figura 23, a seguir.

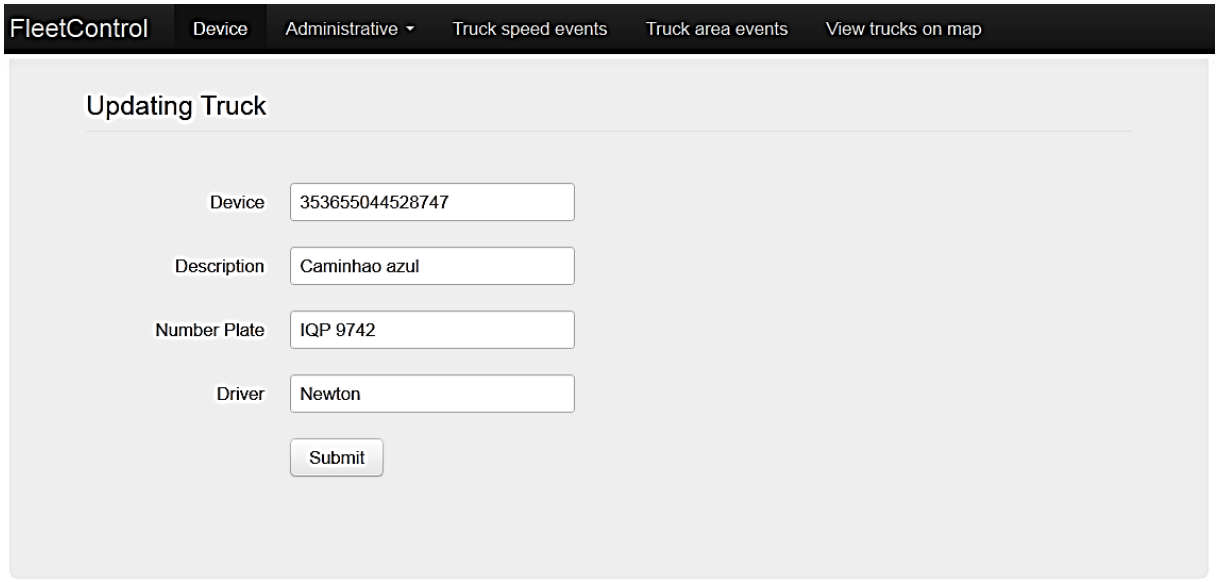

Figura 23 – Página para edição de registro de caminhão
Fonte: elaborada pelo autor (2012)

A seguir, na Figura 24, é possível ver a reutilização da página de cadastro de parâmetros de funções espaço-temporais. Nessa figura também é possível visualizar que a função SPEED receberá o parâmetro *"max=50"* quando analisar os dados do dispositivo móvel *Caminhão azul*. Dessa forma, se o dispositivo ultrapassa a velocidade de 50km/h, um evento semântico é gerado.

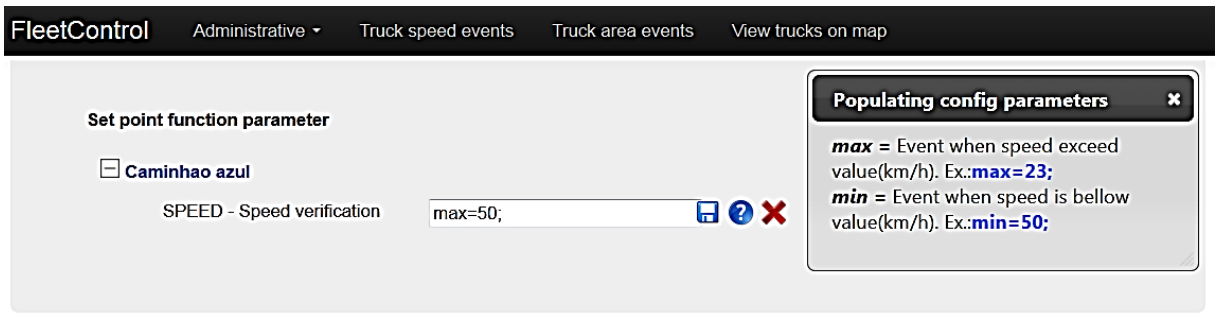

Figura 24 – Página para parametrização de função espaço-temporal
Fonte: elaborada pelo autor (2012)

A aplicação de controle de frotas utilizou também uma função espaço-temporal para áreas. Na Figura 25 é apresentada a página de parametrização de funções espaço-temporais para áreas. Constata-se, a partir dessa imagem, que o dispositivo móvel *Caminhão azul* gerará um evento INTERSECTS_AREA caso interseccione a área *Lake Karlsruhe.*

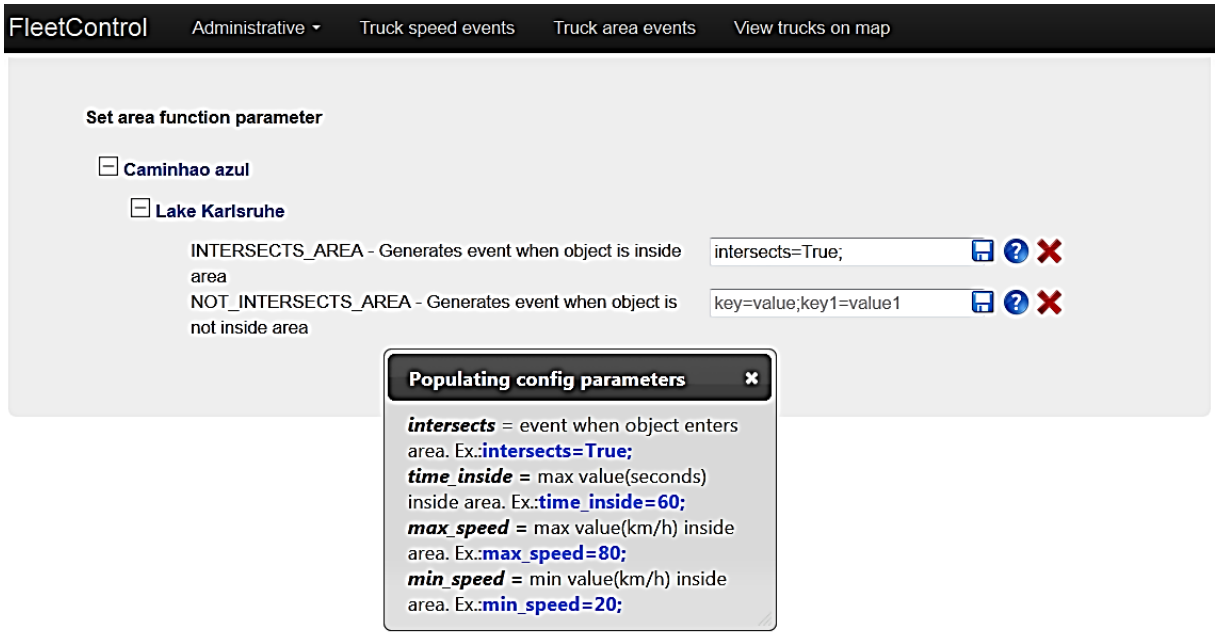

Figura 25 – Página para parametrização de função espaço-temporal que envolve áreas geográficas
Fonte: elaborada pelo autor (2012)

Na Figura 26, a seguir, é possível visualizar um mapa com um polígono que representa a área *Lake Karlsruhe*, bem como os pontos do dispositivo móvel *Caminhão azul* interseccionando essa área. Do lado direito da imagem os eventos SPEED e INTERSECTS_AREA são apresentados sinalizando que eles foram gerados pelo mesmo dispositivo móvel, isto é, o *Caminhão azul*. Clicando nos botões *Event Info*, as janelas do lado esquerdo da imagem são apresentadas e nelas podemos ver que o dispositivo móvel *Caminhão azul* excedeu o limite de velocidade, pois trafegou a 158,26 km/h. Também podemos ver que o caminhão interseccionou a área *Lake Karlsruhe*.

