UNIVERSIDADE FEDERAL DO MARANHÃO

Curso de Ciência da Computação

André Filipe Sousa Barreto

Uso do simulador SUMO na avaliação de aplicações de mobilidade em cidades inteligentes

São Luís

2023

André Filipe Sousa Barreto

Uso do simulador SUMO na avaliação de aplicações de mobilidade em cidades inteligentes

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.

Orientador: Prof. Dr. Francisco José da Silva e Silva

São Luís

2023

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

Sousa Barreto, André Filipe.
Uso do simulador SUMO na avaliação de aplicações de mobilidade em Cidades Inteligentes / André Filipe Sousa Barreto. - 2023.
57 f.

Orientador(a): Francisco José da Silva e Silva.
Curso de Ciência da Computação, Universidade Federal do Maranhão, UFMA, 2023.

1. Cidades Inteligentes. 2. Mobilidade. 3. SUMO. I. da Silva e Silva, Francisco José. II. Título.

André Filipe Sousa Barreto

Uso do simulador SUMO na avaliação de aplicações de mobilidade em cidades inteligentes

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 _____ em São Luís, 18 de dezembro de 2023:

Prof. Dr. Francisco José da Silva e Silva
Orientador

Prof. Dr. Primeiro Membro
Examinador

Profa. Dra. Segundo Membro
Examinador

São Luís
2023

Agradecimentos

Agradeço, primeiramente, a Deus e à Nossa Senhora, pois nunca me faltaram forças para lidar com todas as adversidades encontradas nessa jornada.

À minha mãe, por todo o esforço, suporte e educação que me foram concedidos desde meus primeiros dias de vida. Ao meu pai, por sempre se doar ao máximo para me oferecer todas as condições necessárias para chegar até aqui, e por ser o modelo de homem que quero me tornar.

À minha companheira de vida, Lays Eloi, por compartilhar comigo todos os momentos bons e ruins nessa árdua rotina de graduação, e ao meu filho, Thomás, por sua inocente e contagiante felicidade. Vocês são a principal razão por eu nunca querer desistir.

Agradeço aos meus muitos irmãos, pois vocês sempre serão parte integrante de minha personalidade. Aos meus avôs e avó Nilza, que mesmo não mais em vida, compõem minha motivação. À minha avó Dalva, pelo imenso carinho, e aos demais familiares, por também fazerem parte de muitos momentos especiais.

Agradeço também, com carinho, aos seletos amigos que fiz durante a graduação, pois vocês foram essenciais para que eu me mantivesse focado e determinado a continuar. Dividimos muitos momentos especiais, e que com certeza ficarão marcados.

Por fim, agradeço a todos os professores que fizeram parte da minha jornada acadêmica, o conhecimento que me foi passado será fundamental nas próximas etapas de minha vida. Em especial, agradeço ao Prof. Dr. Francisco José da Silva e Silva, por toda a paciência e orientação, me motivando sempre a buscar novos conhecimentos e a continuar meu caminho de estudante.

"Lembrete
Se procurar bem você acaba encontrando.
Não a explicação (duvidosa) da vida,
Mas a poesia (inexplicável) da vida."

Carlos Drummond de Andrade, em *"Lembrete"*

Resumo

As cidades inteligentes se apresentam como uma alternativa de melhora frente aos inúmeros problemas trazidos pela acelerada urbanização. Dessa forma, novas aplicações estão sendo desenvolvidas para atuarem nesse contexto, em áreas como mobilidade urbana, saúde, infraestrutura, gerenciamento de recursos, dentre outras. Assim, os *softwares* de simulação surgem como uma ferramenta necessária para o desenvolvimento de soluções eficientes, uma vez que possibilitam o planejamento e teste de aplicações em um ambiente virtualizado, diminuindo custos e poupando tempo de implementação em ambientes reais. Desse modo, o *Simulation of Urban MObility* (SUMO) é um exemplo de simulador que pode ser utilizado, pois disponibiliza uma série de elementos e funções que possibilitam o desenvolvimento de ambientes de trânsito de forma próxima à realidade. Este trabalho teve por objetivo investigar o uso do SUMO para criação de aplicações de mobilidade em cidades inteligentes, através de cenários controlados e da condução de experimentos, além de realizar uma análise da integração desse simulador com o *middleware Context Data Processing Orchestrator* (CDPO), uma infraestrutura para processamento de fluxo em múltiplas camadas por aplicações de cidades inteligentes, a fim de verificar seu desempenho com plataformas externas. Os resultados obtidos apontam que o SUMO pode ser utilizado no processo de desenvolvimento de aplicações de mobilidade, pois dá suporte à criação de variados cenários, com diferentes elementos e adequação à cada especificidade encontrada. Além disso, foi notado que os resultados obtidos pelo CDPO, após processamentos realizados nos dados advindos do SUMO, tiveram boa acurácia e bom tempo de resposta, apontando que é possível realizar a integração do simulador com plataformas externas.

Palavras-chave: Cidades Inteligentes, SUMO, Mobilidade

Abstract

Smart Cities emerge as an alternative for improvement in the face of numerous problems caused by accelerated urbanization. In this way, new applications being developed to act in this context, in areas such as urban mobility, health, infrastructure, resource management, among others. Therefore, software Simulation emerges as a necessary tool for developing solutions efficient, as it allows the planning and testing of applications in a single virtualized, avoiding costs and saving testing time in real environments. For this Thus, the Urban Mobility Simulation (SUMO) is an example of a simulator that can be used, as it provides a series of elements and functions that allow modeling of realistic traffic environments. This work aimed to investigate the use of SUMO to create mobility applications in smart cities; through controlled scenarios and carrying out experiments, in addition to carrying out an analysis the integration of this simulator with the Context Data Processing Orchestrator middleware (CDPO), an infrastructure for multi-layer stream processing by smart city apps to check your performance with platforms external. The results obtained indicate that SUMO can be used in the process development of mobility applications, as it supports the creation of varied scenarios, with different elements and adaptation to each specificity encountered. Furthermore Furthermore, it was found that the results obtained by the CDPO after the processing carried out in SUMO data had high precision and good response time, pointing out It is possible to integrate the simulator with external platforms.

Keywords: Smart Cities, SUMO, Mobility

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

As cidades com grandes aglomerados e centros urbanos, como conhecemos hoje, teve uma formação marcada pelo início da revolução industrial, por volta do século XVIII e XIX. A partir daí, com mais evidência no século XX e XXI, houve a intensificação do êxodo rural, o que fez com que as cidades recebessem cada vez mais pessoas, aumentando o contingente populacional e consequentemente acentuando problemas de saúde, infraestrutura, saneamento, sustentabilidade, dentre outros (CIDADE, 2013). Nesse contexto, várias soluções são pensadas para amenizar o impacto desses problemas, uma vez que as cidades, em 2021, já abrigavam cerca de 55% da população mundial, e em 2050, esse número deverá ultrapassar os 68% (UN-Habitat, 2022).

Dessa forma, a complexidade desses centros urbanos tende a aumentar cada vez mais, exigindo novas técnicas de gerenciamento de recursos naturais, novos meios de comunicação, mais eficiência na prestação de serviços e novas formas de lidar com várias questões, como saúde, segurança e mobilidade. Dentro desse cenário, as cidades inteligentes (CI) surgem como uma alternativa para proporcionar resoluções aos problemas apresentados, uma vez que utilizam propostas inovadoras através das Tecnologias da Informação e da Comunicação (TICs) para melhorar a qualidade de vida da sua população, aprimorando o uso de seus recursos e provendo melhores serviços (PASETO; MARTINEZ; PRZEYBILOVICZ, 2020).

Não há ainda na literatura um conceito único e completo que defina as cidades inteligentes. Para Hall (HALL, 2000), Cidade Inteligente é uma cidade que monitora e integra todos os seus recursos, como trânsitos, energia, consumo de água e comunicações, a fim de otimizar o uso de sua infraestrutura, auxiliar na tomada de decisões e monitorar aspectos de segurança. Dameri (DAMERI et al., 2013) define Cidade Inteligente como uma área geográfica bem definida, que utiliza tecnologias de ponta, logística e produção de energia, além de inclusão, participação, qualidade e desenvolvimento inteligente. Já Giffinger (GIFFINGER et al., 2007) diz que as cidades inteligentes têm bom desempenho e são construídas a partir da 'combinação inteligente' de recursos e atividades de cidadãos autônomos, independentes e conscientes. De acordo com o exposto, é possível perceber que apesar de não haver apenas uma definição presente para o tema, todas elas convergem para o fato de que as cidades inteligentes possuem o objetivo final de melhorar a qualidade de vida dos habitantes, evidenciando as possibilidades de inovação e integração de tecnologias para o bem coletivo.

De acordo com (ÖBERG; GRAHAM; HENNELLY, 2017), as cidades inteligentes sugerem uma mudança na forma que as empresas atuam e interagem, pois a iniciativa privada encontra um universo potencial de oportunidades, com novos negócios e variadas

formas de monetizar ideias, através de aplicativos, programas, equipamentos e dispositivos que serão necessários para possibilitar a implementação dessa revolução social e tecnológica que elas propõem. Os Estados, por sua vez, se interessam no estudo e exploração desse tema por conta das complexidades sociais, naturais e econômicas cada vez mais crescentes que envolvem as cidades (FERREIRA, 2000), e pela urgência de resolver esses problemas. Além disso, há também uma pressão social para que isso seja feito de modo sustentável e eficiente (GENARI et al., 2018).

