Fernando Benedito Veras Magalhães

Monitoramento de Unidades Móveis Baseado em Restrições de Mobilidade

São Luís

2018

Fernando Benedito Veras Magalhães

Monitoramento de Unidades Móveis Baseado em Restrições de Mobilidade

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, **como parte dos requisitos necessários** para obtenção do grau de Bacharel em Ciência da Computação.

Universidade Federal do Maranhão

Centro de Ciências Exatas e Tecnologia

Departamento de Informática

Curso de Ciência da Computação

Orientador: Francisco José da Silva e Silva

São Luís

2018

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Veras Magalhães, Fernando Benedito.
Monitoramento de Unidades Móveis Baseado em Restrições de Mobilidade / Fernando Benedito Veras Magalhães. - 2018.
37 f.

Orientador(a): Francisco José da Silva e Silva.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2018.

1. Ciência da Computação. 2. Ciências Exatas. 3. Sistemas Distribuídos. I. da Silva e Silva, Francisco José. II. Título.

Fernando Benedito Veras Magalhães

Monitoramento de Unidades Móveis baseado em Restrições de Mobilidade

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, **como parte dos requisitos necessários** para obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. São Luís, 12 de julho de 2018:

Francisco José da Silva e Silva
Orientador
Doutor em Ciência da Computação, UFMA

Prof. Luciano Reis Coutinho
Doutor em Engenharia Elétrica, UFMA

Prof. Alex Oliveira Barradas Filho
Doutor em Engenharia de Eletricidade, UFMA

São Luís
2018

Resumo

Em muitas situações faz-se necessário monitorar a localização de pessoas e veículos e cargas afim de detectar irregularidades em sua movimentação. No contexto de uma empresa que trabalha com máquinas e veículos, monitorar e controlar o posicionamento geográfico desses recursos é importante para garantir que sejam utilizados com eficiência. Para, por exemplo, realocar veículos em caso de atrasos ou desativar temporariamente máquinas que não estejam sendo utilizadas. Há ainda questões de segurança relacionadas a localização como limites de velocidade e áreas cujo acesso é restrito. Define-se aqui como restrições de mobilidade padrões de mobilidade que necessitam ser seguidos, como limites de velocidade. Esse trabalho busca disponibilizar uma abordagem para expressar os padrões de mobilidade esperados de pessoas e objetos, e detectar eventuais quebras desses padrões. Para isso foi desenvolvida uma biblioteca java denominada MobCons que permite instanciar regras de mobilidade e processar fluxos de informações de localização e velocidade baseado nessas regras utilizando Processamento de Eventos Complexos. A solução proposta abrange um amplo conjunto de regras de mobilidade e portando pode ser utilizada em diversos cenários. Este trabalho traz um estudo de caso onde a biblioteca foi utilizada para processar dados de localização adquiridos em smartphones, além de uma avaliação da solução utilizando dados gerados no simulador SUMO (Simulação de Mobilidade Urbana).

Palavras-chaves: Processamento de Eventos Complexos, Monitoramento de Localização.

Abstract

In several situations it is necessary to monitor the location of people, vehicles and cargo to detect irregularities in their movement. In the context of a company that uses machines and vehicles, monitoring and controlling the positioning of these resources, is important to ensure that they are being used efficiently. To for instance, reallocate vehicles in case of delays or temporally deactivate machines that are underused. Also, there are security issues related to location as speed limits and areas of restricted access. Here, mobility constraints are defined as mobility patterns that need to observe, for instance speed limits. This work pursues to provide an approach to express the expected mobility patterns as of people and objects and detect eventual violations to those patterns. Therefore, the java library named MobCons was developed, it allows instantiation of mobility rules and process location data-streams based on those rules using CEP (Complex Event Processing). The proposed solution encompasses a wide set of mobility rules and consequently can be applied in several scenarios. This work was evaluated using real mobility data and data generated using the SUMO (Simulation of Urban Mobility).

Key-words: Complex Event Processing, Location Monitoring.

Sumário

1 Introdução

Nos dias atuais estão disponíveis diversos dispositivos com recursos de localização como tablets, dispositivos vestíveis e sistemas presentes em veículos automotores. Para obter a localização esses dispositivos fazem uso de variadas tecnologias, sejam elas baseadas em satélites como o GPS (Sistema de Posicionamento Global), o GLONASS (Sistema de Navegação Global por Satélite) e o Galileo; ou baseadas em antenas fixas como localização por redes de celular ou localização por wi-fi. Adicionalmente esses aparelhos com funcionalidade de localização, entre os quais destacam-se os smartphones, usualmente também possuem conectividade com a Internet através de rede wi-fi e/ou redes 3G e 4G, o que permite o envio periódico de seus dados de localização. Esses dados podem portanto caracterizar eventos de movimentação. Eventos são registros imutáveis de algo que aconteceu no mundo real ou em um sistema de software e dada uma sequência ordenada de eventos através do tempo tem-se um fluxo de eventos. Define-se aqui como UM (**Unidade Móvel**) entidades cuja movimentação e velocidade serão monitoradas e como **restrições de mobilidade** padrões de mobilidade que necessitam ser seguidos, como limites de velocidade.

Aplicações de geoposicionamento e de navegação outdoor costumam tratar fluxos contínuos de dados de localização. Como consta em (CARNEY et al., 2002) enquanto aplicações baseadas em dados estáticos costumam ser passivas, ou seja esperam uma consulta para iniciar o processamento, aplicações para fluxos contínuos processam os dados a medida que eles são recebidos. Posto isto, outra característica do processamento de fluxos é trazida a tona, a baixa latência, o que significa que a diferença de tempo entre a chegada de um dado e o retorno de resultados referentes a esse dado é reduzida. O trabalho de Carney et al. também ressalta outro atributo do processamento de fluxos que é a capacidade de lidar com a característica temporal dos dados: enquanto aplicações tradicionais consideram apenas o estado atual dos dados, tenham sido eles atualizados recentemente ou não, aplicações que lidam com fluxos de dados podem ignorar dados antigos ou até processar a mudança dos dados ao longo do tempo. Como exemplo de aplicação que faz processamento de fluxos contínuos podemos citar o Waze [1], que além de prover navegação via GPS, processa a movimentação de todos os usuários ativos para detectar de forma automática engarrafamentos e traçar a rota mais rápida até o ponto de destino.

Uma opção para o processamento de fluxos de dados é o uso de CEP (Processamento de Eventos Complexos), uma tecnologia de software para a análise dinâmica de grandes quantidades de dados ou fluxos de eventos em tempo (próximo de) real (RIOS, 2016).

[1] https://www.waze.com/

CEP é particularmente eficiente em detectar padrões em fluxos de eventos, característica que foi particularmente relevante para esse trabalho.

1.1 Motivação

Existem muitas situações onde se observa padrões de mobilidade que devem ser seguidos por unidades móveis. Dentre os cenários onde é necessário especificar padrões de mobilidade pode-se trazer, por exemplo, empresas de transporte urbano onde os condutores devem seguir trajetos específicos e dentro de faixas de tempo esperadas; o tráfego por vias com limites de velocidade, que podem inclusive variar de acordo com o tipo do veículo; o policiamento urbano no qual viaturas de polícia que devem atender a uma área especifica de uma cidade; empresas que trabalham com produtos perigosos onde há a necessidade da manutenção de uma distância segura entre máquinas que transportam esses produtos; e a chegada de um carregamento dentro do prazo esperado. É necessário portando estabelecer uma forma de especificar esses padrões, o que pode ser feito utilizando restrições de mobilidade como limites de velocidade e a demarcação de uma área de atuação, determinando assim o comportamento esperado das unidades móveis.

O não cumprimento de um desses padrões pode caracterizar de problemas de má utilização de recursos a problemas de segurança, o que justifica a utilidade de um sistema de software para monitorar UMs baseado em restrições de mobilidade. A detecção automática e em tempo real de violações a essas restrições pode ser feita por exemplo baseada no processamento de fluxos de dados de localização.

1.1.1 Caracterização do Problema

Baseado no explanado anteriormente, pode-se definir da seguinte forma o problema que este trabalho busca solucionar: como monitorar em tempo real o cumprimento de padrões de mobilidade especificados para UMs a partir de fluxos de dados de posicionamento (atualizações periódicas da posição geográfica ocupada pelas UMs).

Adotada a abordagem de representar esses padrões como restrições, faz-se necessário identificar quando ocorre o desobedecimento de uma delas. A detecção em tempo real de violações a restrições baseada na análise de fluxo de dados de localização bem como o desenvolvimento de uma plataforma para especificação dessas restrições traz consigo os seguintes obstáculos:

1. **Especificação de Restrições:** É necessário prover uma interface simples e abrangente para a especificação de restrições de mobilidade, abstraindo aspectos relativos à programação CEP.

2. **Fluxo de dados:** No modelo tradicional de software o processamento em feito em dados previamente armazenados (bancos de dados). Para a solução proposta nesse trabalho, é necessário que o processamento seja feito em um fluxo contínuo de dados, permitindo que resultados sejam obtidos em tempo próximo ao real.