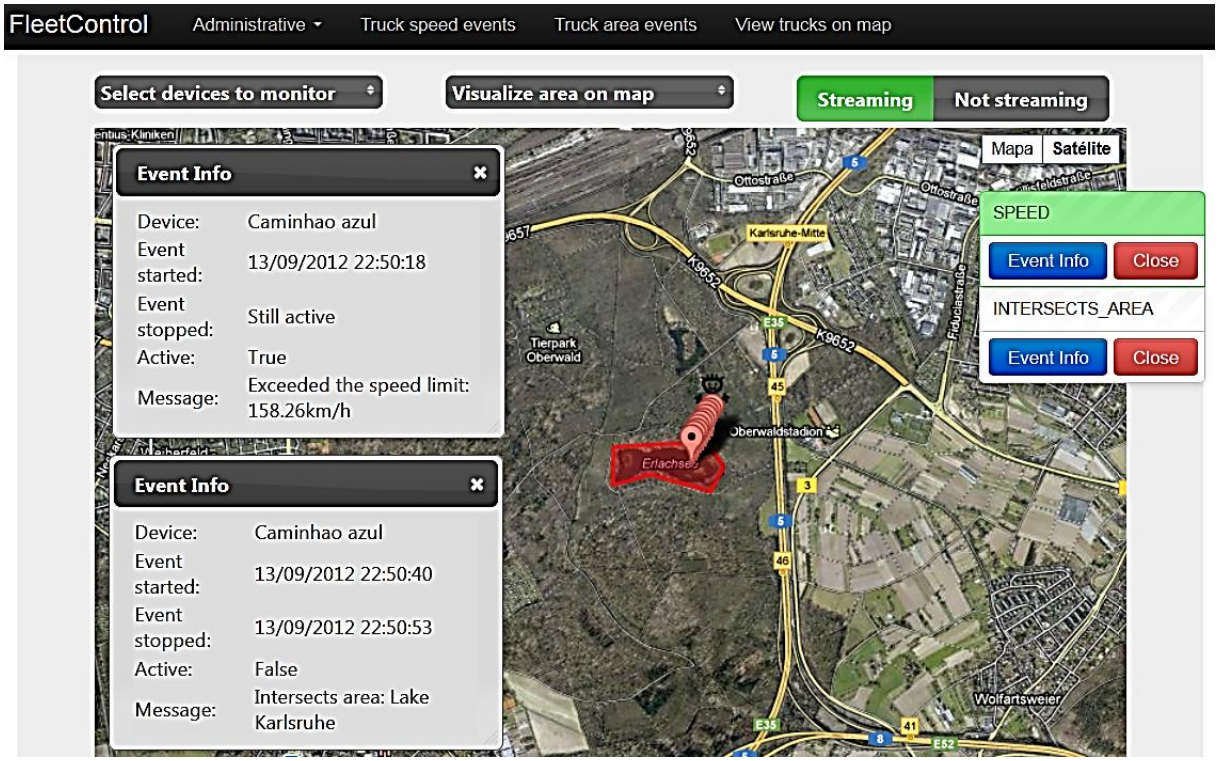

Figura 26 – Página de visualização e monitoramento de eventos semânticos e dados espaço-temporais da Aplicação 2
Fonte: elaborada pelo autor (2012)

Os eventos semânticos de funções espaço-temporais que não envolvem relações com áreas geográficas na Aplicação 2 são exibidos na Figura 27. Nessa página, podemos ver que o *Caminhão azul* ultrapassou o limite de velocidade cinco vezes e que no momento em que a página foi visualizada ele trafegava em velocidade superior ao limite estabelecido, pois o evento semântico continuava ativo.

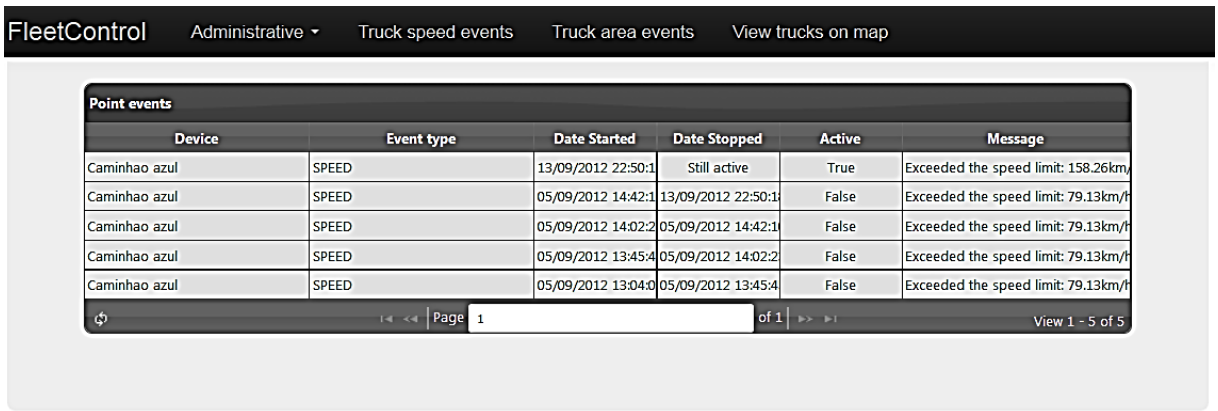

Figura 27 – Página para visualização de eventos semânticos gerados na Aplicação 2
Fonte: elaborada pelo autor (2012)

As funções espaço-temporais que utilizam dados de áreas geográficas na Aplicação 2 geraram os eventos semânticos apresentados na Figura 28. Nessa imagem é perceptível que o caminhão trafegou quatro vezes pela área proibida e que no momento que a página foi visualizada o *Caminhão azul* não interseccionava a área proibida.

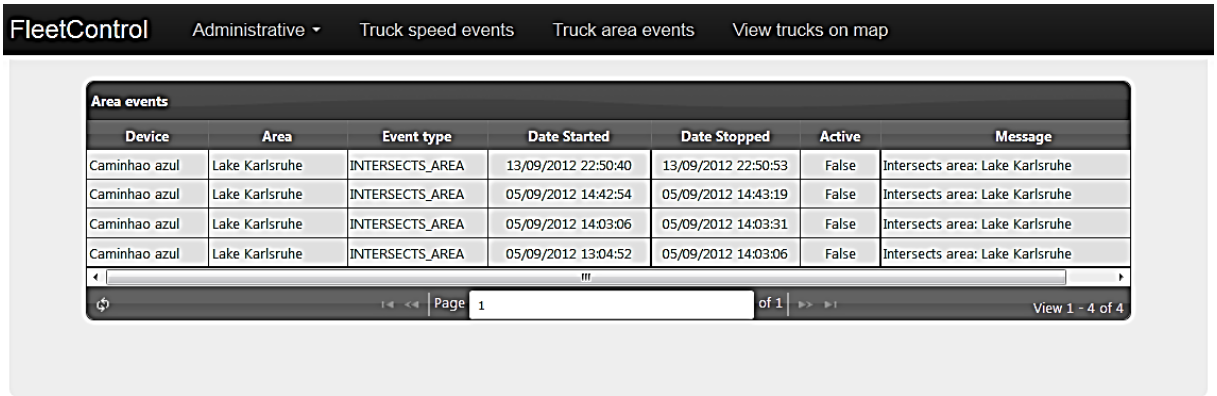

Figura 28 – Página para visualização de eventos semânticos que envolveram áreas geográficas na Aplicação 2
Fonte: elaborada pelo autor (2012)

Para o desenvolvimento da aplicação da avaliação 2 foram necessárias menos de 400 linhas de código, sendo que quase a metade delas foi escrita em *templates*, os quais oferecem bastantes possibilidades de reúso, mas grande trabalho em desenvolvimento.

Considerando que a aplicação poderia ser utilizada para fins comerciais, as 400 linhas de código podem ser consideradas aceitáveis, já que as funcionalidades proporcionadas pelo uso do *framework* e a criação de interfaces costumam envolver bastante trabalho, portanto, muitas horas/homem.

4.3 – APLICAÇÃO 3 – IDENTIFICAÇÃO DE OBJETOS MÓVEIS MUITO PRÓXIMOS UM DO OUTRO

Para demonstrar a identificação de eventos em que os dados espaço-temporais de dois dispositivos móveis necessitam ser comparados, foi criada uma aplicação que simula medidas de proteção criadas por juízes para manter a segurança de vítimas agredidas ou ameaçadas. Nesses casos, há uma distância mínima entre agressor e vítima que deve ser respeitada.

Para desenvolver a aplicação não foi necessária a criação de interfaces, apenas realizou-se a modificação do arquivo de configuração para não apresentar páginas referentes a áreas geográficas.

Na Figura 29 é apresentado um diagrama de camadas em que pode-se visualizar o pequeno esforço de desenvolvimento necessário para a criação da Aplicação 3. O controlador *MinimumDistance* implementou o controlador *GeoFunction* fornecendo o algoritmo espaço-temporal desenvolvido. O controlador *Root*, fornecido pelo *framework*, não precisou ser modificado, visto que a interface não precisou ser modificada.