Uma CI deve possuir uma grande quantidade de sensores espalhados pelo espaço urbano, de modo a conseguir monitorar diversas variáveis em tempo real, como condição climática, qualidade do ar, qualidade da água, consumo de energia dos domicílios e prédios e níveis de congestionamentos, além de atuadores para realizarem intervenções necessárias, como em sinais de trânsito inteligentes. Essa disponibilidade de sensores e atuadores irão ser a base para que aplicações e serviços sejam desenvolvidos em áreas como mobilidade, ao conseguir prever e diminuir a quantidade de engarrafamentos, na saúde, monitorando dados de pacientes em risco e traçando planos para a prevenção de grupos alvo, em infraestrutura, monitorando a estrutura de prédios e edifícios antigos, e em diversas outras áreas que podem ser otimizadas nesse cenário. Além disso, é necessário que haja uma infraestrutura de conexão e comunicação, para permitir que esses sensores e outros dispositivos se conectem à rede e possam se comunicar e trocar dados entre si.

A idealização das cidades inteligentes só se tornou possível por consequência dos avanços nas diversas áreas das TICs, que permitiram o surgimento de uma ampla gama de dispositivos, como sensores e *smartphones*, e uma rede capaz de interligá-los e viabilizar a comunicação entre eles. Nesse cenário, a Internet das Coisas (Internet of Things - IoT) se apresenta como uma abordagem que possibilita um ambiente em que diferentes aparelhos troquem informações entre si, permitindo a criação de novos serviços, programas e sistemas (MEDEIROS et al., 2018).

A Internet das Coisas (IoT) apresenta uma nova percepção sobre a internet, em que não apenas computadores e *smartphones*, mas também geladeiras, sensores, micro-ondas, carros, lâmpadas e outros objetos do cotidiano das pessoas podem estar conectados à rede (FILHO, 2016). Segundo (MALDONADO et al., 2016), por fazer uma abstração virtual do mundo real, a Internet das Coisas é apropriada para fazer o gerenciamento dos dispositivos (ou nós) conectados, e conta com uma infraestrutura que abrange os hardwares utilizados, a plataforma intermediária (*middleware*) que fará a coleta e processamento dos dados, e a camada responsável pela apresentação desse processamento, acessível a gestores e desenvolvedores.

Nas cidades inteligentes, há uma grande quantidade de dispositivos produzindo e enviando dados sobre os diversos ambientes, tais como sensores de temperatura, medidores de consumo de energia, sensores de localização em carros e ônibus, dentre outras fontes.

Nesse contexto, há um contínuo fluxo de dados entre os produtores e os consumidores, tornando necessário o uso de ferramentas de *Big Data* para auxiliar na coleta e análise dessas informações (MALDONADO et al., 2016).

O grande volume e heterogeneidade de dados coletados por esses dispositivos configura uma complexidade para o processamento integrado dessas informações, dificultando uma análise unificada, o que prejudica o gerenciamento e a tomada de decisões eficientes (SILVA, 2015). Dessa forma, é necessária a utilização de mecanismos que propiciem a avaliação conjunta desses fluxos de dados, sendo possível correlacionar e integrar resultados para uma melhor tomada de decisões. Dessa forma, Para lidar com o enorme fluxo proveniente dos dispositivos inteligentes, é preciso que hajam tecnologias capazes de suportar tamanho volume, executando o processamento de maneira eficiente e rápida.

Desenvolver softwares e produtos para serem utilizados em um contexto de cidades inteligentes não é uma tarefa simples, pois há uma diversidade muito grande de aparelhos, linguagens de programação e metodologias de implementação. Assim, surge o projeto InterSCity[1], cuja finalidade é possibilitar o desenvolvimento de aplicações e serviços para cidades inteligentes. A plataforma InterSCity é um projeto composto por uma equipe multidisciplinar, cujo objetivo principal é permitir o desenvolvimento de tecnologias e métodos de código aberto reutilizáveis de forma padronizada (ESPOSTE et al., 2017).

1.1 Motivação

As cidades inteligentes possuem a finalidade de proporcionar aos cidadãos uma melhor qualidade de vida, propondo soluções eficientes para os diversos problemas enfrentados na vida cotidiana. Nesse contexto, a mobilidade urbana é um dos temas de maior interesse nessa área, visto que o crescimento do fluxo de automóveis nas grandes cidades não foi acompanhado por uma estratégia organizada dos gestores, impactando, assim, de forma negativa o planejamento urbano (ALVAREZ, 2017).

O crescimento acelerado da frota de veículo nos centros urbanos desencadeia uma série de complicações que afetam diretamente o dia a dia das pessoas, como congestionamentos, atrasos, acidentes, dentre outros inconvenientes. Para (RESENDE; SOUSA, 2009) essas questões são críticas, e devem ser prioridade das administrações públicas, pois além de afetar negativamente o bem-estar dos cidadãos, causam enormes prejuízos econômicos e sociais.

Para ratificar a importância de encontrar alternativas eficazes para essa problemática, (GIFFINGER et al., 2007) classifica a mobilidade urbana como uma das 6 dimensões principais em uma Cidade Inteligente, como pode ser observado na Figura 1.

[1] https://interscity.org/

Segundo o autor, o parâmetro *Smart Mobility* mede o quanto uma cidade provê facilidades para que seus habitantes utilizem os diferentes tipos de meio de transporte, como ônibus públicos, metrôs, bicicletas e carros, ou seja, quão eficiente é a mobilidade urbana naquela área. Nesse sentido, fatores como tamanho de congestionamentos e porcentagem dos habitantes que utilizam transporte público podem ser utilizados para mensurar o grau de inteligência da cidade nesse quesito.

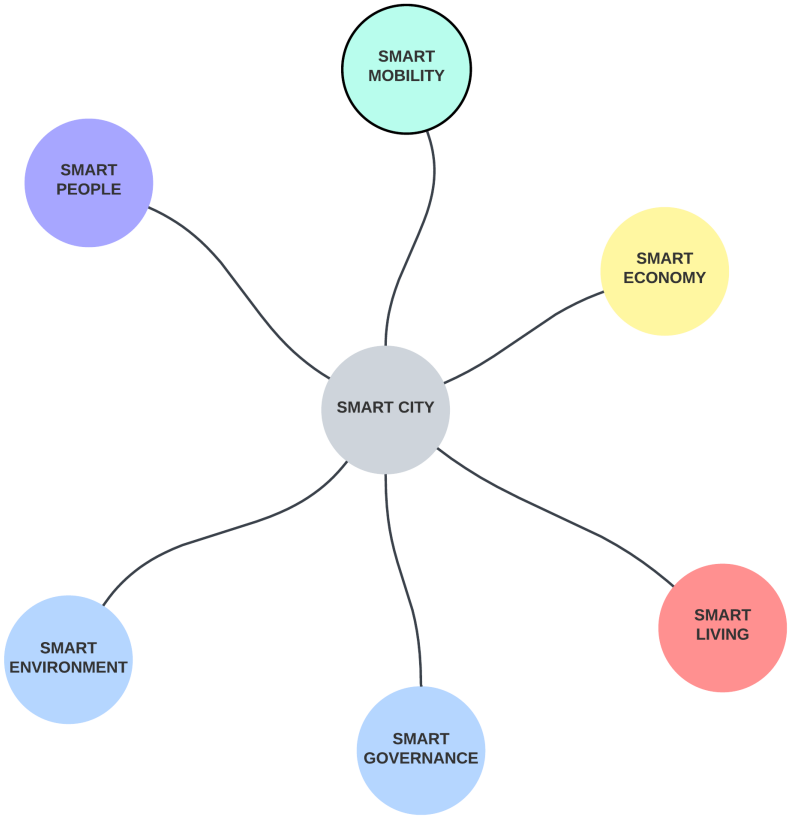

Figura 1 – Dimensões de uma Cidade Inteligente

Sabendo disso, diversas propostas podem ser pensadas para propiciar melhorias no âmbito dessa dimensão, como o acompanhamento de transportes públicos para detectar atrasos, observação em tempo real do fluxo de trânsito para monitorar congestionamentos, fiscalização da velocidade média dos veículos em vias mais críticas, aplicações para disponibilizar vagas de estacionamentos livres, implementação de sinais de trânsito inteligentes, dentre outros serviços que ajudam a gerenciar o fluxo de trânsito.

Em um contexto de cidades inteligentes, o desenvolvimento de novas soluções encontra empecilhos relacionados ao tempo e viabilidade de implementação, uma vez que realizar testes e validações em ambientes reais demandam custos e esforços que fogem do escopo do desenvolvedor. Nesse sentido, muitos softwares dão suporte ao desenvolvimento e testes de aplicações, o que possibilita aos desenvolvedores a elaboração de estratégias

mais eficientes para a execução de projetos, uma vez que podem realizar simulações antes de irem para o ambiente real.