3. **Escalabilidade:** A escalabilidade da solução com relação ao número de unidade móveis monitoradas deve ser levada em consideração durante o seu desenvolvimento, pois busca-se atender os requisitos de usuários e empresas de pequeno e grande porte. O fluxo de dados também aumenta acompanhando o número de UMs e torna-se mais difícil manter o processamento em tempo real.

1.2 Objetivos

1.2.1 Objetivo Geral

Esse trabalho busca elaborar uma abordagem de software para acompanhamento de unidades móveis e detecção autônoma da não conformidade de padrões de mobilidade estabelecidos para as mesmas em tempo próximo ao real.

1.2.2 Objetivos específicos

- Especificar um conjunto abrangente de restrições que englobe grande parte das situações comumente encontradas em ambientes onde o monitoramento de unidades móveis é importante. Uma abordagem abrangente permite a aplicação da solução proposta em diversos casos de uso.
- Desenvolver uma biblioteca para geração de regras CEP a partir de restrições de mobilidade.
- Avaliar a solução desenvolvida utilizando dados reais de movimentação e dados gerados por simulador de tráfego.

1.3 Metodologia:

Este projeto aborda o desenvolvimento de uma abordagem de software que permita instanciar e validar restrições de mobilidade de uma grande quantidade de unidades móveis utilizando dados de contexto, em particular, de localização. Para tanto, a solução proposta deve ser capaz de gerenciar e processar fluxos de dados em larga escala, permitindo ao sistema identificar a ocorrência de eventos complexos por meio do monitoramento de unidades móveis em tempo real. Os serviços disponibilizados pela solução desenvolvida permitirão automatizar o monitoramento de unidades móveis.

Para o desenvolvimento dos componentes de software da infraestrutura foram utilizados princípios de Métodos Ágeis de Desenvolvimento de Software (COCKBURN, 2002) que prevêem a construção de software de forma incremental, iterativa e adaptativa. Esta escolha se deve à necessidade de utilizar-se um processo ágil de desenvolvimento, que seja razoavelmente documentado e fácil de ser gerenciado. Sempre que possível, o desenvolvimento incremental do software foi baseado em técnicas de refatoração (FOWLER; BECK, 1999). Aspectos estruturais e comportamentais do software foram representados com diagramas UML (BOOCH, 2005) e, sempre que possível, foram adotados padrões arquiteturais e de projeto consagrados (SCHMIDT et al., 2013).

Apresentaremos aqui uma aplicação proposta para a ferramenta desenvolvida. juntamente com um cenário de teste utilizando essa aplicação proposta. Também foi realizada uma avaliação utilizando dados provenientes de simuladores de tráfego.

1.4 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma: O capítulo 2 contém a fundamentação teórica do trabalho que compreende conceitos gerais de CEP e a *engine* Esper CEP. O capítulo 3 traz a solução desenvolvida, introduzindo alguns conceitos definidos no âmbito desse trabalho e demonstrando como foram implementados os conceitos definidos no capítulo anterior. O capítulo 4 traz uma avaliação da solução desenvolvida utilizando dados reais e um simulador de tráfego. Por fim, o capítulo 5 conclui o trabalho retomando seus aspectos principais além de trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo serão trazidos alguns conceitos e tecnologias existentes que foram importantes durante a elaboração desse trabalho e necessários para uma completa compreensão da solução desenvolvida. Buscou-se desenvolver um sistema escalável, porém é notável que quanto maior o número de unidades móveis monitoradas maior será o fluxo de dados que o sistema precisará computar. A alternativa utilizada para solucionar esse problema é o emprego de CEP.

2.1 Processamento de Eventos Complexos (CEP)

O autor do termo CEP foi David Luckham em (LUCKHAM, 2002) onde ele declara que CEP é "um conjunto definido de ferramentas e técnicas para analisar e controlar séries complexas de eventos inter-relacionados que orientam sistemas modernos de informação distribuída". CEP é portanto um tipo de processamento de fluxos de eventos focado na detecção de padrões complexos dentro desses fluxos. Eventos representam uma ocorrência ou uma atualização do estado de uma entidade (e.g.: uma partida, uma chegada ou uma atualização de localização). Um evento pode agregar múltiplos atributos, um evento de estado do computador, por exemplo, pode conter o nível de utilização do processador e da memória principal, temperatura e outros atributos relevantes para a aplicação que consumirá esse evento. O que diferencia CEP do simples processamento de fluxos de eventos porém, é a geração de eventos complexos que posteriormente constituem novos fluxos de eventos a serem também processados.

A figura 1 contém um diagrama que demonstra de forma simplificada o funcionamento do processamento de eventos complexos. Uma característica fundamental do CEP é a capacidade de examinar **dados** provenientes de múltiplas **fontes**. Esses dados são encapsulados em **eventos simples**, também chamados de eventos atômicos ou eventos brutos. Os fluxos de eventos são examinados pelo que abstrairemos como **processador de eventos**, o qual conhece as regras (ou consultas) CEP previamente especificadas. As regras são o método disponibilizado pelas ferramentas CEP para especificar como deve ser feita a análise dos eventos. Regras CEP são semelhantes a consultas a um banco de dados, porém enquanto no banco as consultas são feita a informações já armazenadas, uma regra CEP é uma **consulta contínua**, ou seja, todo novo evento recebido é testado contra as regras que já haviam sido instanciadas antes da chegada do evento. O processador de eventos pode também gerar **eventos complexos** a partir da combinação das informações de mais de um evento simples; tal como um Evento de Incêndio gerado a partir de Eventos de Aumento de Temperatura e Eventos de Detecção de Fumaça. Após gerados, eventos

complexos são retroalimentados ao processador de eventos para serem também processados. Ao final, o resultado do processamento de eventos pode ser a tomada de **ações** automáticas e, ou a produção de **informação**.

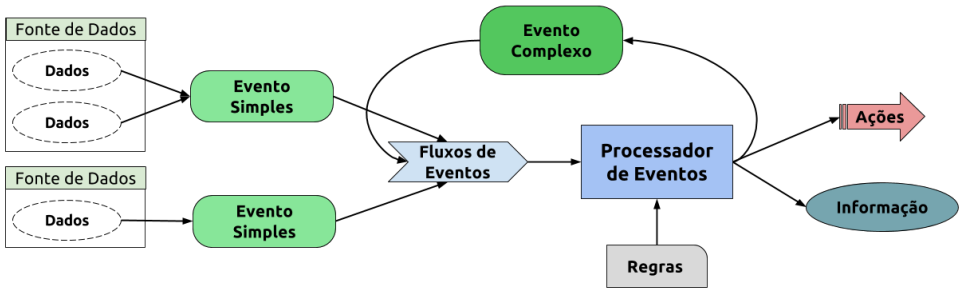

Figura 1 – Modelo de processamento de eventos complexos

2.2 Esper

A EsperTech é uma empresa especializada em análise de séries de eventos e processamento de eventos complexos que oferece as ferramentas de código aberto Esper (API Java) e NEsper (API .NET) para a produção de aplicações que utilizam CEP. No desenvolvimento desse projeto, foi utilizada a Esper para java em sua versão 5.5, disponível na página de downloads[1] do site da EsperTech. O ambiente Esper possui os seguintes componestes básicos:

- *Configuration*: Resposável pela configuração da API Esper. A configuração pode ser feita no código java, ou carregada de um arquivo XML.
- Eventos: Conforme descrito anteriormente;
- *EPStatements*: São as regras escritas em Esper EPL (Linguagem de Processamento de Eventos);
- *EPServiceProvider*: Funciona como o processador de eventos;
- *Listeners e Subscribers*: São objetos java que recebem o retorno de um EPStatement.

2.2.1 Configuração

A configuração da API Esper é necessária para realizar operações fundamentais como especificar tipos de eventos. A configuração pode ser carregada de um arquivo XML, ou feita no código java utilizando a classe com.espertech.client.Configuration ou com.espertech.client.ConfigurationOperations. Uma das vantagens de configurar a API

[1] http://www.espertech.com/download/

utilizando um arquivo XML é que o mesmo arquivo pode ser compartilhado entre mais de uma aplicação CEP. Enquanto a classe *Configuration* foi concebida para configuração antes do início do serviço, a classe *ConfigurationOperation* faz mudanças na configuração em tempo de execução. A configuração permite especificar os seguintes parâmetros:

- Acesso a um Banco de Dados, permitindo que as regras EPL acessem dados contidos no banco e, ou insiram dados novos no banco;
- Adição de tipos de eventos, possibilita adicionar classes específicas como eventos ou todas as classes de um pacote;
- Importar classes e pacotes para o ambiente Esper permitindo que os métodos declarados nessas classes sejam acessíveis nas regras CEP.

2.2.2 Representação de Eventos em Esper

Os tipos de eventos podem ser declarados antes ou durante o processamento e podem ser representados de 4 formas diferentes:

- Um POJO (Plain-Old Java Object) com métodos de get e set seguindo os padrões JavaBean;
- Um mapa (java.util.Map), onde cada entrada é uma propriedade do evento;
- Um vetor de Objetos, onde cada objeto é uma propriedade do evento;
- Representações baseadas em XML;
- Representações criadas como plugins utilizando a API de extensão disponibilizada pela ferramenta.