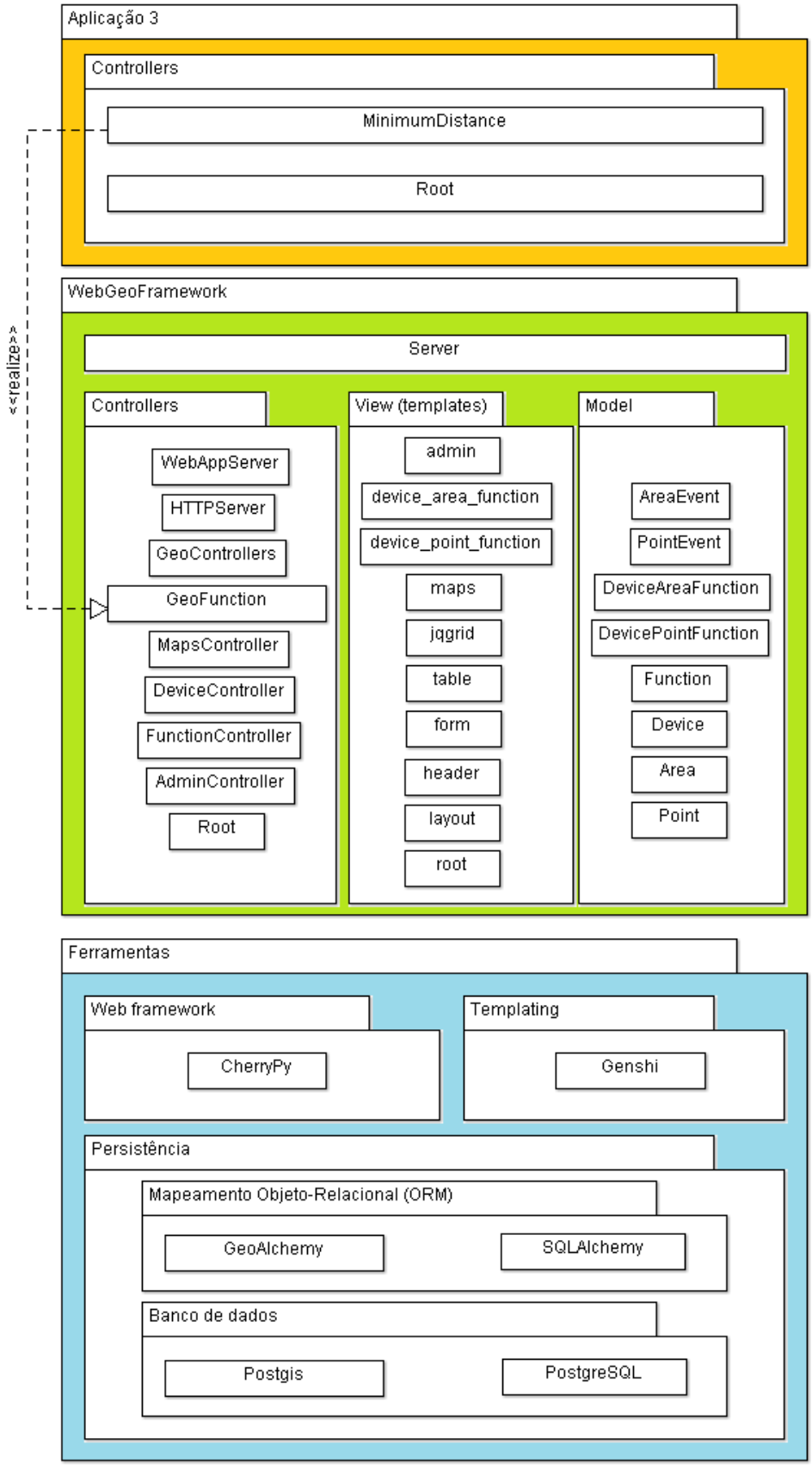

Figura 29 – Página de exibição da função espaço-temporal criada na Aplicação 3
Fonte: elaborada pelo autor (2012)

Foi criada uma função espaço-temporal, visualizada na Figura 30, que utiliza a operação espacial do SGBD de cálculo de distância entre pontos geográficos. Com o uso dessa operação foi possível calcular o afastamento entre as últimas localizações dos dispositivos móveis. Os parâmetros da função espaço-temporal criada são: o valor mínimo de distância e um identificador do dispositivo do qual se deve respeitar a distância mínima.

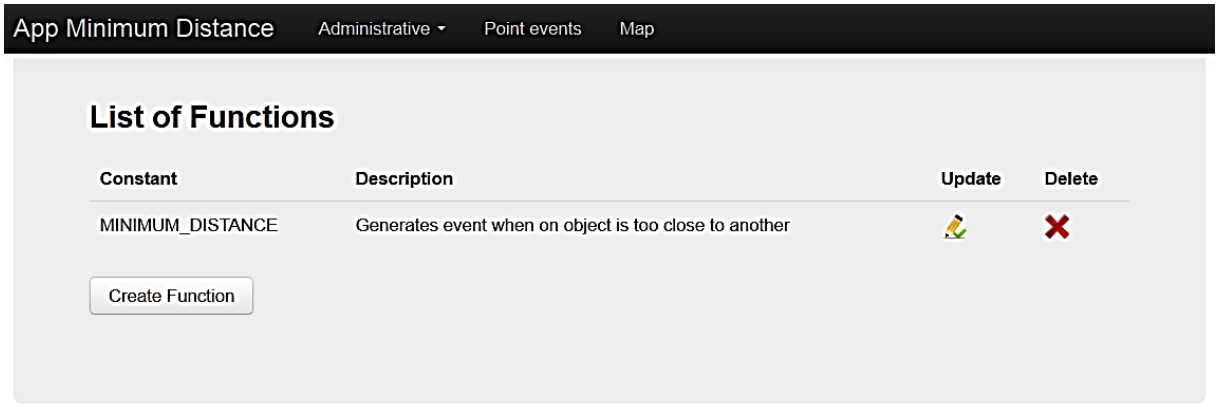

Figura 30 – Página de exibição da função espaço-temporal criada na Aplicação 3
Fonte: elaborada pelo autor (2012)

Dois dispositivos móveis foram criados no sistema para simular objetos móveis que deveriam manter uma distância mínima, os quais podem ser visualizados na Figura 31. Nessa figura é perceptível que a função espaço-temporal desenvolvida (MINIMUM_DISTANCE) só é aplicada ao dispositivo que simula o agressor, isto é, ao objeto móvel que infringe a lei quando fica muito próximo do outro dispositivo.

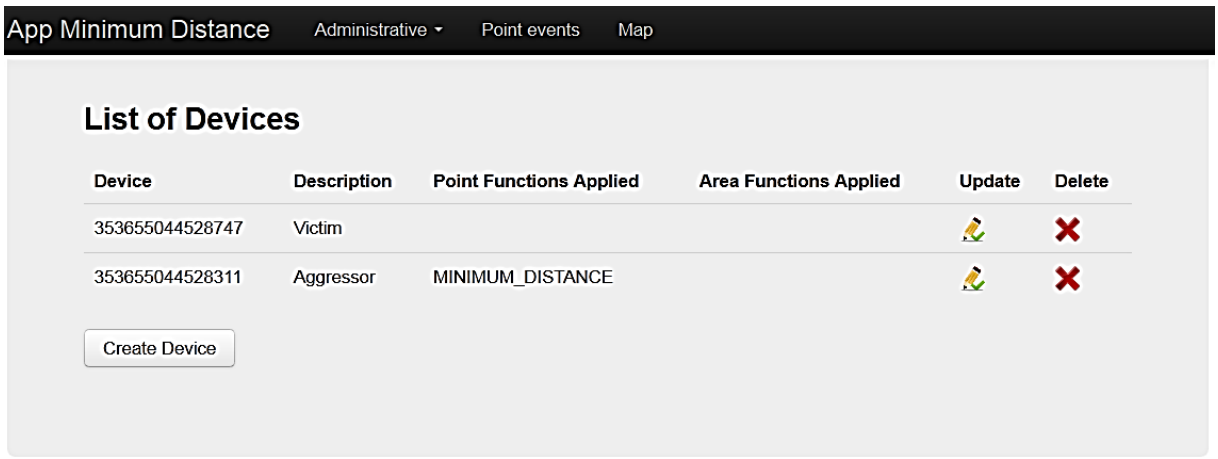

Figura 31 – Página para visualização de dispositivos móveis cadastrados na Aplicação 3
Fonte: elaborada pelo autor (2012)

Na Figura 32 a página de mapa é apresentada e o evento semântico criado na aplicação de avaliação é exibido. A mensagem apresentada pelo evento indica que o dispositivo móvel que representa o agressor ficou a 1.271 metros de distância da vítima, porém, a distância mínima que deveria ser respeitada era de 1.300 metros. Também podemos constatar que o evento permanece ativo, indicando que o agressor continua infringindo a lei no momento em que a página foi acessada.