Nesse cenário, o SUMO [2] (*Simulation of Urban MObility*) é um exemplo de software que tem como objetivo permitir a criação de cenários de simulação para aplicações de mobilidade, pois disponibiliza um ambiente com várias ferramentas e possibilidades de modelagem de fluxos de trânsitos, veículos, rotas e demais elementos, auxiliando no planejamento de aplicações específicas para cada contexto encontrado.

Em vista disso, o SUMO se apresenta como uma alternativa para avaliar as soluções de mobilidade em cidades inteligentes, pois fornece aos desenvolvedores e aos gestores da cidade uma visão realista do estado atual do trânsito, além de demonstrar como a solução sugerida iria impactar nesse contexto, possibilitando uma avaliação mais eficaz e satisfatória antes da implementação real, economizando, assim, tempo e recursos financeiros.

1.2 Objetivos

O objetivo geral desse trabalho é investigar o simulador SUMO como ferramenta para criação de aplicações de mobilidade no contexto de Cidades Inteligentes, através de cenários controlados e da condução de experimentos para verificar sua usabilidade.

1.2.1 Objetivos Específicos

- Estudar as principais características do SUMO e entender o passo a passo para a criação de simulações de mobilidade urbana.
- Implementar cenários controlados através do SUMO para avaliar aplicações de mobilidade em cidades inteligentes.
- Implementar a integração com o *middleware Context Data Process Orchestrator* (CDPO), uma extensão da plataforma InterSCity usado em aplicações para cidades inteligentes, para processar dados advindos do SUMO, e assim, analisar a integração do simulador com plataformas externas.

1.3 Organização do Trabalho

Este trabalho está organizado em cinco capítulos, de forma a apresentar o conteúdo mais claramente, conforme os parágrafos a seguir.

O capítulo 1 apresenta uma introdução sobre cidades inteligentes e internet das coisas, evidenciando as características e desafios inerentes a esses conceitos. Além disso,

[2] https://eclipse.dev/sumo/

esse capítulo apresenta a motivação e os objetivos desse trabalho. O capítulo 2 aborda uma revisão bibliográfica dos principais, tendo como temas principais o Processamento de Eventos Complexos (CEP) e o *Context Data Process Orchestrator* (CDPO). O capítulo 3 discorre sobre o simulador SUMO, tema principal desse trabalho. Nesse capítulo, são apresentados conceitos, características, funções e ferramentas disponibilizadas por esse *software*. No capítulo 4, encontram-se os resultados e discussões dos experimentos realizados. O capítulo 5 apresenta as conclusões desse trabalho.

2 Fundamentação Teórica

Esse capítulo tem a finalidade de apresentar os principais fundamentos relacionados com cidades inteligentes e mobilidade urbana nesses contextos. Sendo assim, serão abordados os principais conceitos de Processamento de Eventos Complexos (do inglês, *Complex Event Processing* - CEP), um paradigma de programação utilizado para processar dados próximo ao tempo real. Será mostrado como essa abordagem pode ser utilizados para tratar dados advindos de veículos equipados com dispositivos de IoT. Nesse trabalho, o CEP foi utilizado para receber fluxo de dados advindos do simulador e identificar padrões de eventos esperados, como atrasos e chegadas de ônibus e detecção de velocidades médias. Além disso, será apresentada a abordagem de processamento de dados em múltiplas camadas e a utilização do *middleware* Context Data Process Orchestrator (CDPO) para possibilitar isso.

2.1 Processamento de Eventos Complexos

A internet das coisas(do inglês, Internet of Things - IoT), é uma abordagem de comunicação que faz referência à interconexão de diversos objetos do cotidiano à internet, tornando a rede mais imersiva e onipresente (ZANELLA et al., 2014). Os avanços na área de IoT fazem surgir uma infinidade de novas possibilidades de aplicações e serviços, uma vez que há um crescente avanço na tecnologia de dispositivos como sensores, aparelhos eletrônicos e veículos, e dessa forma, uma vez conectados à rede, eles podem transmitir e receber dados, podendo adquirir comportamentos dinâmicos e interativos.

Nessa perspectiva, uma cidade inteligente se utiliza dessa gama de dispositivos para criar novos serviços e prover mais qualidade de vida para os habitantes. Uma das aplicações possíveis que a IoT possibilita em um ambiente urbano é o monitoramento de veículos e fluxos de trânsitos através de sensores de localização e velocidade instalados nos veículos, além do uso de câmeras inteligentes para detectarem situações de interesse. Assim, é possível perceber uma aplicabilidade dessa tecnologia na área de mobilidade, pois várias aplicações podem ser pensadas para auxiliar no gerenciamento inteligente do trânsito urbano, ao realizar o processamento dos dados advindos dos sensores de veículos e demais fontes de informações.

No entanto, essa nova realidade em que milhares de dispositivos se conectam e enviam dados para processamento, que muitas vezes devem ser realizados em um tempo próximo ao real, trouxe uma série de desafios aos modelos tradicionais de tratamento de informações. Tal fato acontece pois os bancos de dados são ideais para a persistência e posterior consulta de dados, não sendo ideais para aplicação de consultas em tempo real,

configurando uma certa latência às aplicações. Desse modo, uma boa opção é utilizar CEP para reagir ao fluxo de informações advindas dos veículos e demais elementos de uma rede de tráfego urbano.

O Processamento de Eventos Complexos se refere à capacidade de processar altos fluxos de dados com maior eficiência, de modo a evitar uma grande latência, uma vez que diversas aplicações em cidades inteligentes requerem rapidez na tomada de decisões. Para possibilitar essa rapidez, o CEP aplica consultas contínuas a um determinado fluxo de eventos que chega, ou seja, esse fluxo é a principal fonte de entrada desse paradigma (JUNIOR et al., 2019).

Vários elementos podem ser a fonte de eventos que serão processados, como sensores, smartphones, veículos, dentre outros (NETO; FONSECA, 2020). O componente de software responsável por enviar os eventos gerados pela fonte para o sistema de processamento são os produtores. Os produtores enviam eventos simples, como localização, velocidade, temperatura, dentre outros. Nesse contexto, o CEP disponibiliza primitivas que podem ser aplicadas a esses eventos simples, como filtros, agregações, junções, etc. Essas primitivas de processamento fazem com que sejam geradas derivações, sendo conhecidos como eventos complexos, ou seja, eventos compostos que são formados pelo processamento de eventos mais simples.

Para entender melhor as primitivas de processamento disponibilizadas pelo CEP, é preciso, antes, compreender como os eventos são percebidos nesse paradigma. Um evento pode ser qualquer ação que acontece em determinado domínio, como uma mudança de temperatura, alteração de velocidade, horário de chegada, dentre outros. Nesse contexto, um evento terá a seguinte estrutura: Tipo, que definirá sobre o que se trata o evento, a hora em que esse evento foi gerado - *timestamp* - e o *payload*, que é o conteúdo específico do evento (LUCKHAM, 2001).

O CEP disponibiliza alguns operadores que aplicados a um fluxo de eventos, são capazes de gerarem outros eventos complexos. Uma dessas primitivas de processamento é o filtro, cuja função é separar ou reter os eventos que chegam de acordo com algum critério (JUNIOR et al., 2019). Por exemplo, utilizando um contexto de mobilidade urbana, em que há um fluxo eventos cujo *payload* é a velocidade de veículos, a primitiva de filtro poderia ser utilizada para capturar apenas as velocidades que excedem um determinado limiar. Supondo que esse limiar seja 80 km/h, a consulta CEP que aplicaria esse filtro seria a seguinte:

```
1    select *  from VelocidadeVehicle(velocidade > 80)
```

Dessa forma, mesmo que chegassem velocidades abaixo de 80km/h, apenas as medições que ultrapassem esse valor seriam capturadas pelas consultas. É importante observar no exemplo acima a sintaxe utilizada pelo CEP. Para realizar as consultas, é

utilizada uma linguagem declarativa que se assemelha à sintaxe SQL, reaproveitando muitos termos e símbolos dessa linguagem, como as cláusulas `FROM, WHERE, ORDER BY, SELECT`, dentre outras. Essas consultas que são aplicadas a fluxo de eventos pertencem à linguagem EPL (Event processing Languages), uma vez que mesmo contendo similaridades com a linguagem SQL, possuem particularidades específicas para tratar fluxo de dados (GOMES,).

É possível, também, usar banco de dados de fontes externas para atribuir novas informações ou modificar os eventos que chegam. A primitiva que possibilita tal ato é a de enriquecimento (LUCKHAM, 2001). Outra função de processamento muito utilizada em consultas EPLs é a agregação. Ela reúne os eventos de entrada em um único evento de saída, e é útil quando apenas eventos isolados não suficientes para se chegar a uma conclusão. A partir disso, é possível utilizar algumas funções conhecidas da linguagem SQL, como `avg(), max() e min()` (JUNIOR et al., 2019). A EPL abaixo apresenta um exemplo onde se quer calcular a média de velocidade dos veículos cuja velocidade é maior que 30km/h.

```
1    select avg(velocidade) as media
2    from VelocidadeVehicle(velocidade > 30)
```

Nesse exemplo, é interessante perceber que há o uso de duas primitivas: a primitiva de agregação `avg(velocidade)` e a primitiva de filtro `(velocidade > 30)`. O resultado gerará um atributo derivado denominado `media`, cujo conteúdo é a média das velocidades que são maiores que 30km/h. Nesse contexto, é possível perceber que a utilização de mais de uma primitiva possibilita que consultas mais elaboradas sejam feitas, a fim de se conseguir extrair a maior quantidade de informações possíveis presentes em um fluxo de eventos, ou seja, é possível combinar várias funções de processamento para que se tenha o resultado esperado.