As propriedades de um evento são como os atributos de um objeto, ou seja elas possuem as informações sobre o evento. As propriedades de um evento podem ser:

- Simples: Propriedades que armazenam apenas um valor.
  Ex.: LeituraSensor.temperatura=50 (A temperatura lida por um sensor).
- Indexadas: Propriedades que armazenam vários valores, os quais são acessíveis através de um índice, de forma semelhante a um vetor.
  Ex.: CargaBateria.celula[2]=200 (A quantidade de amperes armazenada na terceira célula de uma bateria).

- Mapeadas: Propriedades que armazenam vários valores, os quais são acessíveis através de uma chave, de forma semelhante a um mapa.
  Ex.: ReposicaoEstoque.produto('aluminio')=5 (A quantidade de alumínio que foi reposta no estoque).

- Aninhadas: Propriedades dentro de outras propriedades.
  Ex.: EnvioEncomenda.destino.cep=65065-545 (O CEP de destino de uma encomenda.)

É possível ainda combinar os tipos acima.
Ex.: NovoFrame.pixel[70].intensidadecor('amarelo')=30 (A intensidade da cor amarela no pixel 70 de um frame).

2.2.3 A linguagem Esper EPL

Os EPStatements são os componentes do ambiente Esper que permitem a declaração de consultas a eventos, além de constituírem um dos meios para a definição de tipos de eventos. A linguagem utilizada nos EPStatements é uma EPL própria da API Esper, essa EPL não é semelhante a RAPIDE-EPL proposta por (LUCKHAM, 2002), ela é baseada nos padrões CQL (Linguagem de Consulta Contínua) o que a torna semelhante a linguagem SQL. Por exemplo:

```
SELECT temperatura FROM LeituraSensor
```

Seleciona a propriedade temperatura do último evento do tipo LeituraSensor recebido.
As cláusulas `SELECT`, `FROM`, `JOIN`, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `UPDATE` dos padrões SQL estão disponíveis . A cláusula `INSERT INTO` permite inserir dados de um tipo de evento em outro fluxo de eventos (ESPERTECH INC, 2016). Para oferecer as funções necessárias para o processamento de eventos, foram acrescentadas cláusulas e opções a Esper EPL não presentes no padrão SQL. As pricipais adições seram ligeiramente abordadas a seguir:

- Janelas de tempo e de número de eventos;
- Filtros;
- Especificação de padrões utilizando `PATTERN` e `MATCH RECOGNIZE`.

2.2.3.1 Janelas

Janelas tem o objetivo de permitir selecionar apenas os últimos n eventos ou os eventos recebidos em determinado intervalo de tempo. Os exemplos abaixo limitam a consulta aos últimos 10 eventos e aos eventos recebidos nos últimos 10 segundos respectivamente:

```
SELECT temperatura FROM LeituraSensor.win:length(10)
SELECT temperatura FROM LeituraSensor.win:time(10 sec)
```

Os exemplos acima retornam resultados sempre que um novo evento que se encaixa na consulta é processado. Isso significa que, a partir do décimo evento, a primeira consulta conterá nove eventos que já foram recuperados anteriormente e um evento que foi o último recebido. A opção `batch` permite que as consultas recuperem os eventos em lotes, a consulta aguarda até que o lote seja preenchido para retornar todos os eventos contidos nele de uma vez. Cada novo lote é preenchido apenas com eventos que não estavam presentes nos lotes anteriores. Abaixo, uma consulta que seleciona os eventos de 10 em 10 e outra que seleciona os eventos em faixas de tempo de 10 segundos não sobrepostas respectivamente:

```
SELECT temperatura FROM LeituraSensor.win:length_batch(10)
SELECT temperatura FROM LeituraSensor.win:time_batch(10 sec)
```

2.2.3.2 Filtros

Filtros funcionam de forma semelhante a cláusula `WHERE`, permitindo especificar valores (ou faixas de valores) para as propriedades de um evento nas consultas. Porém enquanto um filtro restringe os eventos que entraram na janela, as cláusulas `WHERE` especificam dentre os eventos da janela quais serão retornados pela consulta. Por exemplo, para selecionar eventos com temperatura acima de 100 graus podemos usar:

```
SELECT temperatura
FROM LeituraSensor(temperatura>100).win:length(10)
```

Seleciona os últimos 10 eventos com temperatura maior que 100, sempre retorna 10 eventos.

```
SELECT temperatura
FROM LeituraSensor.win:length(10)
WHERE temperatura>100
```

Seleciona dentre últimos 10 eventos, aqueles com temperatura maior que 100, pode retornar 10 ou menos eventos.

2.2.3.3 Padrões

Padrões são sequencias de eventos com algum relacionamento entre si. A API Esper disponibiliza duas formas para especificar padrões: `PATTERN` e `MATCH_RECOGNIZE`. Patterns utilizam operadores lógicos para descrever uma sequencia de eventos `and, or,` ->(seguido por). Além dos seguintes métodos para especificar repetições:

- `every`: Seleciona todos as sequências de eventos que se encaixam no padrão especificado;
- `every distinct`: Seleciona todas as sequências de eventos que se encaixam no padrão especificado e possuem uma pelo menos uma propriedade diferente das sequencias já retornadas;
- `[n]` retorna as n próximas sequencias que se encaixam no padrão;
- `until` retorna sequencias de eventos que se encaixam no padrão especificado até que uma expressão booleana seja verdadeira.

Ex.:

```
select * from pattern [every (A=LeituraSensor(temperatura>60) ->
B=LeituraSensor(radiacao>3.5) where timer:within(20 sec)]
```

Verifica se um evento onde a temperatura estava acima de 60 graus foi seguido por um evento onde a radiação estava acima de 3.5.

Já a especificação de padrões utilizando `MATCH_RECOGNIZE` é baseada em expressões regulares. Possuindo os operadores de:

- Agrupamento: `()`
- Quantificadores: `*, +, ?, {min, max}`
- Concatenação: `AB`
- Alternância: `|`

Ex.:

```
select * from LeituraSensor
match_recognize (
measures A.temperatura as a_temp, B.temperatura as b_temp, C.radiacao
as c_rad
pattern (A (B | C))
define
A as A.temperatura >= 50,
B as B.temperatura > A.temperatura,
C as C.radiacao >= 4)
```

Busca eventos com temperatura superior a 50 graus seguidos por um aumento de temperatura ou por um evento com radiação superior a 4 mSv.

3 A biblioteca MobCons

A abordagem adotada para solucionar o problema definido na motivação desse trabalho (item 1.1) foi a elaboração de um ferramental de software no formato de uma biblioteca java denominada como MobCons. Antes e durante o desenvolvimento da biblioteca foram definidos alguns conceitos teóricos de representação das informações de movimentação de UMs e das regras que definem as restrições de mobilidade aplicadas a essas UMs, esses conceitos serão apresentados na seção 3.1. A implementação da biblioteca será discutida na seção 3.2. Apresentamos ainda um cenário proposto de utilização da MobCons na seção 3.3.

3.1 Conceitos teóricos

Nessa seção introduz-se conceitos definidos no design da ferramenta MobCons. Esses conceitos definem as informações das unidades móveis que necessitam ser coletadas e processadas e como funciona a definição dos padrões esperados de mobilidade. um modelo para descrição de restrições de mobilidade bem como para definir as unidade móveis que submetem-se a tais restrições.

3.1.1 *MobilityEvent* (Evento de Mobilidade)

As informações de mobilidade de uma UM adquiridas em determinado momento são agregadas em eventos denominados *MobilityEvents*. Cada evento possui um conjunto de propriedades que serão posteriormente processadas de acordo com as restrições de mobilidade definidas. Essas propriedades estão ilustradas na figura 2 e serão explicadas a seguir.

- *MUID*: O identificador único de uma unidade móvel.
- *point*: Um ponto com as coordenadas geográficas da unidade no momento da geração do evento.
- *speed*: A velocidade da unidade.
- *timestamp*: Uma marca de tempo do momento em que o evento foi gerado
- *kindOfMU*: Contém o tipo da unidade (e.g. Pedestre, Ônibus, Trem).

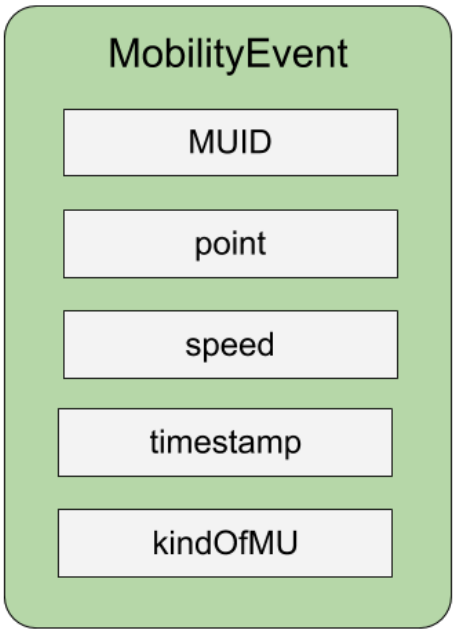

Figura 2 – O Evento de Mobilidade

3.1.2 Regras de mobilidade

Os padrões esperados de mobilidade são definidos em regras de mobilidade. Para delimitar regras de mobilidade é necessário conhecer os conceitos de contextos, restrições e cláusulas temporais. A figura 3 exibe o conjunto suportado de cada um desses conceitos os quais serão melhor esclarecidos a seguir. As regras de mobilidade foram divididas em três partes por dois motivos: para permitir a reutilização, por exemplo um mesmo contexto pode ser combinado com múltiplas restrições gerando múltiplas regras de mobilidade; e para facilitar a especificação de regras de mobilidade.