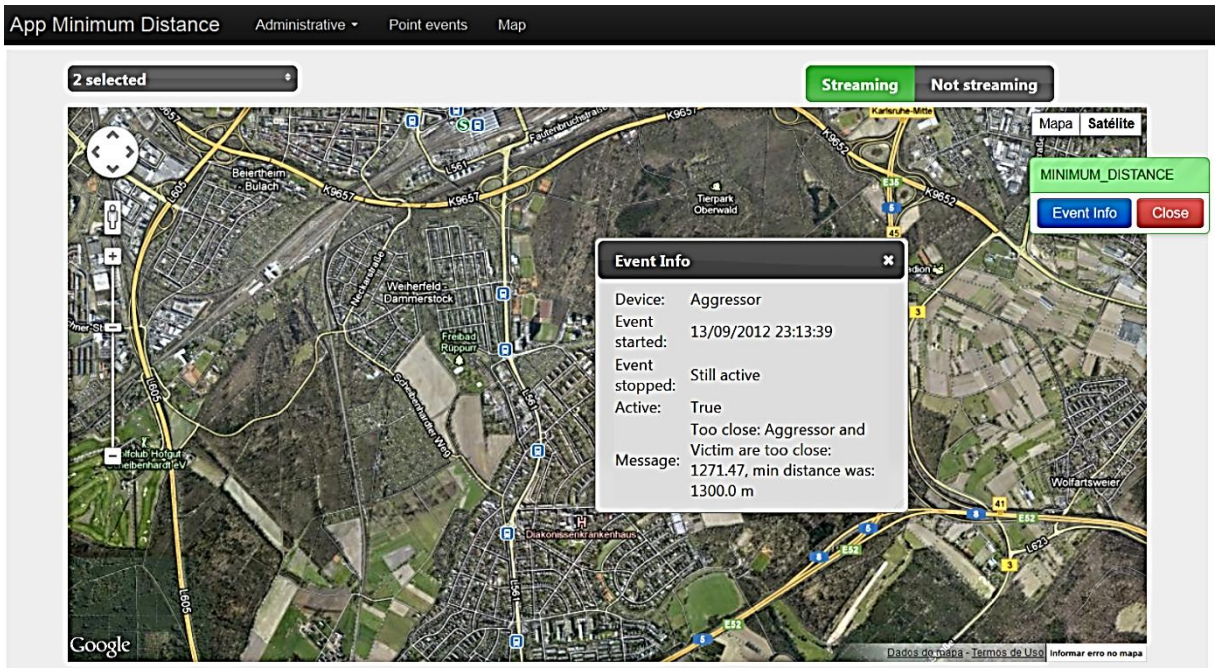

Figura 32 – Página de monitoramento e visualização de eventos semânticos e de dispositivos móveis da Aplicação 3
Fonte: elaborada pelo autor (2012)

O histórico de eventos semânticos demonstra o número de vezes que o agressor infringiu a lei ao não respeitar a distância mínima da vítima. Esse histórico pode ser visualizado na Figura 33, apresentada a seguir.

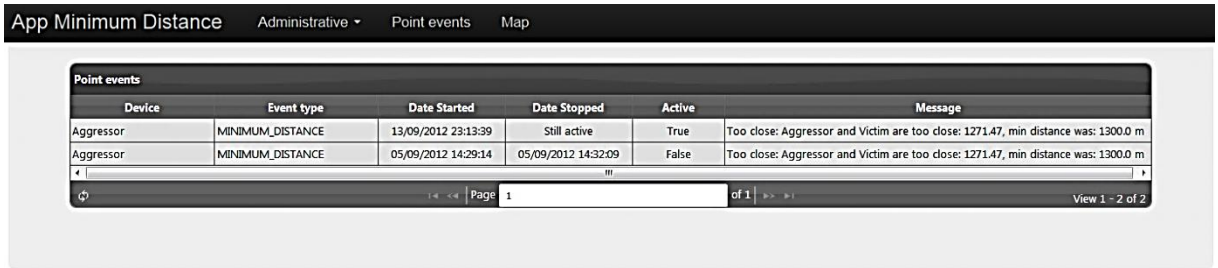

Figura 33 – Página com lista de eventos semânticos gerados por distância mínima não respeitada
Fonte: elaborada pelo autor (2012)

Para o desenvolvimento dessa aplicação foram necessárias menos de 50 linhas de código. Destas, aproximadamente 20 foram criadas apenas para o desenvolvimento da função espaço-temporal. Dessa forma, deixa-se claro, mais uma vez, que as aplicações que não necessitam de interfaces customizadas necessitam de uma quantidade muito menor de código-fonte para a extensão do *framework*.

No Gráfico 1 pode ser visualizado, em número de linhas de código, o esforço necessário para estender o *framework* nas aplicações de avaliação. É perceptível que o *framework* foi muito efetivo para a criação e a utilização de novas funções espaço-temporais, pois nas avaliações 1 e 3 o número de linhas de código criadas no desenvolvimento de código de aplicação foi muito pequeno.

Na avaliação 2, que utilizou funções espaço-temporais providas pelo *framework*, consideramos aceitáveis as 400 linhas de código, pois a quantidade de modificações visuais que a aplicação necessitou foi grande e o uso das funções espaço-temporais do *framework*, que não foram modificadas, compõem o fim principal dessa aplicação.

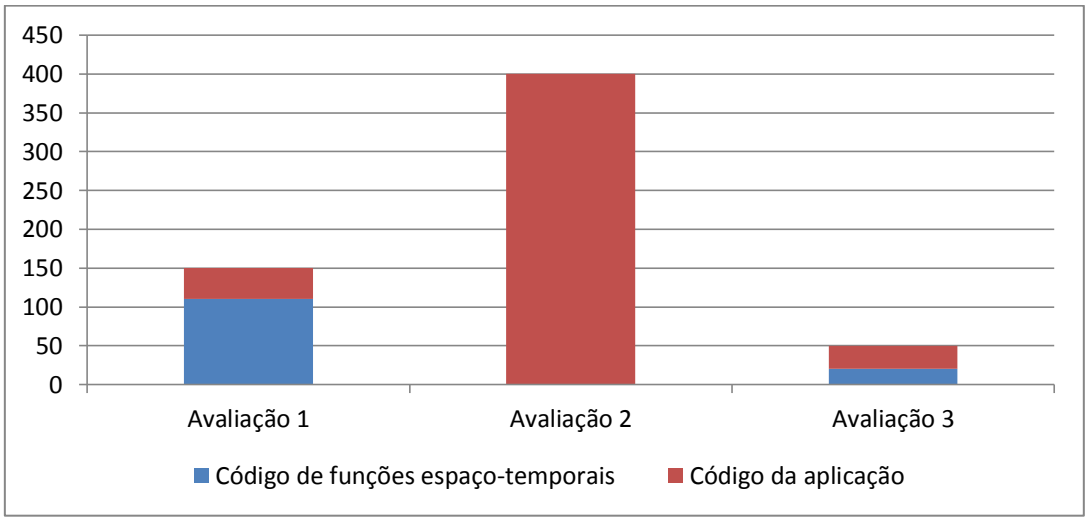

Gráfico 1 – Número de linhas necessárias para o desenvolvimento de aplicações de validação
Fonte: elaborada pelo autor (2012)

5 CONCLUSÃO

Objetivou-se, com este trabalho, o desenvolvimento de um *framework web* para recebimento, análise e visualização de dados espaço-temporais provenientes de objetos móveis. Além da criação de um *framework*, pretendia-se que ele oferecesse funções espaço-temporais para a identificação de eventos semânticos em objetos móveis, como a identificação de velocidade máxima ultrapassada e de objeto móvel interseccionando área geográfica sem permissão para fazê-lo.

O *framework* proposto neste trabalho e também as funções espaço-temporais citadas foram implementados. Para a validação do *framework*, desenvolveu-se três aplicações, que o estenderam e mostraram a viabilidade de uso do *framework* tanto no meio acadêmico como no meio corporativo.

O desenvolvimento das aplicações de avaliação do *framework* foi bastante simples e rápido. Extensões e modificações foram feitas na interface de interação com o usuário e também nas classes de persistência, mostrando que o *software* criado poderá ser utilizado de forma ágil tanto para pesquisas como para aplicações comerciais.

As possibilidades de utilização do *framework* são variadas e numerosas e poderão ser aumentadas em trabalhos futuros. Além da criação de outras funções espaço-temporais, há a possibilidade de desenvolvimento de um módulo que forneça o armazenamento das trajetórias nas quais o evento semântico está ativo. Outro componente que pode ser facilmente desenvolvido é um módulo para exibição, em um plano cartesiano, dos dados espaço-temporais, que poderia oferecer o *download* de imagens e relatórios.

REFERÊNCIAS

ALVARES, L. O; LOY, A. M.; RENSO, C.; BOGORNY, V. An Algorithm to Identify Avoidance Behavior in Moving Object Trajectories. **Journal of the brazilian computer society**, v. 17, n. 3, p. 193-203, out. 2011.

BOGORNY, V.; AVANCINI, H.; PAULA, B. C. de; KUPLICH, C. R.; ALVARES, L. O. Weka-STPM: from trajectory samples to semantic trajectories. **Transactions in GIS**, New Jersey, v. 15, n. 2, p. 227-248, abr. 2011.