Em processamento de eventos complexos, há ainda as primitivas de detecções de padrões, cujo objetivo principal é analisar uma série de eventos para verificar se eles atendem a determinada regra pré-estabelecida (RODRIGUES et al., 2020). Para realizar essa análise, essas primitivas utilizam operadores lógicos, como o `and, or` e `not` para expressarem as regras desejadas. Além dos operadores lógicos, outros termos podem ser utilizados na criação de regras que irão compor as EPLs, como o `every`, o operador '`-->`', utilizado quando se quer ter uma ordem temporal dos eventos e expressões como `timer:within()`, quando se deseja detectar padrões que aconteceram dentro de um tempo específico (EsperTech, 2023) [1].

As primitivas de padrão possuem uma nomenclatura ligeiramente diferente, pois exigem a expressão `from pattern` para que as consultas sejam aplicadas corretamente.

[1] http://esper.espertech.com/release-8.8.0/reference-esper/html/processingmodel.html

Além disso, em muitas consultas, há a presença da palavra `where`. Esse termo, assim como a primitiva de filtro, é usado para restringir o fluxo de eventos de acordo com alguma condição. Há, contudo, uma sutil diferença entre a primitiva de filtro e a cláusula `where`, que diz respeito a como os eventos são separados: o filtro restringe os eventos que entram na janela de processamento, já a cláusula `where` separa, dentre os eventos que chegam na janela, aqueles que atendem aos critérios (JUNIOR et al., 2019).

Outra primitiva de padrão muito utilizada é a primitiva de sequência, que busca relacionar eventos pela ordem de chegada (*timestamp*) e por seus tipos (JUNIOR et al., 2019). A sintaxe utilizada nessa primitiva é do tipo `A ->B ->C`, que pode ser interpretada como a chegada do evento A, seguida da chegada do evento B, seguida da chegada do evento C, ou seja, a consulta só gerará resultados se essa sequência for seguida. Além dos exemplos apresentados, outros operadores são suportados pela primitiva de detecções de padrões, como conjunções, disjunções e negações, dentre outras (JUNIOR et al., 2019).

Conhecidas as primitivas de processamento e como elas podem ser utilizadas para produzirem consultas EPL, é essencial conhecer o conceito de janelas de tempo ou de tamanho. As janelas, também chamadas de contexto, são utilizadas para subdividirem o fluxo de eventos em uma ou mais partições (LUCKHAM, 2001), isto é, o fluxo de eventos não será processado de uma forma uniforme, mas a consulta será aplicada a cada partição, separadamente. Em CEP, há duas formas de janela que podem ser utilizadas: as janelas deslizantes *sliding windows* e as janelas em lote *(*batch windows) (EsperTech, 2023). A Figura 2 (JUNIOR et al., 2019) retrata o modo de funcionamento dessas duas janelas. A principal diferença entre essas duas abordagens é em como os eventos ficam retidos para serem processados em cada uma. O tipo de janela deslizante move uma janela sobre os eventos que chegam durante o tempo t, isto é, o processamento é realizado nos eventos que fazem parte da janela naquele determinado momento, e a cada unidade de tempo que passa, a janela vai se movendo de modo a retirar os eventos mais antigos e adicionar novos. Já a janela em lote utiliza uma estratégia de reter todos os eventos que chegam em um determinado tempo t e só realiza o processamento quando a janela está preenchida, ou seja, o processamento é realizado sobre um grupo de eventos e depois esses são descartados, a fim de que novos eventos preencham a janela e sejam processados posteriormente (RODRIGUES et al., 2020).

Para realizar o processamento de eventos, o CEP utiliza entidades denominadas de *Event Processing Agent* (EPAs) - Agente de Processamento de Eventos - cuja responsabilidade é aplicar aos eventos as funções de processamento, como filtragem, agregações, padrões, dentre outras, e gerar resultados para o usuário final ou gerar um evento derivativo para ser consumido por outros EPAs (JUNIOR et al., 2019). Quando há a interconexão de vários EPAs, tem-se uma abstração chamada de Event Processing Network

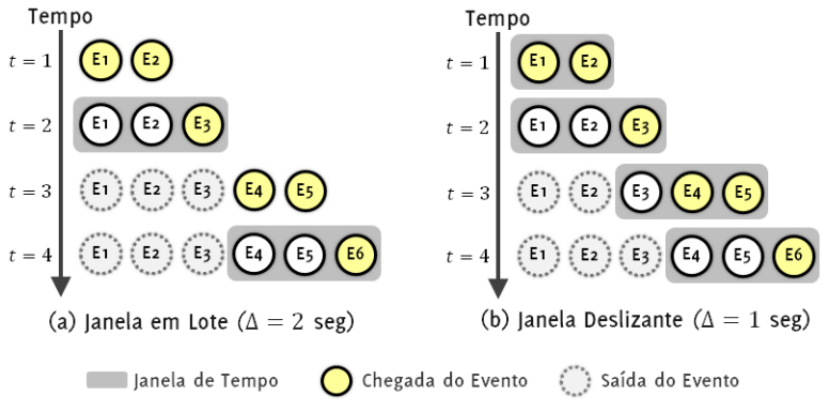

Figura 2 – Fluxo de Eventos

- Rede de Processamento de Eventos -, cuja finalidade é realizar um processamento mais detalhado de um fluxo de eventos.

As EPNs representam a arquitetura central do CEP, visto que normalmente, um único EPA não é suficiente para gerar um evento complexo, e dessa forma, é preciso que vários agentes sejam combinados para tratar o fluxo (RODRIGUES et al., 2020). A figura 3 (JUNIOR et al., 2019) demonstra o funcionamento dos EPAs, que recebem os eventos dos elementos produtores, se interconectam com outros agentes, formando uma EPN, e entregam os resultados para os consumidores finais.

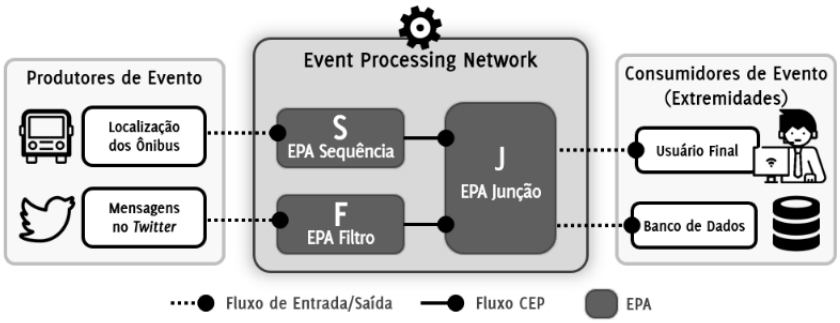

Figura 2.2: Exemplo de EPN para detecção de anomalias no trânsito.

Figura 3 – EPA e EPN

Por fim, um motor de inferência CEP é necessário para realizar a implementação dos conceitos vistos até aqui, como as primitivas de processamento, consultas EPL, eventos, EPAs e EPNs (JUNIOR et al., 2019). Um dos motores CEP, também denominados de engines CEP, mais utilizados é o motor de inferência Esper (EsperTech, 2023), uma vez que é implementado na linguagem JAVA, sendo, assim, acessível e de fácil manipulação, podendo ser utilizado como extensão ou até mesmo biblioteca da linguagem (RODRIGUES et al., 2020). Dessa forma, com um motor Esper instanciado, é possível utilizar o CEP para resolver diversos problemas que necessitem de um alto processamento de fluxo de

dados. Há outros motores de inferência CEP disponíveis, e a principal diferença entre cada um serão na estrutura e nas sintaxes utilizadas para realizarem as consultas.

Portanto, é possível notar a variedade de aplicações que podem empregar o uso de processamento de eventos complexos para tratarem dados de forma eficaz e com baixa latência. No presente trabalho, essa abordagem foi utilizada, através do *middleware* CDPO, para realizar o processamento de dados advindos de veículos presentes no simulador SUMO, a fim de detectar situações de interesse como atraso e chegada de ônibus em paradas e velocidade média de veículos que transitam por determinada via.

2.2 Context Data Process Orchestrator (CDPO)

No ambiente de cidades inteligentes, é necessário que o armazenamento e processamento de informações seja realizado em sistemas computacionais distribuídos e escaláveis, devido ao grande número de dispositivos IoT conectados e à dispersão física desses. Dessa forma, a *Cloud Computing* se apresenta como uma alternativa possível, pois ela permite que os dados sejam armazenados e processados em computadores e servidores compartilhados através de uma rede Internet (SCHENFELD, 2017).