A primeira parte de uma regra de mobilidade é um **contexto**. Contextos servem para detalhar que unidades devem obedecer uma regra de mobilidade. O uso de um contexto é obrigatório para instanciar uma regra de mobilidade. Os seguintes tipos de contexto foram definidos e são suportados:

- *SingleMUContext:* Um contexto que representa uma única unidade móvel identificada por seu MUID.
- *GroupMUContext:* Um contexto que incorpora múltiplas unidades móveis identificadas por seu MUID.
- *KindOfMUContext:* Um contexto que inclui todas as unidades móveis de um tipo, o mesmo especificado no campo *KindOfMU* do evento de mobilidade.
- *AreaContext:* Usado para designar uma regra de mobilidade para uma área, determinando que todas as unidades móveis dentro da área devem obedecer a regra.

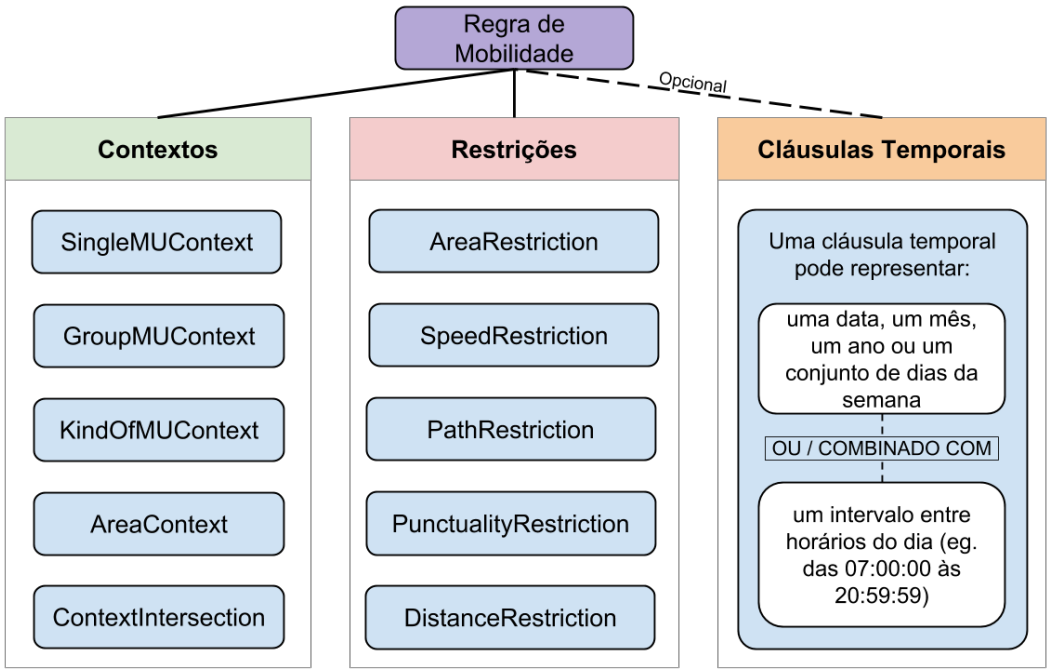

Figura 3 – Diagrama de Contextos, Restrições e Cláusulas Temporais

- *ContextIntersection:* Um contexto usado para combinar dois ou mais contextos. As unidades contidas nesse contexto são a intersecção dos contextos combinados. Por exemplo, pode-se combinar um *KindOfMUContext* para unidades do tipo "Ônibus"com um *AreaContext* contendo uma área escolar em um *ContextIntersection* e posteriormente utilizar esse *ContextIntersection* para determinar um limite máximo de velocidade de 40Km/h para ônibus na área escolar.

A segunda e mais importante parte de uma regra de mobilidade é uma **restrição**. Restrições representam o padrão esperado de mobilidade e também são obrigatórias para definir uma regra de mobilidade. Os seguintes tipos de restrição são suportados:

- *AreaRestriction:* Pode ser subdividida em restrições de acesso, determinando uma área que não deve ser acessada pelas unidades e restrições de permanência, determinando uma área que não deve ser deixada pelas unidades.
- *SpeedRestriction:* Pode delimitar um limite máximo de velocidade, um limite mínimo ou os dois.
- *PathRestriction:* Define um caminho que deve ser mantido pelas unidades. A unidades podem trafegar livremente por qualquer via do caminho sem violar a restrição.
- *PunctualityRestriction:* Assim como *PathRestriction* também definem um caminho para as unidades, porém utiliza pontos de controle que devem ser alcançados pelas unidades antes da marca temporal que é associada a cada um deles.

- *DistanceRestriction:* Estabelece uma distância mínima ou máxima entre dois contextos. Por exemplo se for utilizada para restringir uma distância máxima de 5Km entre o *GroupMUContext* A e o *GroupMUContext* B a restrição é violada quando qualquer uma das unidades de A distanciar-se mais de 5Km de qualquer unidade do grupo B. Se a mesma restrição estabelecesse uma distância mínima, ela seria violada quando qualquer unidade de A estivesse a menos de 5Km de qualquer unidade de B.

A última parte de uma regra de mobilidade é uma **cláusula temporal**. Cláusulas temporais são utilizadas para expressar quando a regra é válida. Cláusulas temporais são opcionais e quando nenhuma cláusula temporal é utilizada a regra será permanentemente válida. Cláusulas temporais possuem os seguintes tipos:

- Um conjunto ou intervalo de dias da semana, por exemplo terças-feiras, quintas-feiras e sábados ou de segunda à sexta.
- Uma data ou parte de uma data, por exemplo 2018/10/23, 2018/10, todo outubro (de qualquer ano), todo dia 23 (de qualquer mês, qualquer ano) ou até todo dia 23 de 2018.
- Um intervalo entre horários do dia (e.g., das 09:00:00 às 17:00:00 todo dia)
- Uma combinação de um intervalo entre horários e um dos outros tipos de cláusula temporal, por exemplo toda quarta-feira das 06:00:00 às 12:00:00 ou das 16:00:00 às 18:00:00 durante todo o mês de setembro de 2019.

3.2 Projeto e Implementação

Os conceitos definidos na seção anterior foram implementados no formato de uma biblioteca java denomindada MobCons. O objetivo dessa biblioteca é prover a programadores uma ferramenta para facilitar a declaração de regras de mobilidade, abstrair o processamento de eventos de mobilidade baseado nessas regras e disponibilizar *listeners* para receber o resultado desse processamento. A figura 4 exibe uma visão geral dos 3 pacotes que compõem a MobCons.

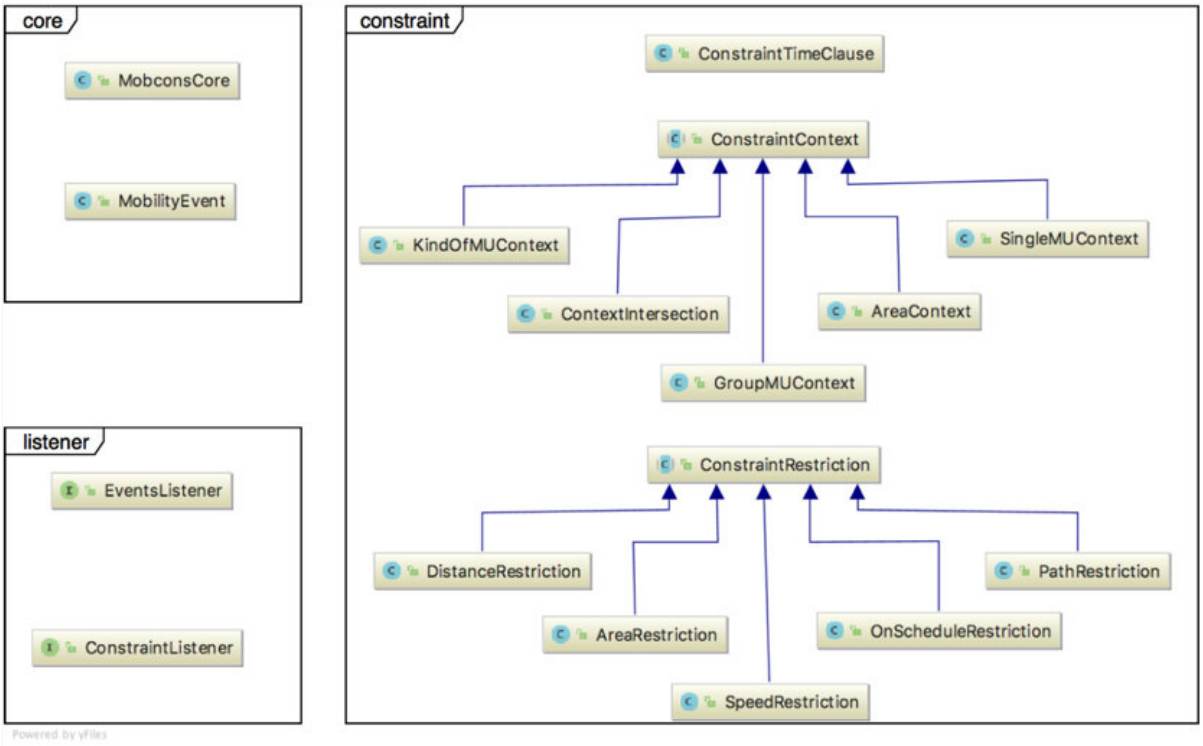

Figura 4 – Uma visão geral da biblioteca java MobCons

3.2.1 Pacote *core*

O pacote *core* contém duas classes: a *MobilityEvent* e a *MobConsCore*. A classe *MobilityEvent* é simplesmente uma representação do evento apresentado no item 3.1.1 no formato POJO (Plain Old Java Object). A classe *MobConsCore* (figura 5) é responsável por lidar com a ocorrência de eventos e possui métodos para registrar novas regras de mobilidade. Os atributos *epService* e *epAdm* são necessários para configurar e utilizar a *engine* Esper CEP, o *evtListener* é uma implementação da interface *EventsListener* (explicada a seguir) que pode ser configurada pelo programador para receber todos os eventos que serão processados. Os métodos da classe *MobConsCore* são:

- *init*: Configura o ambiente Esper CEP e o prepara para o recebimento de eventos.
- *setEvtListener*: determina um *listener* para receber os eventos que serão processados.
- *newMC*: usado para instanciar uma nova regra de mobilidade, possui uma assinatura com Cláusula Temporal (*TimeClause*) e outra sem já que esse parâmetro é opcional.
- *handleEvent*: envia novos eventos para a *engine* Esper e para o *EventsListener* se um foi determinado.

Figura 5 – A classe *MobConsCore*

3.2.2 Pacote *listener*

O pacote *listener* (figura 6) contém duas interfaces: a *EventsListener* para receber todo novo evento que será processado e a *ConstraintListener* para receber notificações das violações a uma regra de mobilidade. O método *newConstraintViolation* recebe a restrição desrespeitada e o conjunto de eventos que violou a regra de mobilidade.

Figura 6 – As interfaces do *MobConsCore*

3.2.3 Pacote *constraint*

O pacote *constraint* contém os contextos, restrições e cláusulas temporais que podem compor uma regra de mobilidade, os mesmos especificados no item 3.1.2. Todos os contextos (figura 7) herdam da classe abstrata *ConstraintContext* essa classe provê um ID único para cada contexto e contém o método abstrato *getWhereCondition* que é utilizado durante a geração de regras CEP. O *SingleMUContext* assim como o *GroupMUContext* inspeciona o MUID de cada evento para checar se ele se encaixa no contexto, similarmente o *KindOfMUContext* inspeciona o atributo KindOfMU de cada evento. O *ContextIntersection* recebe múltiplos contextos em seu construtor e inclui somente eventos que se encaixam em todos esses contextos. O *AreaContext* inclui os eventos cujas coordenadas (*point*) encontram-se dentro de uma área especificada. Essa área pode ser de três tipos:

1. Circular: Especificada utilizando um ponto central e um valor decimal para o raio do círculo em quilômetros.

2. Retangular: Especificada utilizando dois vértices opostos de um retângulo.

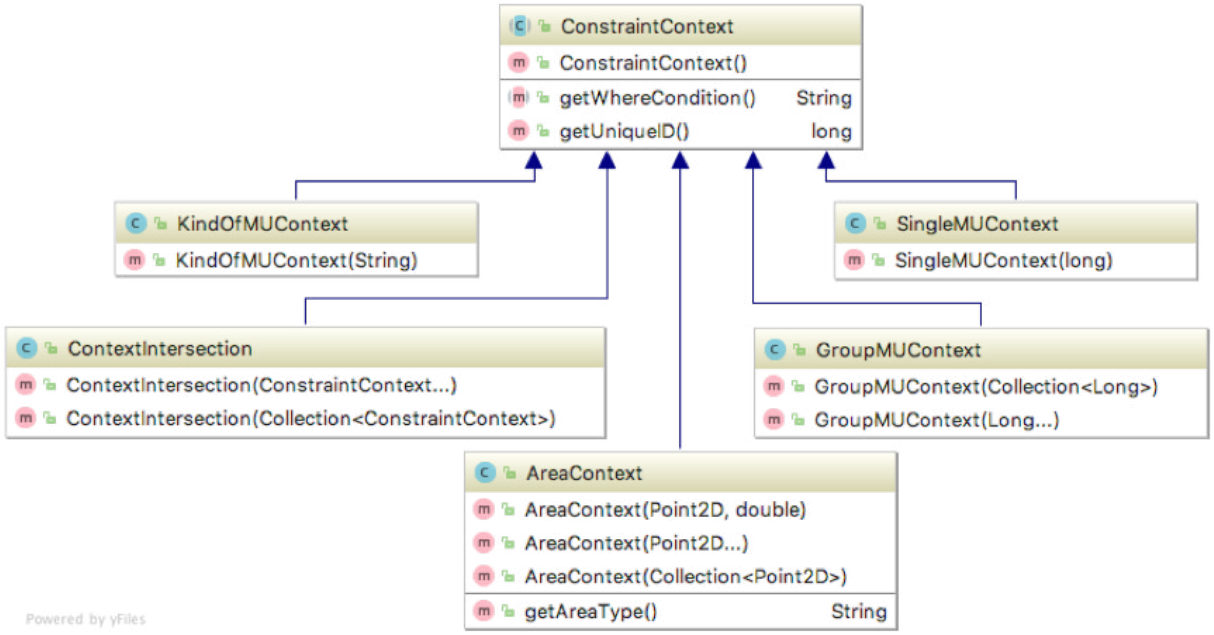

Figura 7 – A classe abstrata *ConstraintContext* e suas classes filhas

3. Poligonal: Especificada utilizando a sequência (em sentido horário ou anti-horário) de todos os vértices do polígono. Pode representar qualquer polígono convexo desde que ele seja composto apenas de arestas retas.

As restrições apresentadas anteriormente nesse trabalho foram implementadas como filhas da classe abstrata *ConstraintRestriction* (figura 8). Essa classe, similarmente a classe *ConstraintContext*, provê um ID único e possui o método *generateEPL* para a geração de regra CEP. Esse método é onde as regras CEP são realmente geradas, o método *newMC* da classe *MobConsCore* chama o método *generateEPL* da restrição para adquirir as regras CEP e então as instancia na *engine* Esper. Essa abordagem foi adotada pois algumas restrições necessitam de mais de uma regra CEP para funcionar corretamente. A *SpeedRestriction* suporta limites máximos e / ou mínimos de velocidade. Os tipos de área suportados pela *AreaRestriction* são os mesmos suportados no *AreaContext*, porém a *AreaRestriction* necessita de um booleano em seu construtor para especificar se as unidades móveis não devem entrar na área ou não devem deixar a área. A *DistanceRestriction* estabelece uma distância mínima ou máxima entre dois contextos, portanto necessita que um dos contextos seja especificado no seu construtor e outro na chamada do método *newMC*. A *PathRestriction* recebe um conjunto de pontos (coordenadas). O caminho a ser mantido é então o conjunto de segmentos de reta que ligam esses pontos na sequência em que foram especificados. A *PunctualityRestriction* é um tipo de *PathRestriction* porém a cada ponto é associado uma marca de tempo, além de não desviar do caminho as unidade devem atingir cada ponto antes que a sua marca de tempo expire.