BRAZ, F. J.; BOGORNY, V. **Introdução a trajetórias de objetos móveis**. 1. ed. Joinville: Univille, 2012.

CÂMARA, G.; CASANOVA, M. A.; HEMERLY, A. S.; MAGALHÃES, G. C.; MEDEIROS, C. M. B. **Anatomia de sistemas de informação geográfica**. Rio de janeiro: INPE, 1996.

CARBONI, E. M. **Análise de dados espaço-temporais gerados por dispositivos móveis**. 2011. 51 f. Trabalho de conclusão de curso (Bacharel em Sistemas de Informação)–Universidade Federal de Santa Catarina, Florianópolis, 2011.

CHERRYPY. [S.l.], 2012. Disponível em: <www.cherrypy.org>. Acesso em: 24 maio 2011.

EDGEWALL SOFTWARE. **Genshi**. [S.l.], 2012. Disponível: <http://genshi.edgewall.org>. Acesso em: 3 dez. 2012.

FISCHER, T.; MACDONALD, C. An overview of the Canada Geographic Information System (CGIS). In: INTERNATIONAL SYMPOSIUM ON CARTOGRAPHY AND COMPUTING: APPLICATIONS IN HEALTH AND ENVIROMENT, 4., 1979, Reston. **Proceedings...** [S.l.]: Cartography and Geographic Information Society.

FORCIER, J.; BISSEX, P.; CHUN, W. **Python web development with Django: developers library**. Boston: Pearson Education, 2009.

FOURSQUARE. **About Foursquare**. [S.l.], 2012. Disponível em: <https://foursquare.com/about>. Acesso em: 2 nov. 2012.

HELLEGOUARCH, S. **CherryPy Essentials: Rapid Python Web Application Development**. Birmingham: Packt Publishing, 2007.

HIBERNATE. **ORM**. [S.l.], 2012. Disponível em: <http://www.hibernate.org/about/orm>. Acesso em: 2 maio 2012.

HUFFMAN, S. In: **PyCon US Videos - 2009, 2010, 2011**. Disponível em: <http://blip.tv/pycon-us-videos-2009-2010-2011/keynote-reddit-steve-huffman-and-alexis-ohanian-1960972>. Acesso em: 3 dez. 2012.

JOHNSON, R. E.; FOOTE, B. Designing reusable classes. **Journal of Object-Oriented Programming**, Urbana-Champaign, v.1, n.2, p. 22-35, Jun./Jul. 1988.

KUBICA, M. **HOWTO Use Python in the web**. [S.l.]: Python Software Foundation, 2012. Disponível em: <http://docs.python.org/2/howto/webservers.html>. Acesso em: 3 dez. 2012.

LEMOINE, É.; SINGH, S.; SAUERWEIN, T. **GeoAlchemy v0.6 documentation**.Disponível em: <www.geoalchemy.org>. Acesso em: 24 maio 2011.

MARTELLI, A.; ASCHER, D. **Python Cookbook**. Sebastopol, CA: O'Reilly, 2002.

MIKFI. **Wikiplaces**. [S.l.], 2012. Disponível em: <http://www.mifki.com/wikiplaces>. Acesso em: 2 nov. 2012.

OPENGIS CONSORTIUM INC.**OpenGis® simple features specification for SQL**. Wayland, MA, 1999.

ORACLE. **What is template?** [S.l.], 2012. Disponível em: <http://sun025.sun.ac.za/portalHelp2/ohw?topic=potmwhat_htm&locale=en>. Acesso em: 3 dez. 2012.

PYTHON. **OrganizationsUsingPython**. [S.l.], 2012. Disponível em: <http://wiki.python.org/moin/OrganizationsUsingPython>. Acesso em: 3 dez. 2012.

TURBOGEARS. **TurboGears Documentation**. [S.l.], 2008. Disponível em: <http://turbogears.org/2.0/docs>. Acesso em: 3 dez. 2012.

REFRACTIONS RESEARCH INC. **PostGis 1.5.3 Manual**. Victoria, BC, Canadá, 2011.

_____. **PostGIShistory**. Victoria, BC, Canadá, 2012. Disponível em: <http://www.refractions.net/products/postgis/history>. Acesso em: 18 abr. 2012.

_____. **PostGIS technical overview**.Victoria, BC, Canadá, 2012.Disponível em: <http://www.refractions.net/products/postgis/technicaloverview>. Acesso em: 18 abr. 2012.

RIEHLE, D. **Framework design: a role modeling approach**.Tese (Ph.D)–Eidgenössische Technische Hochschule Zürich. Zurique, 2000.

SIQUEIRA, F. L.; BOGORNY, V. Discovering Chasing Behavior Patterns in Moving Object Trajectories. **Transactions in GIS**, New Jersey, v. 15, n. 5, p. 667-688, out. 2011.

SPACCAPIETRA, S.; PARENT, C.; DAMIANI, M. L.; MACÊDO, J. A. F. de; PORTO, F.; VANGENOT, C. **A conceptual view on trajectories**. Itália: DKE, 2008.

SQLALCHEMY. [S.l.], 2012. Disponível em: <www.sqlalchemy.org>. Acesso em: 24 maio 2012.

W3C. **Sever-sent events**. [S.l.], 2012. Disponível em: <http://dev.w3.org/html5/eventsource>. Acesso em: 2 nov. 2012.

YI, B.; MEDEIROS, C. B. **Um modelo de dados para objetos móveis**. 2004. 74 f. Dissertação (Mestrado em Ciências da Computação)–Universidade Estadual de Campinas, Campinas, 2004.

APÊNDICE A – Código-fonte de eventos semânticos

```
class GenericEvent(Exception):
    """
        GenericEvent is an Exception raised when a geo_function
        find the right value.
        All raises of geo_functions need to extend this class
    """
    message = None

#================================================================================
# POINT EVENTS
#================================================================================
class MaxSpeedEvent(GenericEvent):
    def __init__(self, speed):
        self.speed = speed
        self.message = "Exceeded the speed limit: %.2fkm/h" % speed

class MinSpeedEvent(GenericEvent):
    def __init__(self, speed):
        self.speed = speed
        self.message = "Speed below the minimum: %.2fkm/h" % speed


#================================================================================
# AREA EVENTS
#================================================================================
class Intersects(GenericEvent):
    def __init__(self, area_name):
        self.area_name = area_name
        self.message = "Intersects area: %s" % area_name

class MaxTimeInside(GenericEvent):
    def __init__(self, area_name, max_time, time_inside):
        self.area_name = area_name
        self.max_time = max_time
        self.time_inside = time_inside
        self.message = "Intersected area: %s \
            for too much time: %ss, max \
            time was %ss" % (area_name,
                             time_inside,
                             max_time)
```