Entretanto, apesar de fornecer um ambiente escalável e distribuído, facilitando a integração dos diversos dispositivos IoT com a nuvem, a *Cloud Computing* pode apresentar algumas limitações quando se deseja processar informações com baixa latência. Isso pode ser observado em aplicações de segurança, sistemas médicos, detecção de anomalias, dentre outros. Essas aplicações, também conhecidas como aplicações sensíveis à latência de comunicação, precisam ter uma resposta próxima ao tempo real, e a *Cloud Computing*, por receber muitas requisições e geralmente estar fisicamente longe dos dispositivos, pode gerar um atraso na entrega de respostas.

Assim, a abordagem de processamento em múltiplas camadas surge para auxiliar na resolução desse problema, pois dá suporte ao recebimento de um grande fluxo de eventos, processando-o em diferentes níveis de complexidade. A ideia dessa abordagem é fazer com que a *Cloud Computing* receba um menor volume de dados e processamento, utilizando, para isso, camadas intermediárias, como a *Fog Computing* e a *Edge Computing*.

A *Fog Computing* pode ser definida como uma camada que fornece serviços de computação, comunicação e armazenamento entre os dispositivos finais e os centros tradicionais de computação em nuvem (COUTINHO; CARNEIRO; GREVE, 2016). Essa camada recebe os dados dos dispositivos e realiza um processamento com menor latência, por estar localizado mais próximo do dispositivo e receber um número menor de informações. A *Edge Computing*, por sua vez, surgiu a partir da evolução dos dispositivos móveis, como *Smartphones*, *Smartwatches* e sensores, pois suas capacidades de processamento aumentaram, possibilitando que haja um tratamento prévio de informações no próprio

dispositivo, com um tempo de resposta ainda menor que a *Fog*. A Figura 4 (YOU, 2023) demonstra a distribuição de dados em uma arquitetura de múltiplas camadas.

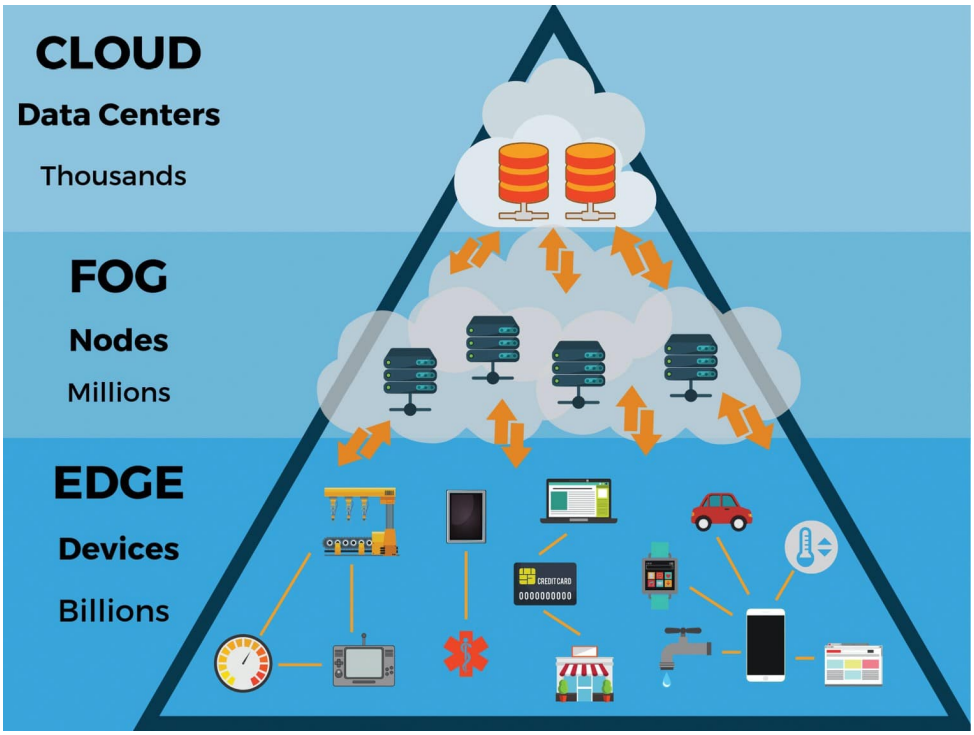

Figura 4 – Edge, Fog e Cloud

É possível notar, através da Figura 4, que essa abordagem distribuída de dados proporcionou uma flexibilidade maior para as aplicações IoT, possibilitando o desenvolvimento de serviços que interagem mais rapidamente com o usuário, uma vez que concentra em dispositivos do cotidiano o processamento de informações menos complexas, além de se ter uma camada intermediária que diminui ainda mais o tráfego de dados para a nuvem.

Nesse contexto, foi desenvolvido pelo Laboratório de Sistemas Distribuídos Inteligentes (LSDi-UFMA) o *Context Data process Orchestrator* - CDPO - que é um *middleware* cujo objetivo principal é dar suporte ao processamento de fluxo de dados nas camadas *Cloud*, *Fog* e *Edge*. Esse *middleware* é uma extensão da plataforma *InterSCity*, pois provê funcionalidades que auxiliam no desenvolvimento de aplicações para cidades inteligentes. Para definir a arquitetura do CDPO, foram especificados alguns requisitos que são necessários para que o *middleware* consiga atuar com eficiência nesses cenários.

A escalabilidade é um requisito essencial para suportar o grande número de dispositivos e componentes presentes em uma Cidade Inteligente, bem como o alto fluxo de dados proveniente desse contexto. Além disso, o CDPO deve ser capaz de realizar o *deployment* de EPNs nas camadas solicitadas pela especificação da regra CEP, e após o fim do processamento, deve ser capaz de realizar seu o *undeployment*. Ainda como requisito

essencial, o *middleware* deve permitir a gestão dinâmica de regras baseadas em localização geográfica, tempo, ou outras informações contextuais.

Após levantar os requisitos principais, foi possível estabelecer uma arquitetura com os microsserviços necessários para realizar essas tarefas. A Figura 5 apresenta os microsserviços que atuam juntos para orquestrar a implantação e o processamento das EPNs e múltiplas camadas.

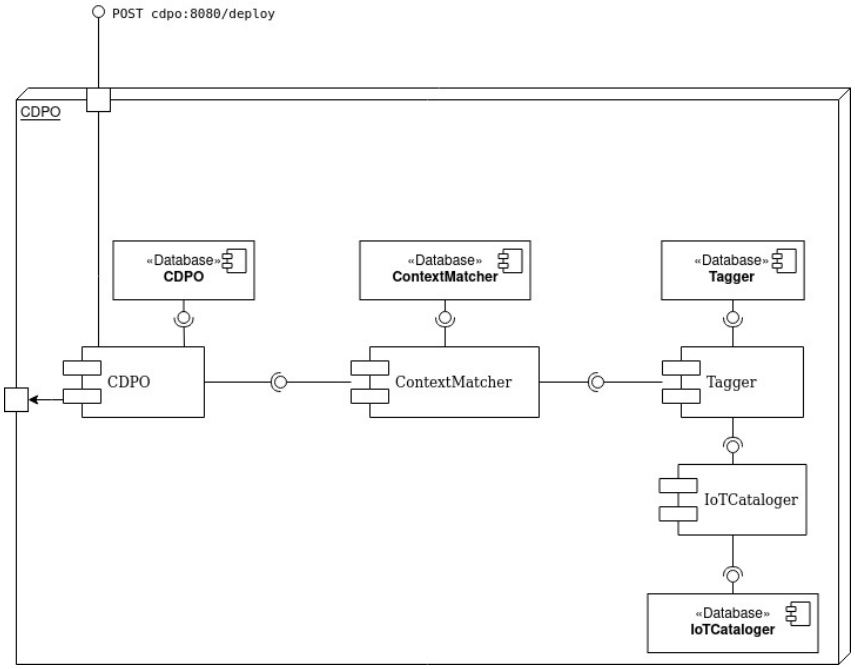

Figura 5 – Componentes do CDPO

Observando a Figura 5, é possível visualizar os microsserviços do CDPO. O `IoTCataloguer` é responsável por catalogar os dispositivos e recursos presentes no contexto de uma aplicação, armazenando-os em uma base de dados, além de disponibilizar métodos para recuperar, cadastrar, atualizar ou remover esses elementos. O `Tagger` é capaz de fazer uma espécie de categorização dos elementos, atribuindo uma marcação do tipo *chave-valor*. O `ContextMatcher`, por sua vez, tem a função de buscar os nós em que o processamento deverá ser realizado, e os guardar em uma base da dados. Pra finalizar, o `CDPO` é o principal microsserviço dessa arquitetura, uma vez que ele é o componente responsável por orquestrar o *deployment* das regras nos diferentes níveis. A Figura 6 apresenta as camadas do *middleware* CDPO.