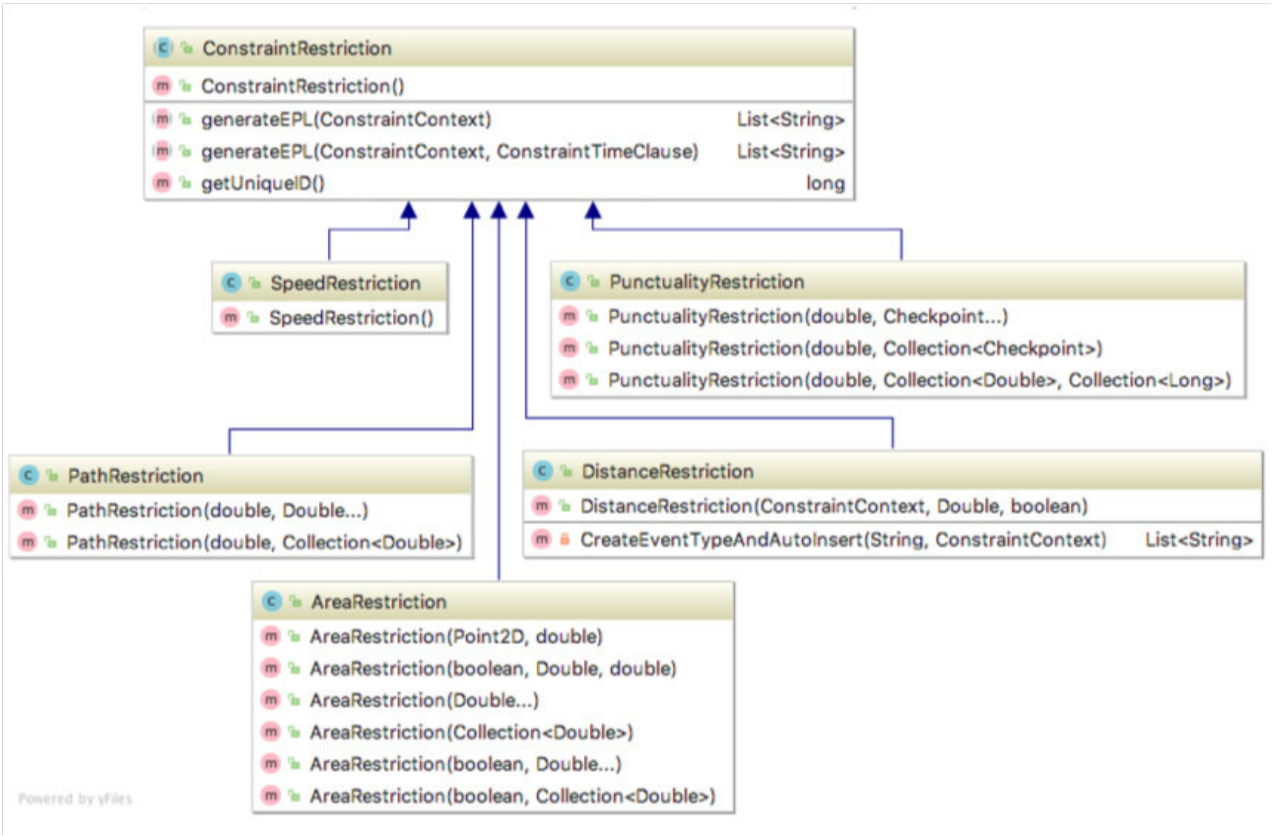

Figura 8 – A classe abstrata *ConstraintRestriction* e suas classes filhas

As cláusulas temporais foram todas definidas dentro da classe *TimeClause* (figura 9), ao invés de disponibilizar construtores, essa classe disponibiliza um conjunto de métodos estáticos que retornam instâncias de si própria. Apesar de tornar a definição de cláusulas temporais diferente da definição de contextos e restrições, essa abordagem permite utilizar métodos cujos nomes exprimem melhor o tipo de cláusula temporal que retornam.

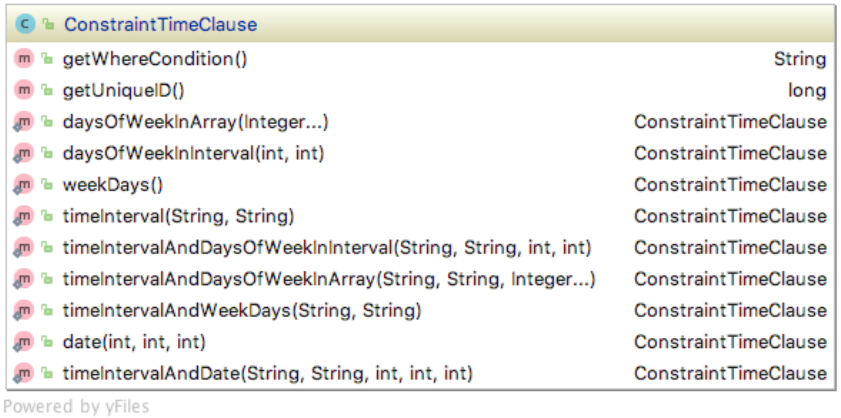

Figura 9 – A classe *TimeClause*

3.2.4 Exemplos de regras CEP geradas

Após combinar um contexto, uma restrição e opcionalmente uma cláusula temporal para estabelecer uma regra de mobilidade, esse regra será convertida para regra(s) CEP em Esper EPL (Linguagem de Processamento de Eventos). A seguir dois exemplos de regras CEP geradas utilizando a ferramenta:

1. A unidade móvel de MUID 2001 não deve exeder 80 Km/h de segundas-feiras às sextas-feiras entre as 8h e as 18h.

   - Contexto: SingleMUContext(muID=1001)
   - Restrição: SpeedRestriction(maxSpeed=80.0)
   - CláusulaTemporal: TimeClause.timeIntervalAndDaysOfWeekInInterval(beginTime="08:00:00", endTime="18:00:00", beginDay=2, endDay=6)

```
select * from MobilityEvent as evt where evt.speed>80.0
and evt.umID=2001 and (evt.timestamp.getDayOfWeek() in (2:6))
and (evt.timestamp.between (evt.timestamp.withTime(08,00,00,0),
evt.timestamp.withTime(18,00,00,0)))
```

2. Caminhões carregando produtos perigosos constituem o tipo de unidade "CargaPerigosa"e devem manter uma distância mínima de 20 metros (0,02 Km) dos ônibus para condução de funcionários que caracterizam o tipo de unidade "TransporteFuncionarios".

   - ContextoA: KindOfMUContext(kind="TransporteFuncionarios")
   - ContextoB: KindOfMUContext(kind="CargaPerigosa")
   - Restrição: DistanceRestriction(cntxtA=ContextoA, distance=0.02, distanceIsMax=false)

```
insert into Event1 select * from MobilityEvent as evt
where evt.kindOfMU="TransporteFuncionarios"

insert into Event2 select * from MobilityEvent as evt
where evt.kindOfMU="CargaPerigosa"

select evt.muID as muID, evt.point as point,
evt.speed as speed, evt.timestamp as timestamp,
evt.kindOfMU as kindOfMU from
pattern[every a=Event1 -> every evt=Event2
where timer:within(1.5 sec)]
where (((a.point.x - evt.point.x)*(a.point.x - evt.point.x)
+(a.point.y - evt.point.y)*(a.point.y - evt.point.y))
<=8.087793508722446E-9)
```

3.3 Cenário de Aplicação Proposto

Como aplicação proposta da biblioteca desenvolvida foi estabelecido sistema baseado em smartphones para aquisição de informações de mobilidade e geração de eventos e um servidor na nuvem que processa esses eventos utilizando a biblioteca MobCons. A arquitetura proposta desse sistema está demonstrada na figura 10. O primeiro componente é um **aplicativo android** esse aplicativo utiliza o middleware M-Hub/CDDL para aquisição e distribuição das informações de mobilidade. O M-Hub/CDDL disponibiliza uma interface para adquirir informações de sensores o que já provê para o aplicativo a posição geográfica e velocidade da unidade móvel. O aplicativo então gera eventos de mobilidade utilizando essas informações juntamente com o identificador e o tipo da unidade os quais são configuráveis pelo usuário. O middleware M-Hub/CDDL adota o modelo publicador/subscritor e necessita de um ***broker* MQTT** (Transporte de telemetria do serviço de enfileiramento de mensagens) para permitir a troca de mensagens entre nós. O próximo componente é um **servidor na nuvem** que recebe os eventos e os analisa utilizando uma aplicação java que aplica a biblioteca MobCons para processar os eventos. A interface de monitoramento recebe os eventos e as violações de regras utilizando os recursos disponibilizados no pacote *listener* da biblioteca desenvolvida. Um programador necessita declarar as regras de mobilidade utilizando as classes do pacote *constraint*.