APÊNDICE B – Código-fonte de funções espaço-temporais

```
from geoalchemy.postgis import pg_functions as functions
from geo.events.generated_event import *
import cherrypy

class GeoFunction(object):
  """
      Generic class that all geo_function need to extend
  """
  def apply_function(self, *k, **kw):
      assert NotImplementedError


class SPEED(GeoFunction):
  """
    Functions related to device point and speed:
    Current functions:
      * max
        raise MaxSpeedEvent when speed passes parameters['max']
      * min
        raise MinSpeedEvent when speed is bellow parameters['min']
  """

  def apply_function(self, point, parameters):
    """
      Calculate distance between last inserted point and current point from the
same device.
      Calculate time difference between both points.
      Speed = distance/time_difference
      Multiplies by 3.6 to get speed in km/h
    """
    speed = None
    session = cherrypy.request.db
    if cherrypy.engine.serverDB.has_key(point.device_id, "last_point"):
      last_inserted_point = cherrypy.engine.serverDB.get_data(
                                      point.device_id, "last_point")
      last_inserted_point_geo = last_inserted_point.geo_point
      distance =  session.scalar(
                  functions.distance(
                        functions.transform(point.geo_point,2163),
                        functions.transform(last_inserted_point_geo,2163)))
      device_time = last_inserted_point.device_timestamp
       time_difference = (point.device_timestamp - device_time).seconds
      speed = distance/time_difference

      speed *= 3.6  # converting m/s to km/h

    if speed and parameters:
      if parameters.has_key('max') :
        max_speed = parameters.get('max')
        if speed>float(max_speed):
          raise MaxSpeedEvent(speed=speed)
      if parameters.has_key('min'):
        min_speed = float(parameters.get('min'))
        if speed<min_speed:
```

```
            raise MinSpeedEvent(speed=speed)
        return speed


class INTERSECTS_AREA(GeoFunction):
    """
        Functions related to device point intersecting area:
        Current parameters:
            * intersects
              raise Intersects
            * time_inside
              raise MaxTimeInside
            * max_speed (inside an area)
              raise MaxSpeedInside
            * min_speed (inside an area)
              raise MinSpeedInside
    """

    def apply_function(self, point, parameters, area):
        self.point=point
        self.parameters=parameters
        self.area= area
        self.server_db = cherrypy.engine.serverDB
        self.session = cherrypy.request.db
        if 'intersects' in parameters:
            if parameters['intersects'].upper()=="TRUE":
                self.intersects()
        elif 'time_inside' in parameters:
            try:
                time_inside = int(parameters['time_inside'])
            except:
                time_inside = None
            if time_inside:
                self.time_inside()
        elif 'max_speed' in parameters:
            try:
                max_speed = float(parameters['max_speed'])
            except:
                max_speed = None
            if max_speed:
                self.max_speed()
        elif 'min_speed' in parameters:
            try:
                min_speed = float(parameters['min_speed'])
            except:
                min_speed = None
            if min_speed:
                self.min_speed()

    def intersects(self):
        """
            Raise Intersects when point is intersecting area
        """
        intersects =  self.session.scalar(
                            functions.intersects( self.point.geo_point,
                                                  self.area.geom))
        if intersects:
            raise Intersects(self.area.name)
```

```
def time_inside(self):
  """
    Raise MaxTimeInside if the device
    stay more than 'time_inside'seconds inside area
  """
  time_inside = int(self.parameters['time_inside'])
  intersects =  self.session.scalar(functions.intersects(
                self.point.geo_point, self.area.geom))
  if intersects:
    first_time_inside = self.server_db.get_data(
                              self.point.device_id, "time_inside")
    if not first_time_inside:
      self.server_db.set_data(
          self.point.device_id,
          "time_inside",
          (self.area.id, self.point.device_timestamp))
    else:
      if first_time_inside[0]==self.area.id:
        time_first_intersect = first_time_inside[1]
        if self.point.device_timestamp>time_first_intersect:
          device_time = self.point.device_timestamp
          difference_between_time = device_time-time_first_intersect
          if difference_between_time.seconds>time_inside:
            raise MaxTimeInside(
                self.area.name,
                str(time_inside),
                str(difference_between_time.seconds))
  else:
    first_time_inside = self.server_db.get_data(
                self.point.device_id, "time_inside")
    if first_time_inside:
      # device was not for enough time
      # intersecting area, so, we need to delete 'time_inside'
      if first_time_inside[0]==self.area.id:
        self.server_db.remove_key(self.point.device_id, "time_inside")

def max_speed(self):
  """
    Raise MaxSpeedInside if the device
    get over 'max_speed' inside area
  """
  speed_limit = float(self.parameters['max_speed'])
  intersects =  self.session.scalar(
            functions.intersects(
                  self.point.geo_point,
                  self.area.geom))
  if intersects:
    last_point = self.server_db.get_data(self.point.device_id, "max_speed")
    if last_point:
      if last_point[0]==self.area.id:
        last_point_object = last_point[1]
        last_inserted_point_geo = last_point_object.geo_point
        distance =  self.session.scalar(
                functions.distance(
                    functions.transform(self.point.geo_point,2163),
                    functions.transform(last_inserted_point_geo,2163)))
        time_difference = (self.point.device_timestamp -
```

```
                    last_point_object.device_timestamp).seconds
          speed = distance/time_difference
          speed *= 3.6
          if speed>speed_limit:
            raise MaxSpeedInside(self.area.name, str(speed_limit), str(speed))

      self.server_db.set_data(self.point.device_id,
              "max_speed", (self.area.id, self.point))
    else:
      last_point = self.server_db.get_data(self.point.device_id, "max_speed")
      if last_point:
        if last_point[0]==self.area.id:
          self.server_db.remove_key(self.point.device_id, "max_speed")

  def min_speed(self):
    """
      Raise MinSpeedInside if the device is above 'min_speed' inside area
    """
    speed_limit = float(self.parameters['min_speed'])
    intersects =  self.session.scalar(
            functions.intersects(self.point.geo_point, self.area.geom))
    if intersects:
      last_point = self.server_db.get_data(self.point.device_id, "min_speed")
      if last_point:
        if last_point[0]==self.area.id:
          last_point_object = last_point[1]
          last_inserted_point_geo = last_point_object.geo_point
          distance =  self.session.scalar(
                  functions.distance(
                    functions.transform(self.point.geo_point,2163),
                    functions.transform(last_inserted_point_geo,2163)))
          time_difference = (self.point.device_timestamp -
                     last_point_object.device_timestamp).seconds
          speed = distance/time_difference
          speed *= 3.6
          if speed<speed_limit:
            raise MinSpeedInside(self.area.name,
                        str(speed_limit),
                        str(speed))
      self.server_db.set_data(self.point.device_id,
                  "min_speed",
                  (self.area.id, self.point))
    else:
      last_point = self.server_db.get_data(self.point.device_id, "min_speed")
      if last_point:
        if last_point[0]==self.area.id:
          self.server_db.remove_key(self.point.device_id, "min_speed")
```

APÊNDICE C – Código-fonte de classe persistida com ORM

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, \
    Integer, Unicode, String, \
    DateTime, BOOLEAN, PickleType, \
    ForeignKey, UniqueConstraint
from sqlalchemy.orm import relationship, backref
from sqlalchemy.ext.associationproxy import association_proxy

import geoalchemy
from geoalchemy import WKTSpatialElement

from datetime import datetime

Base = declarative_base()

class Point(Base):
  __tablename__ = 'points'
  id = Column(Integer, primary_key=True)
  device_id = Column(Integer, ForeignKey('devices.id'))
  timestamp = Column(DateTime, default=datetime.now)
  geo_point = geoalchemy.GeometryColumn(geoalchemy.Point(2))
  device_timestamp = Column(DateTime, default=datetime.now)

  device = relationship("Device",backref=backref("device_points",
               cascade="all, delete-orphan"))

  def __init__(self, device, lati, longi, timestamp=None, device_timestamp=None):
    self.device = device
    self.timestamp = timestamp
    self.device_timestamp = device_timestamp
    self.geo_point = WKTSpatialElement("POINT(%s %s)" %(lati, longi))

  def __repr__(self):
    return "<Point('%s','%s', '%s')>" % (self.device_id,
                        self.timestamp,
                        self.device_timestamp)
```

APÊNDICE D – Código-fonte de *template web*

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  xmlns:py="http://genshi.edgewall.org/">
<xi:include href="layout.html" />
<head></head>
<body>
  <h3 py:if="defined('title')">List of ${title}s</h3>
  <table py:if="items" class="table table-hover">
    <thead>
      <tr>
        <py:for each="column_name in columns_names">
          <th>${column_name}</th>
        </py:for>
        <th>Update</th>
        <th>Delete</th>
      </tr>
    </thead>
    <tbody>
      <py:for each="i in range(len(items))">
        <tr>
          <py:for each="attribute in list_columns">
            <py:choose test="defined(attribute)">
              <py:when test="1">
                <td>${value_of(attribute)[i]}</td>
              </py:when>

              <py:otherwise>
                <td>${getattr(items[i], attribute)}</td>
              </py:otherwise>
            </py:choose>
          </py:for>
          <td>
             <a href="request/${items[i].id}">
                   <img src="${url('/media/images/update.png')}"
                         width="20" height="20"
                         alt="update" />
             </a>
          </td>
          <td>
             <a href="delete/${items[i].id}">
                   <img src="${url('/media/images/delete.png')}"
                         width="20" height="20"
                         alt="delete" />
             </a>
          </td>
        </tr>
      </py:for>
    </tbody>
  </table>
  <p> <a class="btn" href="request">Create ${title}</a></p>
</body>
</html>
```

APÊNDICE E – Artigo

Um *Framework* para Construção de Aplicações *Web* com Dados Espaciais e Espaço-Temporais

Newton A. Sander[1] and Vania Bogorny[2]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

[2]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

newtonsan@inf.ufsc.br, vania@inf.ufsc.br

Abstract. *This work had as goal the creation of a framework that identifies events through the analysis of spatio-temporal data of mobile objects and the development of a web application that uses the features provided by this framework. To validate the framework, three evaluations were developed to show the viability of commercial and academic use of the framework. Thus, it was demonstrated that the developed framework can be used by developers to facilitate the creation of websites for academic, commercial and government purposes.*

Resumo. *Este trabalho teve como objetivo a criação de um* framework *que identifique eventos por meio da análise de dados espaço-temporais de objetos móveis e o desenvolvimento de uma aplicação* web *que utilize as funcionalidades disponibilizadas por esse* framework*. Para validar o funcionamento do* framework *foram desenvolvidas três avaliações, as quais mostram a viabilidade de seu uso para propósitos comercial e acadêmico. Comprovou-se que o* framework *proposto pode ser utilizado por desenvolvedores para facilitar a criação de* websites *tanto para o meio acadêmico como para o comercial e governamental.*

1. Introdução e Motivação

Devido à popularização de dispositivos móveis coletores de pontos localizados no tempo e no espaço (como os sistemas de posicionamento global – do inglês *Global Positioning System* (GPS) –, os telefones celulares, as redes de sensores, entre outros) e à grande quantidade de aplicações que podem se beneficiar com o uso dos dados coletados por tais dispositivos, existe uma demanda crescente por sistemas que manipulem dados espaço-temporais.