A partir da Figura 6, é possível perceber que a arquitetura de camadas do *Context Data Process Orchestrator* é composta por quatro microsserviços em cada camada. Primeiramente, tem-se o `Collector`, responsável por atuar na coleta dos dados provenientes de sensores e de outras consultas, além de distribuí-los para o `worker` ou para as outras camadas. O componente `Worker` tem a função de realizar o processamento CEP, instanciando o motor *esper* para receber o *deployment* da EPN e gerar o resultado

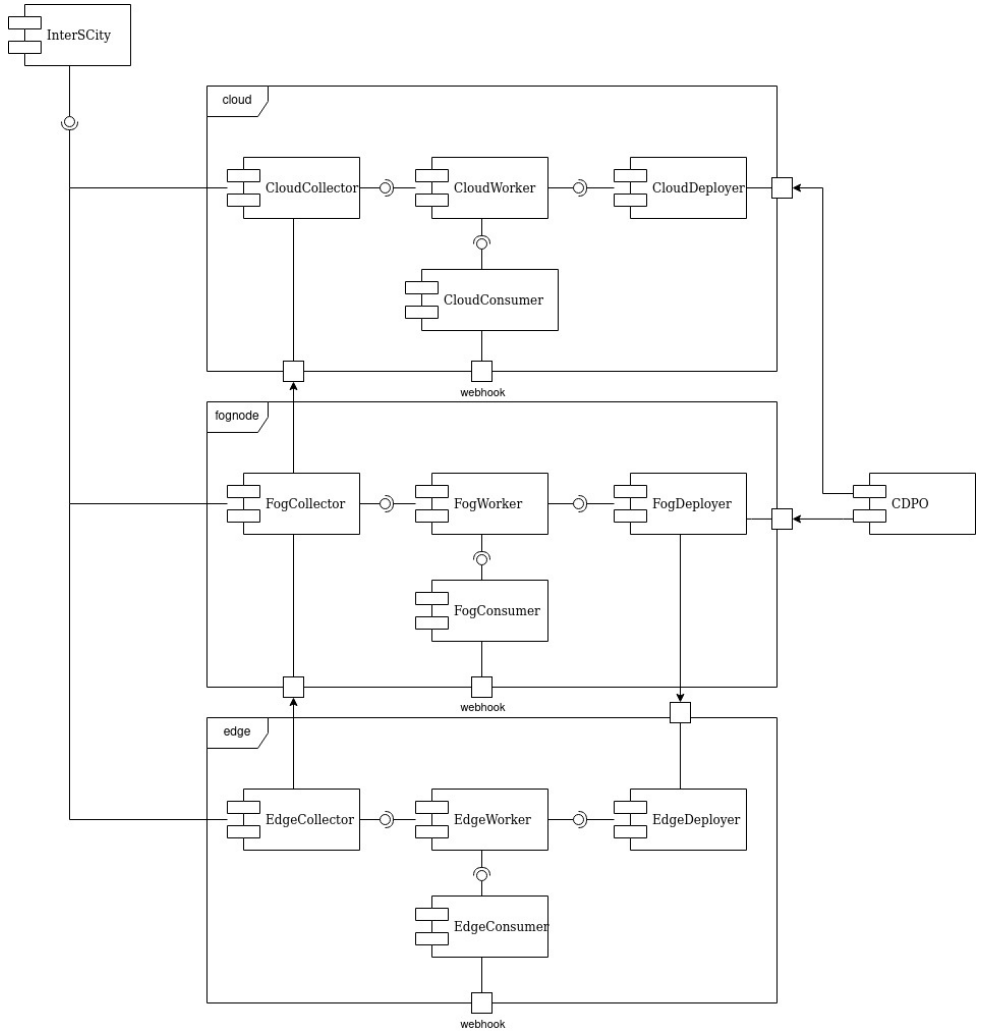

Figura 6 – Camadas do CDPO

do processamento. Já o microsserviço `Deployer` tem a finalidade de receber a requisição e realizar o *deployment* da regra CEP no `worker`. E por fim, o `Consumer` é responsável por direcionar o resultado de processamento para o *webhook* da aplicação cliente, caso esse seja o destino dos dados.

É observável, portanto, que o CDPO dá suporte ao processamento distribuído de dados, possibilitando a instanciação de regras CEP em diferentes camadas. Dessa forma, é possível utilizar esse *middleware* em aplicações de mobilidade, visto que há uma grande quantidade de veículos enviando dados de velocidade e localização, e pode-se realizar um processamento dessas informações nas camadas *edge* e *fog*, reduzindo o tráfego de informações para a nuvem e consequentemente a latência de resposta.

No presente trabalho, o CDPO foi utilizado para aplicar consultas CEP aos dados advindos do simulador SUMO, a fim de identificar algumas situações de interesse, como eventos de chegada e atraso dos ônibus, e monitoramento da velocidade média dos veículos, em uma abordagem de múltiplas camadas.

3 Simulation of Urban Mobility - SUMO

O SUMO [1] - *Simulation of Urban MObility* - é um software de código aberto, desenvolvido principalmente por funcionários do Instituto de Sistemas de Transporte do Centro Aeroespacial Alemão (LOPEZ et al., 2018). Esse software possui a finalidade de possibilitar a criação de simulações de tráfego de trânsito, que podem possuir carros, motos, caminhões, trens, pedestres e diversos outros elementos.

O SUMO engloba, em seu domínio, diversos elementos e ferramentas necessárias para possibilitar a criação de cenários e ambientes desejados para realizar simulações, desde simples até grandes redes complexas. Utilizando as funcionalidades disponibilizadas, o usuário pode criar cenários personalizados para analisar diversas situações, como engarrafamentos, viagens de ônibus, transportes públicos, acidentes, dentre outros. A flexibilidade desse software permite que sejam desenvolvidos cenários controlados das mais variadas formas, seja pra analisar o desempenho de um veículo em determinadas condições de trânsito, ou pra analisar a mobilidade urbana nos horários de pico, e até mesmo analisar o impacto de emissão de carbono dos veículos. A Figura 7 apresenta a interface do SUMO. Percebe-se que são disponibilizadas diversas opções para modelagem de fluxos de trânsito, rotas e veículos.

Figura 7 – Interface do SUMO

Para ratificar a flexibilidade e aplicabilidade do SUMO para analisar cenários de mobilidade urbana, são encontradas na literatura vários exemplos de trabalhos que usam esse software como uma etapa de validação da eficiência da solução proposta. (BORN et al., 2015) utiliza o SUMO para avaliar abordagens mais otimizadas de gerenciamento do tempo dos estados de semáforos, a fim de reduzir a emissão de poluentes na atmosfera. No contexto desse problema, o simulador oferece recursos para a modelagem de sinais de trânsito e também permite que se defina quais tipos de poluentes os veículos presentes na simulação vão emitir. Do mesmo modo, (RONDON; ROCHA-FILHO; VILLAS, 2021)

[1] https://sumo.dlr.de/docs/Downloads.php

utiliza-o para gerar fluxo de tráfego, a fim de verificar a eficiência de um novo protocolo a ser utilizado em redes veículo a veículo (v2v). Já (MANIKA et al., 2019) propõe um esquema de automatização da criação de vias e fluxos de trânsito para o SUMO, de modo a diminuir o trabalho manual na criação de simulações. E por fim, (MOURA, 2022) utiliza o *software* para propor um modelo de otimização do transporte público da cidade de Porto Alegre.

Dessa forma, é possível perceber que o SUMO pode ser usado de diversas maneiras para auxiliar o desenvolvimento de aplicações e projetos em mobilidade urbana. Por ser um software livre e de código aberto, seu uso é facilitado e difundido por desenvolvedores e demais pessoas interessadas nessa área de atuação. No entanto, é importante citar que existem outros softwares com o mesmo propósito do SUMO, ou seja, que também possibilitam a criação de cenários e ambientes de trânsito urbano. Esses softwares podem ser divididos, basicamente, em duas classificações gerais: em relação ao seu método de funcionamento e em relação à sua licença. Quanto ao método de funcionamento, os simuladores podem ser microscópicos, quando focam na mobilidade individual de cada veículo ou elemento presente na simulação, ou macroscópicos, quando se concentram em uma visão mais geral do trânsito, com controles mais amplos do sistema. Na classificação quanto à licença, há basicamente dois tipos de softwares possíveis: os de código aberto, que podem ser adquiridos e alterados por qualquer pessoa interessada, e os comerciais, cuja utilização está condicionada ao pagamento de alguma taxa. A Figura 8, de (MOURA, 2022), mostra um comparativo entre os simuladores de código aberto. De acordo com o autor, os softwares comerciais, apesar de oferecem um bom suporte para a execução de aplicações, impõem dificuldades para usos acadêmicos e de pesquisa por conta dos valores da licença.

Software	Abordagem Geral	Características Gerais
MATSim	Macroscópica	Poderosa plataforma de simulação de tráfego em cenários de grande escala, no caso de grandes cidades ou mesmo países. Essa abordagem global não é adaptada ao uso em cenários microscópicos (HULTÉN, 2019).
MAINSIM	Microscópica	Esse simulador de tráfego gera gráficos de simulação a partir de material cartográfico de forma totalmente automática e é capaz de simular cidades inteiras com veículos, bicicletas e pedestres. A exclusão dos meios de transporte público e a pouca disponibilidade de documentação on-line limitam sua utilização (DALLMEYER; TIMM, 2012).
SUMO	Microscópica	O SUMO permite a simulação de um determinado tráfego sob demanda, que consiste em um conjunto de veículos individuais que se movimentam através de uma determinada rede rodoviária. A simulação permite abordar um grande conjunto de tópicos de gerenciamento de tráfego e é puramente microscópica: cada veículo é modelado explicitamente, tem uma rota própria e se move individualmente através da rede (PANOVSKI, 2020).