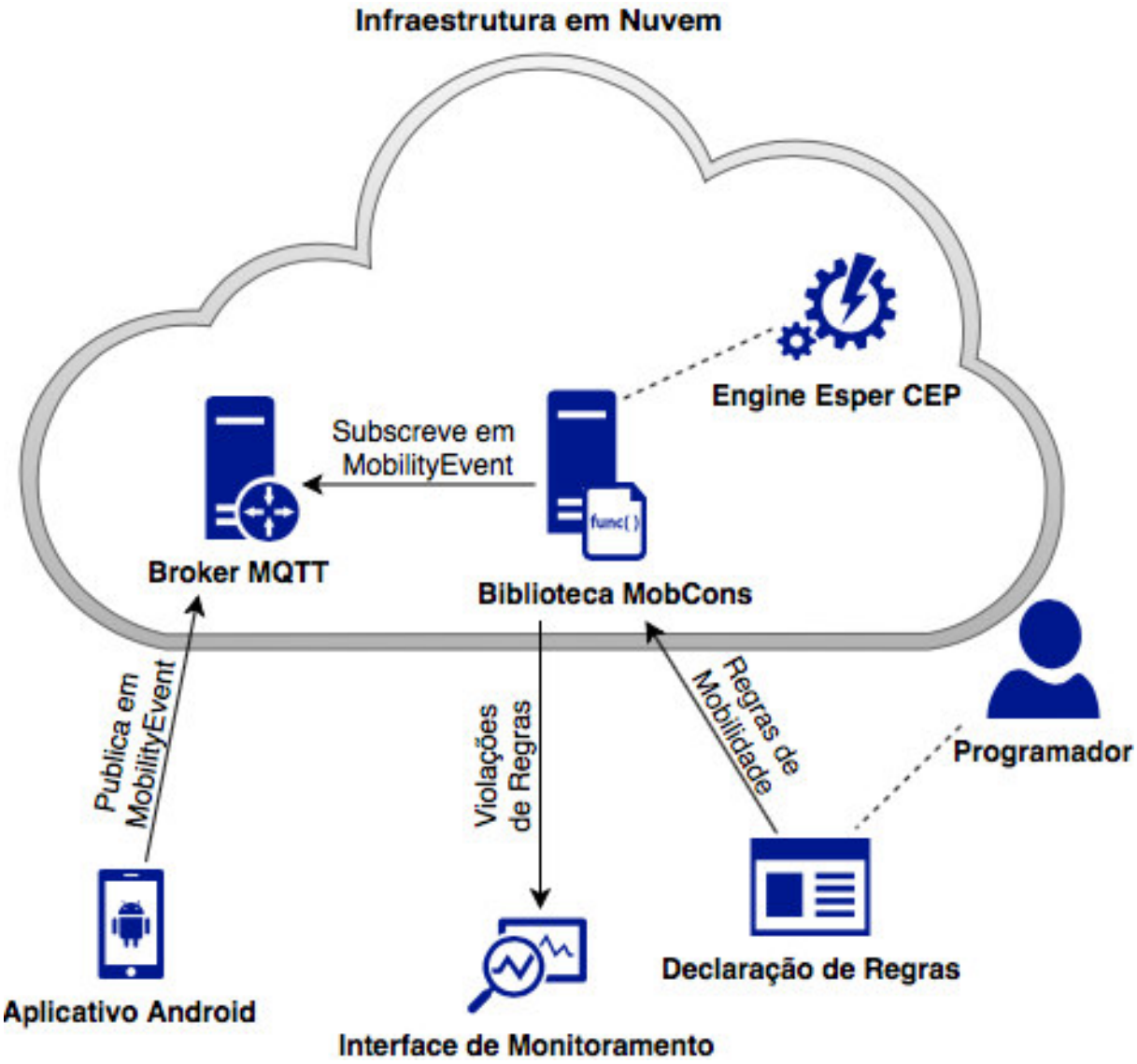

Figura 10 – Arquitetura proposta

4 Avaliação

A avaliação foi dividida em duas partes, uma utilizando dados reais e outra com uso de um simulador de tráfego. Para a avaliação com dados reais, o cenário proposto na seção 3.3 foi implementado. Na avaliação com dados de simulador foi utilizado o simulador SUMO (Simulação de Mobilidade Urbana).

4.1 Avaliação com Dados Reais

Para o teste com dados reais, foi determinado um cenário simples com o objetivo de verificar se o sistema proposto é funcional:

Um estudante que chega na Universidade e se dirige ao Restaurante Universitário, faz um retorno próximo ao Centro de Ciências Sociais (CCSO) e estaciona no Centro de Ciências Exatas e Tecnologia (CCET) onde assistirá aulas durante a tarde. O estudante deve respeitar o limite de velocidade das vias da Universidade que é de 30Km/h.

Figura 11 – O trajeto do Estudante. Fonte: Google Maps

Testou-se então se o sistema detectaria a passagem do estudante pelos pontos marcados no mapa (Figura 11) e velocidades acima do limite. Para isso foram criadas restrições de área circular proibida, tomando como centro cada um dos pontos marcados e raio de 30 metros e uma restrição de velocidade máxima de 30 Km/h. O aplicativo foi configurado para que o tipo da unidade móvel fosse CAR e ao trafegar pelos pontos marcados o sistema reportou as violações. O teste do limite de velocidade foi feito fora da Universidade, na Avenida dos Portugueses onde o limite de velocidade é de 60Km/h e o sistema detectou as violações como esperado.

4.2 Avaliação com Simulador

Uma outra avaliação da biblioteca MobCons foi feita utilizando dados gerados em simulador. Foi adotado o SUMO[1] (Simulação de Mobilidade Urbana) e a biblioteca python TraCI (Interface de Controle de Tráfego) (WEGENER et al., 2008) para controlar a simulação. O SUMO é um simulador de tráfego de código aberto desenvolvido pelo Instituto de Pesquisas em Transporte do Centro Aeroespacial Alemão em parceria com Centro de Informática Aplicada de Colônia. Ele foi adotado pois além de permitir importar mapas adquiridos no OpenStreetMaps [2], possui um amplo escopo para definição de rotas e características de veículos. A biblioteca TraCI permite acompanhar e intervir em uma simulação em execução no SUMO. Ela oferece métodos para por exemplo instanciar novas unidades móveis, designar uma rota para uma unidade ou até controlar semáforos. Na avaliação buscou-se testar as restrições *PathRestriction*, *PunctualityRestriction* e *DistanceRestriction* pois são as restrições mais complexas compreendidas nesse trabalho e não constaram no estudo de caso previamente apresentado.

Para testar a funcionalidade da *PathRestriction* foi definida uma rota principal e uma rota com um pequeno desvio da rota principal. A biblioteca TraCI foi utilizada para em alguns momentos da simulação substituir a rota de uma unidade da rota principal para a rota com desvio, tais momentos foram guardados em um registro. Foi então definida uma *PathRestriction* utilizando a biblioteca desenvolvida para detectar esses desvios. Após cada momento de desvio presente no registro, a biblioteca detectou uma série eventos que violavam a restrição de caminho. A figura 12 demonstra uma dessas detecções.

```
*------- Constraint Violation detected! [start] -------*
Restriction: PathRestriction{path=[Point2D.Double[3274.83, 1533.1],
Point2D.Double[3335.64, 1481.42], Point2D.Double[3346.98, 1461.9],
Point2D.Double[3346.23, 1438.88], Point2D.Double[3327.51, 1329.87],
Point2D.Double[3322.74, 1276.75], Point2D.Double[3314.27, 1264.03],
Point2D.Double[3040.27, 1289.29], Point2D.Double[3038.63, 1364.73],
Point2D.Double[3256.62, 1535.06]], threshold=25.0}
Event[0]: MobilityEvent{muID=1001, point=Point2D.Double[3306.06, 1426.01],
speed=26.676, timestamp=1530684699, kindOfMU='shuttle'}
*------- Constraint Violation detected! [end] -------*
```

Figura 12 – Exemplo de Violação a *PathRestriction*

Para o teste da *PunctualityRestriction*, foram definidos dois tipos de veículos, um com uma velocidade máxima de 30Km/h e outro com uma velocidade máxima de 16Km/h e foi utilizada a rota principal definida na *PathRestriction*. Para alguns pontos da rota foram definidas marcas de tempo de modo que os veículos com velocidade máxima superior

[1] http://sumo.sourceforge.net/
[2] https://www.openstreetmap.org/

alcançariam cada ponto antes que sua marca de tempo expire e os veículos com velocidade inferior não conseguiriam realizar o mesmo. Por volta da segunda metade da rota, os veículos de velocidade inferior começaram a apresentar atrasos. A figura 13 contém um exemplo detecção desses atrasos a partir da biblioteca MobCons.

```
*------- Constraint Violation detected! [start] -------*
Restriction: PunctualityRestriction{path=[Checkpoint{point=Point2D.Double[3274.83,
1533.1], timestamp=1530669699}, Checkpoint{point=Point2D.Double[3335.64, 1481.42],
timestamp=1530679699}, Checkpoint{point=Point2D.Double[3346.98, 1461.9],
timestamp=1530684699}, Checkpoint{point=Point2D.Double[3346.23, 1438.88],
timestamp=1530687699}, Checkpoint{point=Point2D.Double[3327.51, 1329.87],
timestamp=1530702699}, Checkpoint{point=Point2D.Double[3322.74, 1276.75],
timestamp=1530709699}, Checkpoint{point=Point2D.Double[3314.27, 1264.03],
timestamp=1530710699}, Checkpoint{point=Point2D.Double[3040.27, 1289.29],
timestamp=1530730699}, Checkpoint{point=Point2D.Double[3038.63, 1364.73],
timestamp=1530738699}, Checkpoint{point=Point2D.Double[3256.62, 1535.06],
timestamp=1530837699}], threshold=25.0}
Event[0]: MobilityEvent{muID=1001, point=Point2D.Double[3294.25, 1406.91],
speed=28.008, timestamp=1530731699, kindOfMU='shuttle'}
*------- Constraint Violation detected! [end] -------*
```

Figura 13 – Exemplo de Violação a *PunctualityRestriction*

O SUMO provê dentro da especificação de veículos um parâmetro chamado *minGap* que determina a distância miníma a ser mantida do veículo a frente. Para o teste da *DistanceRestriction* foram definidos dois tipos de veículos um com *minGap* de 6 metros e outro com um *minGap* de 3 metros. Durante a simulação veículos não mantém uma distancia constante do veículo a frente, eles se aproximam e se afastam a medida que passam pelas curvas e retas no trajeto. A biblioteca TraCI foi utilizada para armazenar distância entre os veículos a cada passo da simulação em um registro. Foi então definida uma *DistanceRestriction* entre os veículos para detectar os momentos em que os veículos se aproximassem mais de 5 metros. Nem todos os momentos em quem foi registrada pela TraCI uma distância menor que 5 metros foram detectados pela biblioteca desenvolvida. Isso ocorre pois na implementação feita nesse trabalho são necessários dois eventos com localização próxima para que uma violação a restrição seja detectada. A figura 14 demonstra um dos momentos em que a biblioteca MobCons detectou uma violação a restrição.

```
*------- Constraint Violation detected! [start] -------*
Restriction:DistanceRestriction{cntxtA=SingleMUContext{muID=1001},
distance=0.05, distanceIsMax=false}
Event[0]: MobilityEvent{muID=1002, point=Point2D.Double[3269.95,
1343.29], speed=29.124, timestamp=1530636699, kindOfMU='shuttle'}
*------- Constraint Violation detected! [end] -------*
```