No entanto, as aplicações que manipulam esses dados são mais complexas que as aplicações que utilizam dados convencionais, pois manipulam não só os dados não espaciais, mas também os dados geográficos e a dimensão tempo. Além disso, tais aplicações fazem uso de consultas mais complexas, envolvendo o tempo, o espaço e as informações semânticas, por exemplo: buscar um aumento excessivo da velocidade de um objeto móvel [CARBONI, 2011]; buscar trajetórias que se interseccionam em um certo intervalo de tempo; ou identificar determinados padrões de comportamento, como perseguição [SIQUEIRA; BOGORNY, 2011] ou desvio [ALVARES, L. O; LOY, A.M.; RENSO, C.; BOGORNY, V., 2011].

As aplicações *web* que utilizam dados espaço-temporais podem fazer um amplo uso de *web frameworks full stack* genéricos, os quais diminuem o retrabalho, isto é, diminuem a necessidade de realizar trabalhos já feitos. Apesar disso, ainda existe muito retrabalho que

pode ser extirpado com a utilização de um *web framework* específico para a manipulação de dados espaço-temporais.

Pensando nisso, neste trabalho desenvolveu-se um *web framework* que, além dos serviços essenciais de um *web framework* (sistema de *templates*, servidor de aplicação e gerenciador de requisições), proporcionará ao desenvolvedor um módulo *object-relational mapping* (ORM) com suporte a dados geográficos; um servidor para o recebimento e o armazenamento dos dados geográficos; uma interface para a visualização dos dados localizados no tempo e no espaço; e, por fim, um pacote de funções espaço-temporais, que fornecerão uma maior abstração na manipulação de dados geográficos, não sendo necessária a utilização direta das operações espaciais disponíveis no banco de dados geográfico.

O *web framework* desenvolvido pode ser utilizado em aplicações nas quais é necessário fazer análise on-line de dados geográficos, os quais podem ser inseridos no banco de dados por meio do serviço de recebimento de dados provido pelo próprio *framework*. Além disso, também pode facilitar o desenvolvimento e os testes de novos algoritmos para manipular e analisar dados geográficos, que poderão ser rapidamente incorporados por meio da extensão do *framework*.

Assim, ao desenvolvedor da aplicação caberá apenas: a criação dos algoritmos de análise de dados espaço-temporais necessários para o domínio da aplicação que não estiverem implementados no *framework*; a definição dos fatos que geram eventos (e.g. o objeto entra em área proibida); o desenvolvimento de tratamentos para o evento gerado (e.g. soar um alarme); e, por fim, a criação da interface específica ao domínio da aplicação que não tem relação com a manipulação dos dados geográficos (e.g. uma página para cadastro de presidiários contendo número de cadastro de pessoa física (CPF), a foto e o nome completo dos indivíduos).

2. O Método Proposto

A partir do levantamento de requisitos necessários para o desenvolvimento de aplicações *web* que façam a análise de dados espaço-temporais em tempo real, foram definidas as funcionalidades que o *framework* suportaria. São elas: (1) o recebimento de dados geográficos por requisições *HyperText Transfer Protocol* (HTTP) ou de carregamento destes a partir de um arquivo *extensible markup language* (XML); (2) a análise dos dados em busca de eventos semânticos; (3) a possibilidade de uso de funções espaço-temporais, que podem ser associadas aos dispositivos móveis, podendo ser parametrizadas e assim gerar os eventos específicos para os objetivos de projeto; (4) a disponibilização de interface para a visualização dos dados espaço-temporais utilizando Google™ Maps; e (5) a possibilidade de uso do esquema de banco de dados definido para suportar essas funcionalidades.

Uma vez definidas as funcionalidades, desenvolveu-se o *framework web*, chamado WebGeoFramework, utilizando uma junção de *frameworks* e bibliotecas Python.

A Figura 1, a seguir, apresenta um diagrama de atividades do procedimento de análise de dados espaço-temporais no *framework*. As atividades que compõem este diagrama serão brevemente descritas no texto que o segue, com exemplos para facilitar o entendimento.

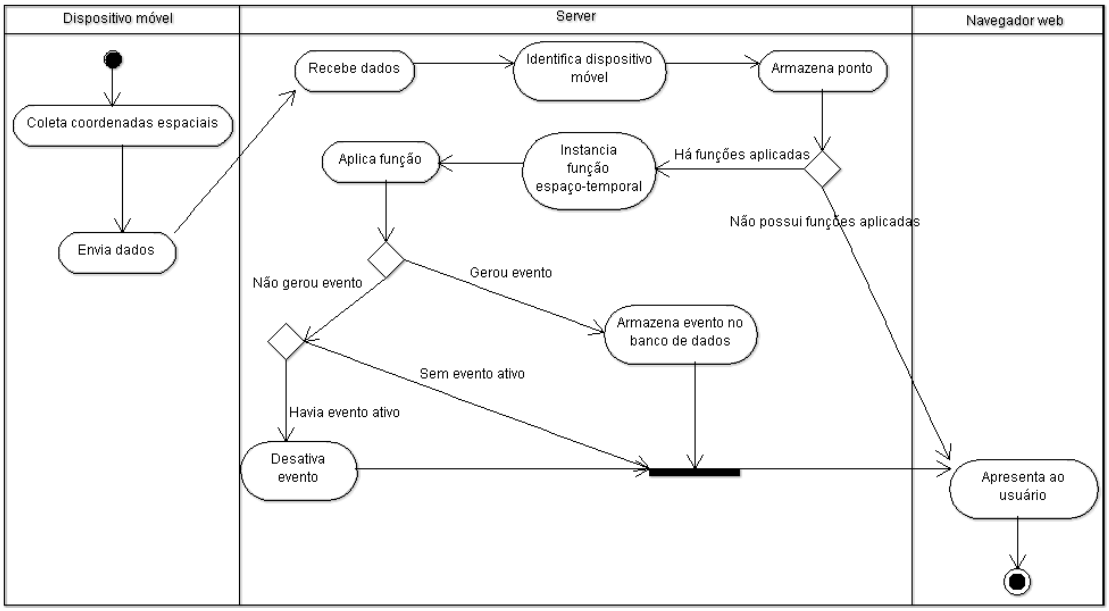

Figura 1. Diagrama de atividades da análise de dados espaço-temporais

Como apresentado na Figura 1, os dados espaço-temporais são coletados pelo dispositivo móvel, que os envia com um identificador do dispositivo e com um carimbo de tempo da coleta para a aplicação que estende o *framework*. Ao receber esses dados, a aplicação os armazena no banco de dados, chamando-os de *Point* (em inglês, ponto). Após o armazenamento é feita uma busca, no banco de dados, por parametrizações de funções espaço-temporais que estejam associadas ao dispositivo gerador do ponto. Toda parametrização encontrada é passada com o ponto para a função espaço-temporal, que então é aplicada.

Função espaço-temporal é um termo definido neste trabalho para identificar o método aplicado aos dados espaço-temporais com o objetivo de identificar eventos semânticos. As funções espaço-temporais fazem a busca e a identificação do evento desejado analisando os dados espaço-temporais provenientes de dispositivos móveis. A execução da função espaço-temporal pode, por exemplo, fazer a identificação de ocorrências de velocidade excessiva nos dados enviados por um aparelho celular acoplado a um caminhão. Quando o fato é identificado, ela deve gerar um evento com as informações da ocorrência. Esse evento é armazenado no banco de dados e apresentado ao usuário *web* que estiver monitorando o dispositivo gerador do ponto.

Cada função espaço-temporal possui à sua disposição o acesso simplificado aos parâmetros que definem seu funcionamento. Além disso, pode acessar todos os dados e operadores espaciais do banco de dados geográfico. Assim sendo, a função espaço-temporal de SPEED (velocidade), por exemplo, que deve gerar eventos quando a velocidade máxima for ultrapassada, pode ser implementada da forma descrita a seguir.

Suponha-se que tenha ocorrido a parametrização da função SPEED associando-a com um dispositivo coletor e que o valor da velocidade máxima permitida tenha sido parametrizado como sendo 50km/h. Quando um ponto for recebido pela aplicação, esta aplicará a função espaço-temporal SPEED passando o valor máximo de velocidade permitido e os dois últimos pontos gerados pelo dispositivo móvel.