Figura 8 – Softwares de simulação de código aberto

3.1 Geração de rede de tráfego

Para gerar uma simulação, é preciso seguir uma sequência de passos que vão auxiliar na construção do cenário desejado. Em um primeiro momento, é necessário pensar no

ambiente de simulação, isto é, quais características vão fazer parte daquele ambiente, como localidade, estrutura das vias, quantidade de faixas e demais elementos que irão compor o espaço físico. Após isso, é preciso definir quais tipos de veículos estarão presentes, bem como a rota que irão traçar e a quantidade desses. Em SUMO, as simulações são microscópicas, isto é, cada elemento presente será modelado de forma individual, contendo sua própria definição e rota, sendo independente no contexto da aplicação.

Vários tipos de elementos podem ser utilizados na construção de um cenário de simulação, como vias, sinais de trânsito, paradas de ônibus, veículos dos mais variados tipos e outros elementos adicionais. O SUMO oferece diversas maneiras para se construir uma rede de tráfego, seja através de um programa adicional chamado *netedit*, onde se é possível construir vias e fluxo de veículos manualmente, ou através da importação de mapas de sites especializados, como o *OpenStreet* [2], na qual se usa um mapa já existente e é realizada a conversão desse para um formato que o SUMO entenda, utilizando um programa chamado *netconvert*, ou mesmo utilizando a biblioteca python *TraCI* para gerar vias e fluxos de modo aleatório. Nota-se que há uma liberdade muito grande para essa etapa, e a escolha do método de construção irá depender do contexto da aplicação a ser desenvolvida, isto é, se o desenvolvedor deseja criar um ambiente novo, para verificar a adição de vias e elementos, ou se deseja realizar testes em alguma localidade já existente.

Além disso, é possível também combinar mais de um método para gerar um ambiente controlado. Neste trabalho, foram utilizadas as combinações de duas etapas para gerar o cenário de simulação: a importação do mapa da UFMA através do *OpenStreet* e a adição de elementos adicionais a partir do *netedit*. A Figura 9 apresenta alguns dos elementos disponibilizados pelo *netedit*.

Como observado na Figura 9, o *netedit* oferece algumas opções de edição, como adicionar ou deletar vias, inserção de elementos adicionais como sinais ou paradas de ônibus, conexão de duas vias, dentre outras. Essa interface gráfica permite que vários elementos de tráfegos sejam incorporados à simulação de forma intuitiva e simples. Permite, ainda, a seleção de cada elemento individualmente, além de uma série de atributos que irão definir suas características dentro do ambiente de simulação. Por exemplo, ao escolher o objeto *Edge*, é possível criar uma estrada entre dois pontos do espaço, e a partir disso, definir quais serão os atributos dessa via, como número de faixas, velocidade máxima permitida, largura, dentre outros. A Tabela 1 apresenta os atributos possíveis para as *Edges*.

Do mesmo modo, outros objetos da simulação irão conter atributos que irão descrevê-los. Na Figura 9, ainda, é possível observar elementos como *Traffic Light*, ou sinais de trânsito, e *Additional*, que representam as paradas de ônibus. Nos sinais de trânsito, é possível definir atributos como as fases, que são a sequência de estados de cores que

[2] https://www.openstreetmap.org/

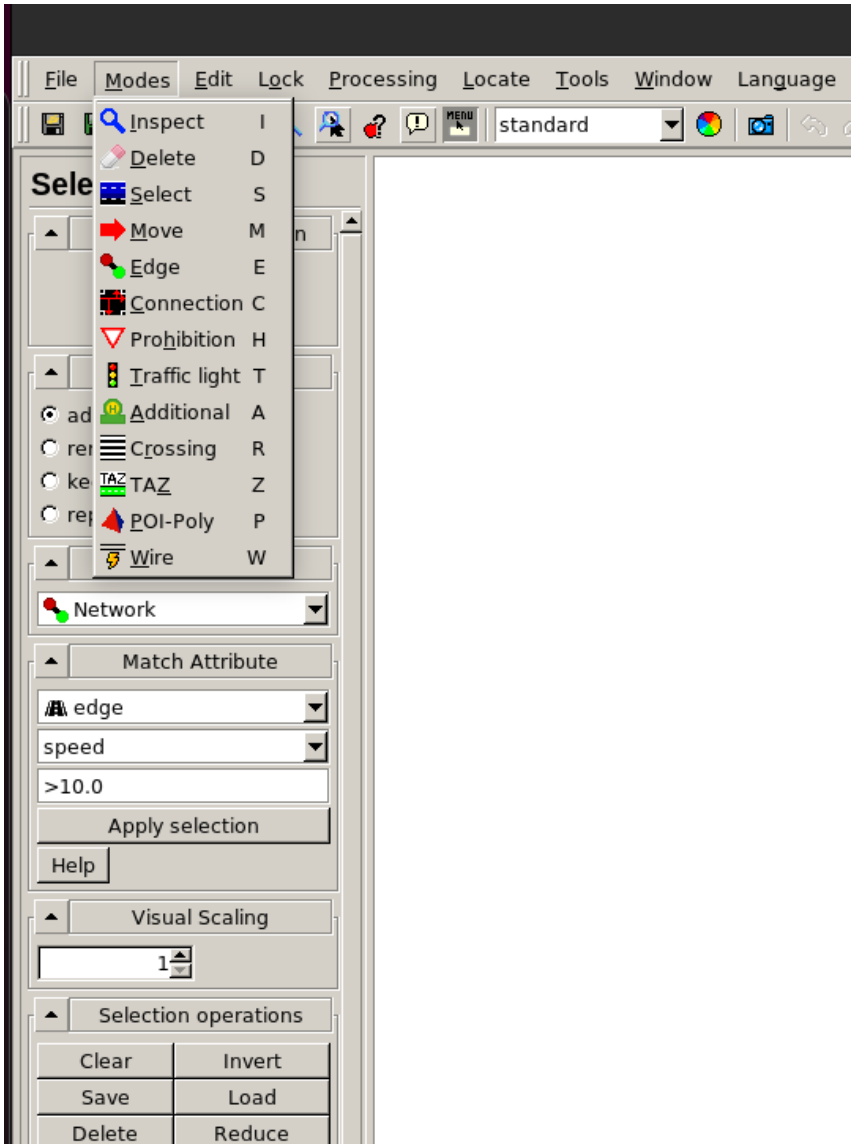

Figura 9 – Elementos para simulação

irão aparecer, bem como a duração de cada uma. Já nas paradas de ônibus, é possível definir comprimento, capacidade e o tempo que os ônibus poderão permanecer parados. O atributo *id* é comum a todos os objetos pertencentes à simulação, isto é, cada objeto terá seu *id* único.

As informações referentes à rede de tráfego, independentemente da forma como foi construída, será guardada em um arquivo com extensão *.net.xml*, que é um formato padrão do SUMO para ler e interpretar graficamente as configurações. Esse arquivo gerado irá conter informações das vias, juntamente com seus atributos, a lógica de semáforos, cruzamento das vias bem como suas regras de passagem, além de outros objetos que estejam presentes na simulação. O trecho de código abaixo demonstra como é a estrutura de um arquivo *net.xml*.

Nome	**Tipo**	**Descrição**
id	string	O identificador da faixa
index	unsigned int	Um número de execução, começando com zero na faixa mais à direita
speed	float	A velocidade máxima permitida nesta faixa [m/s]
length	float	O comprimento desta faixa [m]
shape	vetor de posição	A geometria da faixa, dada por uma polilinha que descreve a linha central da faixa; não deve ser vazia ou ter menos de duas posições

Tabela 1 – Descrição dos atributos

```
1  <type id="highway.unsurfaced" priority="1" numLanes="1"
2  speed="8.33"disallow="tram rail_urban rail rail_electric
3  rail_fast ship" oneway="0"/>
4 <type id="railway.tram" priority="17" numLanes="1"
5 speed="13.89" allow="tram" oneway="1"/>
6 <edge id=":10030246756_0" function="internal">
7 <lane id=":10030246756_0_0" index="0"
8 disallow="pedestrian tram rail_urban  rail_fast ship"
9 speed="3.90" length="2.57" shape="823.14,630.25 822.44,
10 630.22 821.89,630.41 821.50,630.85 821.27,631.51"/>
11     </edge>
```

De acordo com a documentação do SUMO (LOPEZ et al., 2018), apesar de ser um *XML* legível, esse arquivo não deve ser editado manualmente pelo desenvolvedor, pois isso aumenta consideravelmente a chance de ocorrência de erros devido à sua complexidade. Assim, ele deve ser gerado de forma automática pelo *netconvert* ou pelo *netedit*, pois assim irá incorporar de forma correta a sintaxe e os elementos necessários.