Figura 14 – Exemplo de Violação a *DistanceRestriction*

4.3 Trabalhos Relacionados

Foram encontrados alguns trabalhos relacionados com objetivos semelhantes os quais serão comentados nessa seção. Dentre os trabalhos encontrados, (AL-KHEDHER, 2012) e (BEHZAD et al., 2014) demonstram apenas a capacidade de obter a localização de veículos e a repassar para um nó central que exibe essa localização em um mapa. Esses trabalhos não fazem nenhum processamento das informações de mobilidade e portanto a detecção de quaisquer irregularidades na movimentação dos veículos não é feita de forma automática. Os trabalhos (AL-TAEE; KHADER; AL-SABER, 2007) e (JOY et al., 2016) porém, já possuem a funcionalidade de detecção de quebras de limites de velocidade. O trabalho de (PUNETHA; MEHTA, 2014) oferece detecção de quando uma unidade monitorada sai de uma área especificada, e o trabalho de (ALMOMANI et al., 2011) oferece tanto detecção de quebra de limites de velocidade quanto detecção de saída de uma área especificada. Também foi encontrado o trabalho de (OLIVEIRA et al., 2013) que permite detecção de desvio de rota. As tabelas 1 e 2 dispõem os trabalhos relacionados e suas principais funcionalidades no monitoramento de unidades móveis.

Trabalho	**Exibe a localização em um mapa**	**Monitora limites de velocidade**	**Monitora a saída de uma área**
Al-Khedher	Sim	Não	Não
Behzad et. al.	Sim	Não	Não
Al-Taee;Khader; Al-Saber	Sim	Sim	Não
Puneta; Mehta	Não	Não	Sim
Almomani et. al.	Sim	Sim	Sim
Oliveira et. al.	Sim	Não	Sim
Este trabalho	Não	Sim	Sim

Tabela 1 – Resumo esquemático de trabalhos relacionados (parte 1)

Trabalho	**Monitora desvios de rota**	**Regras para unidades específicas**	**Regras para áreas específicas**
Al-Khedher	Não	Não	Não
Behzad et. al.	Não	Não	Não
Al-Taee;Khader; Al-Saber	Não	Não	Não
Puneta; Mehta	Não	Sim	Não
Almomani et. al.	Não	Sim	Não
Oliveira et. al.	Sim	Sim	Não
Este trabalho	Sim	Sim	Sim

Tabela 2 – Resumo esquemático de trabalhos relacionados (parte 2)

A maioria desses trabalhos são focados em monitorar veículos, com exceção do trabalho de Puneta e Mehta que foi desenvolvido para rastreamento de pessoas e o de Oliveira que foi desenvolvido para rastreamento de mercadorias. Também é notável que esses trabalhos associam as restrições de mobilidade (limites de velocidade, permanência em uma área ou rota) com as unidades monitoradas, com exceção do trabalho de Joy et al. que associa as restrições a uma área específica. Na solução abordada nesse trabalho, foi desenvolvido um sistema mais flexível que permita rastrear pessoas, veículos, máquinas ou qualquer outra coisa que possa carregar um smartphone. Essa solução cobre os limites de velocidade (máximo e / ou mínimo), desvio de rota e entrada em uma área restrita ou deixa uma área projetada, além de detecção de atrasos e determinação de distâncias mínimas ou máximas. Além disso, as restrições de mobilidade criadas podem estar associadas a uma unidade móvel, um grupo de unidades ou uma área. Nota-se porém que no escopo desse trabalho não foi desenvolvida uma ferramenta que exiba a localização em um mapa das UMs monitoradas.

5 Conclusão e Trabalhos Futuros

5.1 Retrospectiva do Trabalho

Esse trabalho compreendeu o desenvolvimento de uma abordagem de software para a descrição de regras de mobilidade, bem como processar fluxos contínuos de dados de localização baseado nessas regras. Como fundamentação teórica foram apresentados conceitos gerais de CEP, bem como a *engine* Esper CEP que foi utilizada no processamento de eventos de mobilidade. Antes e durante o desenvolvimento do software foram estabelecidos alguns conceitos incluindo uma metodologia de como definir regras de mobilidade baseada em contextos, restrições e cláusulas temporais. A abordagem de software desenvolvida consistiu de uma biblioteca java chamada MobCons cujos aspectos de implementação foram discutidos na seção 3.2 desse trabalho. Como cenário proposto de aplicação da biblioteca foi apresentado um estudo de caso onde utilizou-se smartphones para adquirir informações de localização e gerar eventos de mobilidade os quais foram processados em um servidor na nuvem utilizando a biblioteca desenvolvida. A avaliação foi dividida em duas partes, uma com uso de dados reais e outra utilizando um simulador de tráfego. Os objetivos estabelecidos foram alcançados porém não foi feita uma avaliação da escalabilidade da biblioteca desenvolvida.

5.2 Trabalhos futuros

Como trabalhos futuros, pretende-se escrever o javadoc e um manual para a biblioteca desenvolvida. A maior parte das classes e métodos da biblioteca já possuem um javadoc que foi escrito no próprio código, exceto alguns construtores de classes do pacote *constraint*. Após completa a escrita do javadoc no código, pretende-se gerar uma versão do javadoc no formato de página web. O manual terá como objetivo introduzir os conceitos apresentados na seção 3.1 desse trabalho, além de exemplos demonstrativos da utilização da biblioteca. Pretende-se também realizar testes mais complexos utilizando o simulador SUMO com o objetivo de avaliar a escalabilidade da solução desenvolvida e posteriormente escrever um artigo científico do presente trabalho. Por fim foi notada a importância de uma ferramenta de visualização que exiba a localização das unidades móveis em um mapa e eventuais violações as regras de mobilidade.

Referências

AL-KHEDHER, M. A. Hybrid gps-gsm localization of automobile tracking system. *arXiv preprint arXiv:1201.2630*, 2012. Citado na página 33.

AL-TAEE, M. A.; KHADER, O. B.; AL-SABER, N. A. Remote monitoring of vehicle diagnostics and location using a smart box with global positioning system and general packet radio service. In: *2007 IEEE/ACS International Conference on Computer Systems and Applications*. [S.l.: s.n.], 2007. p. 385–388. ISSN 2161-5322. Citado na página 33.

ALMOMANI, I. M. et al. Ubiquitous gps vehicle tracking and management system. In: *2011 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*. [S.l.: s.n.], 2011. p. 1–6. Citado na página 33.

BEHZAD, M. et al. Design and development of a low cost ubiquitous tracking system. *Procedia Computer Science*, v. 34, n. Supplement C, p. 220 – 227, 2014. ISSN 1877-0509. The 9th International Conference on Future Networks and Communications (FNC'14)/The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14)/Affiliated Workshops. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050914008692>. Citado na página 33.

BOOCH, G. *The unified modeling language user guide*. [S.l.]: Pearson Education India, 2005. Citado na página 11.

CARNEY, D. et al. Monitoring streams: A new class of data management applications. In: *Proceedings of the 28th International Conference on Very Large Data Bases*. Hong Kong, China: VLDB Endowment, 2002. (VLDB '02), p. 215–226. Citado na página 8.

COCKBURN, A. *Agile software development*. [S.l.]: Addison-Wesley Boston, 2002. v. 177. Citado na página 11.

ESPERTECH INC. *Esper Reference*. [S.l.], 2016. Version 5.5.0. Citado na página 15.

FOWLER, M.; BECK, K. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 1999. Citado na página 11.

JOY, S. P. et al. A novel security enabled speed monitoring system for two wheelers using wireless technology. In: *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*. [S.l.: s.n.], 2016. p. 1–7. Citado na página 33.

LUCKHAM, D. *The power of events*. [S.l.]: Addison-Wesley Reading, 2002. v. 204. Citado 2 vezes nas páginas 12 e 15.

OLIVEIRA, R. R. et al. Swtrack: An intelligent model for cargo tracking based on off-the-shelf mobile devices. *Expert Systems with Applications*, v. 40, n. 6, p. 2023 – 2031, 2013. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417412011359>. Citado na página 33.

PUNETHA, D.; MEHTA, V. Protection of the child/ elderly/ disabled/ pet by smart and intelligent gsm and gps based automatic tracking and alert system. In: *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. [S.l.: s.n.], 2014. p. 2349–2354. Citado na página 33.

RIOS, L. E. T. *An Energy-aware IoT Gateway, with Continuous Processing of Sensor Data*. 2016. Citado na página 8.

SCHMIDT, D. C. et al. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*. [S.l.]: John Wiley & Sons, 2013. v. 2. Citado na página 11.

WEGENER, A. et al. Traci: An interface for coupling road traffic and network simulators. In: *Proceedings of the 11th Communications and Networking Simulation Symposium*. New York, NY, USA: ACM, 2008. (CNS '08), p. 155–163. ISBN 1-56555-318-7. Disponível em: <http://doi.acm.org/10.1145/1400713.1400740>. Citado na página 31.