Na execução da função espaço-temporal SPEED é utilizada a operação espacial do banco de dados geográfico que, usando os dois últimos pontos gerados pelo dispositivo,

calcula a distância entre os dados espaciais. Tendo a distância entre os dois últimos pontos, é necessário calcular a diferença de tempo entre eles, o que pode ser feito subtraindo os atributos de carimbo de tempo dos pontos. Com os valores de distância e de intervalo de tempo, é possível calcular a velocidade e então verificar se ela é maior que o máximo permitido, isto é, 50km/h. Caso seja, é necessário gerar um evento com as informações da ocorrência.

A apresentação dos eventos semânticos para o usuário é feita por meio do navegador *web* e utiliza-se de uma funcionalidade do HTML5 chamada *Server-Sent Events*. Essa funcionalidade, suportada pela maioria dos navegadores mais utilizados, faz com que o servidor possa mandar os dados para o usuário quando estes estiverem prontos para o envio, e não somente quando o navegador os requisitar. Dessa forma, é possível enviar pontos geográficos assim que eles forem recebidos pelo servidor, fornecendo para o usuário uma visualização em tempo real dos dispositivos móveis e dos eventos semânticos.

3. Experimentos

Para validar os requisitos do projeto do *framework*, além da aplicação funcional contida nele, foram desenvolvidas três aplicações, que serão descritas a seguir.

A primeira aplicação de avaliação foi baseada nos métodos propostos por Carboni (2011) e buscou mostrar a viabilidade de uso do *framework* no meio acadêmico para a criação e o teste de algoritmos para a análise de comportamento de objetos móveis.

Com essa aplicação, objetivou-se identificar acelerações bruscas, bem como mudanças bruscas de direção, a fim de, principalmente, reconhecer motoristas que dirigem de forma perigosa, podendo danificar ou colocar em risco a carga transportada; e identificar curvas perigosas, trechos sinuosos ou rodovias danificadas, que poderiam ter sinalização apropriada ou melhorada para prevenir acidentes.

O desenvolvimento dessa aplicação exigiu menos de 150 linhas de código, sendo que aproximadamente 110 foram utilizadas no desenvolvimento do algoritmo. Isso significa que o esforço para desenvolver a aplicação *web*, desconsiderando-se o esforço de implementação da função espaço-temporal, resumiu-se a 40 linhas de código. Comprovou-se, dessa forma, que as aplicações acadêmicas para a análise comportamental de objetos móveis podem utilizar o *framework,* tendo ganhos de produtividade e tempo de desenvolvimento.

Já para validar o uso do *framework* por aplicações comerciais, em que comumente há a necessidade de interfaces com o usuário personalizadas, foi feita uma aplicação para o controle de frotas de caminhões na Avaliação 2. Essa aplicação oferece ao usuário uma página para cadastro de dados dos caminhões e respectivos motoristas e utiliza as funções espaço-temporais oferecidas pelo *framework*, que identificam os eventos de velocidade e de intersecção de áreas.

A aplicação da Avaliação 2 foi desenvolvida com menos de 400 linhas de código, sendo que quase a metade delas foi escrita em *templates*, os quais oferecem bastantes possibilidades de reúso, mas grande trabalho em desenvolvimento. Como a aplicação poderia ser utilizada para fins comerciais, as 400 linhas de código podem ser consideradas aceitáveis, já que as funcionalidades proporcionadas pelo uso do *framework* e a criação de interfaces costumam envolver bastante trabalho, portanto, muitas horas/homem.

Por fim, a Avaliação 3 teve como fim demonstrar a identificação de eventos em que os dados espaço-temporais de dois dispositivos móveis necessitam ser comparados. Para isso,

criou-se uma aplicação que simula medidas de proteção criadas por juízes para manter a segurança de vítimas agredidas ou ameaçadas, casos em que há uma distância mínima entre o agressor e a vítima que deve ser respeitada.

Para o desenvolvimento dessa aplicação foram necessárias menos de 50 linhas de código. Destas, aproximadamente 20 foram criadas apenas para o desenvolvimento da função espaço-temporal. Dessa forma, deixa-se claro, mais uma vez, que as aplicações que não necessitam de interfaces customizadas necessitam de uma quantidade muito menor de código-fonte para a extensão do *framework*.

No Gráfico 1, a seguir, pode ser visualizado, em número de linhas de código, o esforço necessário para estender o *framework* nas aplicações de avaliação. É perceptível que o *framework* foi muito efetivo para a criação e a utilização de novas funções espaço-temporais, pois nas avaliações 1 e 3 o número de linhas de código criadas no desenvolvimento de código de aplicação foi muito pequeno.

Na Avaliação 2, que utilizou funções espaço-temporais providas pelo *framework*, consideramos aceitáveis as 400 linhas de código, pois a quantidade de modificações visuais que a aplicação necessitou foi grande e o uso das funções espaço-temporais do *framework*, que não foram modificadas, compõem o fim principal dessa aplicação.

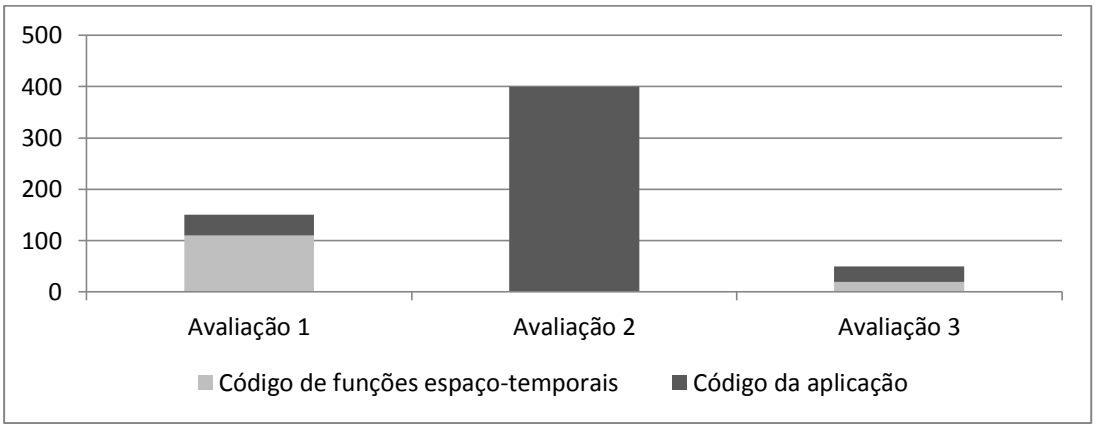

Gráfico 1. Número de linhas necessárias para o desenvolvimento de aplicações de validação

4. Conclusão e Trabalhos Futuros

Objetivou-se, com este trabalho, o desenvolvimento de um *framework web* para recebimento, análise e visualização de dados espaço-temporais provenientes de objetos móveis. Além da criação de um *framework*, pretendia-se que ele possuísse funções espaço-temporais para a identificação de eventos semânticos em objetos móveis, como a identificação de velocidade máxima ultrapassada e de objeto móvel interseccionando área geográfica sem permissão para fazê-lo.

O *framework* proposto neste trabalho e também as funções espaço-temporais citadas foram desenvolvidos. E para a validação do *framework*, desenvolveu-se três aplicações, que o estenderam e mostraram a viabilidade de uso do *framework* tanto no meio acadêmico como no meio corporativo.

O desenvolvimento das aplicações de avaliação do *framework* foi bastante simples e rápido. Extensões e modificações foram feitas na interface de interação com o usuário e

também nas classes de persistência, mostrando que o *software* criado poderá ser utilizado de forma ágil tanto para pesquisas como para aplicações comerciais.

As possibilidades de utilização do *framework* são variadas e numerosas e poderão ser aumentadas em trabalhos futuros. Além da criação de outras funções espaço-temporais, há a possibilidade de desenvolvimento de um módulo que forneça o armazenamento das trajetórias nas quais o evento semântico está ativo. Outro componente que pode ser facilmente desenvolvido é um módulo para exibição, em um plano cartesiano, dos dados espaço-temporais, que poderia oferecer o *download* de imagens e relatórios.

Referências

ALVARES, L. O; LOY, A. M.; RENSO, C.; BOGORNY, V. An Algorithm to Identify Avoidance Behavior in Moving Object Trajectories. **Journal of the brazilian computer society**, v. 17, n. 3, p. 193-203, out. 2011.

CARBONI, E. M. **Análise de dados espaço-temporais gerados por dispositivos móveis**. 2011. 51 f. Trabalho de conclusão de curso (Bacharel em Sistemas de Informação)–Universidade Federal de Santa Catarina, Florianópolis, 2011.

SIQUEIRA, F. L.; BOGORNY, V. Discovering Chasing Behavior Patterns in Moving Object Trajectories. **Transactions in GIS**, New Jersey, v. 15, n. 5, p. 667-688, out. 2011.