3.2 Geração de veículos e rotas

Posterior ao passo de criação do ambiente físico, a próxima etapa necessária é a criação do fluxo de trânsito, ou seja, definir quais os veículos irão fazer parte da simulação e quais as rotas que eles irão percorrer. Assim como o passo anterior, O SUMO oferece, em sua documentação, várias formas de definir os veículos, especificar as rotas que eles irão seguir, e o momento em que os veículos irão aparecer na simulação. Tais configurações podem ser realizadas de forma manual, através de bibliotecas *python* ou através de programas adicionais. Novamente, essa característica flexibiliza as opções para a modelagem da

simulação, possibilitando ao desenvolvedor criar cenários de fluxo variados. A Tabela 2 (LOPEZ et al., 2018) demonstra os principais tipos de veículos disponibilizados pelo SUMO.

Tipo de Veículo	**Descrição**
Passenger	Carros de uso pessoal
Buses	Ônibus de transporte público ou privado
Trucks	Veículos de cargas
Bicycles	Bicicletas
Pedestrians	Pedestres
Emergency	Ambulância, carros de polícia, bombeiros

Tabela 2 – Descrição dos tipos de veículos no SUMO.

De acordo com a Tabela 2, é possível notar que existem várias possibilidades de modelagem de fluxos de trânsito. Cada veículo irá possuir um conjunto de atributos que podem ser definidos, como tipo, id, velocidade máxima e mínima, distância mínima de outros veículos, altura, etc. A documentação [3] apresenta todos os atributos e os respectivos valores que cada campo pode possuir. Uma das formas de realizar essa definição é preenchendo um arquivo de extensão adicional manualmente com a extensão *.rou.xml*. Diferentemente da etapa de criação de rede de tráfego, a geração de veículos e rotas é mais simples de se fazer manualmente, isto é, a sintaxe e complexidade do arquivo a ser gerado não é alta, o que facilita essa abordagem. O exemplo abaixo apresenta em exemplo de uma definição de fluxo usando essa técnica.

```
1    <vType id="Onibus" color="red"  accel="0.8" maxSpeed="5"
2    decel="2.0" sigma="0.5"  vClass="bus"/>
3
4    <route id="Rota_onibus" edges= " v0 v1 v2 v3 ">
5    </route>
6
7    <vehicle depart="0" id="Onibus_1"
8    route="Rota_onibus" type="Onibus" />
```

No exemplo acima, no parágrafo 1, é possível perceber o momento em que o veículo é definido na simulação, contendo alguns atributos como *color*, que define a cor do veículo, os atributos *accel*, *decel* e *maxSpeed*, que interferem na velocidade do veículo durante o percurso, e o atributo *sigma*, que é o grau de imperfeição com a qual o veículo irá se comportar, ou seja, quanto maior esse valor, mais realista fica a simulação. Além disso, o atributo *VClass* define o tipo de veículo que irá aparecer na simulação. Outros valores possíveis para esse atributo são *passanger*, *motorcycle*, *truck*, como já mostrado anteriormente .

[3] https://sumo.dlr.de/docs/Downloads.php

No parágrafo 2, É definida uma rota na simulação. O atributo *id* será o identificador dessa rota, e o atributo *edges* serão as vias que estarão presentes na rota. É importante citar que essas vias devem fazer parte da rede de tráfego construída na primeira etapa do processo, caso contrário, ao tentar percorrer a rota, um erro de execução será gerado. No parágrafo 3, por fim, um veículo do tipo Ônibus está sendo instanciado, irá seguir a rota *Rota_onibus* e irá aparecer na simulação no tempo 0, definido pelo atributo *depart*.

3.3 Configurações e integração

Uma vez definidas a rede de tráfego e os veículos e rotas, o último passo é criar o arquivo de configurações, que deve ter a extensão **.sumocfg**. A partir dele, o SUMO fará a integração dos elementos para a renderização do cenário. É importante destacar que uma simulação é a junção desses três arquivos, uma vez que cada um contém uma parte dos elementos, e eles irão fornecer informações necessárias para a execução do ambiente desejado. A presença de outros tipos de arquivos para incorporar elementos adicionais também é suportada. É importante citar, ainda, que esse software trabalha com o conceito de tempo simulado, isto é, a velocidade com a qual a simulação irá ser executada. É possível variar esse tempo de 0 a 1000 milissegundos, sendo esse tempo relativo ao intervalo de execução dos passos da simulação. Abaixo, a estrutura de um arquivo de configuração. Percebe-se que sua sintaxe é simples, com a função de apenas integrar os outros arquivos.

```
1    <input>
2        <net-file value="teste.net.xml"/>
3        <route-files value="semFluxoLonga.rou.xml"/>
4        <additional-files value="additional.add.xml"/>
5    </input>
```

Adicionalmente, o SUMO possibilita a integração da simulação com uma biblioteca Python chamada TraCI (*Traffic Control Interface)* [4], que oferece uma série de funções que são capazes de recuperar valores como velocidade dos veículos, localização, aceleração, além de também permitir que sejam controlados parâmetros do ambiente,como semáforos, velocidades da via, etc (LOPEZ et al., 2018). Essa biblioteca foi desenvolvida com o objetivo de tornar ainda mais fácil o uso da plataforma SUMO para avaliar e analisar simulações de trânsito, pois permite que os desenvolvedores controlem todos os aspectos do cenário de forma simples e direta. A Figura 10 exibe os elementos que compõem uma simulação.

Através da integração com a biblioteca TraCI, o desenvolvimento de aplicações de mobilidade passou a ser mais dinâmico e interativo, pois se tornou possível atuar sobre o ambiente de simulação em tempo de execução, isto é, pode-se alterar o estado de sinais de

[4] https://pypi.org/project/traci/

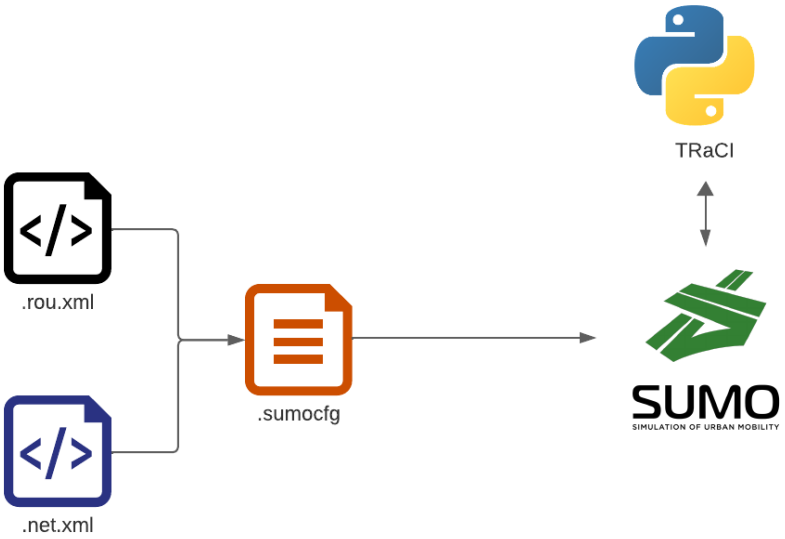

Figura 10 – Componentes de uma simulação

trânsito, velocidade dos veículos, tempos de parada dos ônibus, velocidade máximas nas vias, entre outros atributos. Desse modo, o SUMO ganhou uma aplicabilidade ainda maior para modelar e avaliar aplicações de mobilidade no contexto de cidades inteligentes, uma vez que pode-se tomar decisões de gerenciamento do fluxo de trânsito com informações em tempo real, e não mais ter cenários apenas estáticos.

4 Resultados

Essa seção apresenta os cenários de simulação de mobilidade urbana criados a partir do simulador SUMO para um contexto de cidades inteligentes. Primeiramente, foi realizada uma simulação para identificar a chegada e saída dos ônibus nas paradas de ônibus, para detectar se houve atraso desses em alguma parada. Após isso, outra simulação foi desenvolvida para coletar a velocidade média de veículos que trafegam por uma determinada avenida, de modo a calcular suas velocidades médias. Com isso, buscou-se mostrar que o SUMO possui aplicabilidade para criar e avaliar a execução de aplicações em um ambiente real de cidades inteligentes. Além disso, foi realizada uma análise dos resultados da simulação, em conjunto com o *middleware* CDPO, em métricas como acurácia e latência de detecção de eventos, de modo a verificar se o simulador pode ser integrado com outras infraestruturas.

4.1 Descrição dos Cenários

O primeiro cenário foi pensado com o intuito de detectar a chegada e o eventual atraso dos ônibus às paradas. Essa aplicação é importante para verificar se há a ocorrência de muitos atrasos, uma vez que a permanência dos alunos por muito tempo em uma parada pode ocasionar em problemas de segurança e de pontualidade. É importante destacar que esse cenário foi construído utilizando-se o mapa interno da UFMA, a fim de aproximar a simulação do contexto dos estudantes da instituição, que trafegam diariamente nesses espaços. Assim, foi desenvolvido um experimento em que ônibus circulam internamente pelas vias, e enviam seus dados contextuais para o CDPO, que detecta suas chegadas e saídas aos pontos de ônibus. Alguns eventos de atraso foram programados para verificar se o CDPO está sendo eficiente na detecção desses.

O segundo cenário especificado foi o cálculo da velocidade média de veículos em determinado trecho de uma via. Essa abordagem é interessante do ponto de vista da segurança e da mobilidade, uma vez que a partir dos resultados encontrados, é possível que as autoridades identifiquem veículos que estão em uma velocidade superior á permitida, colocando em risco o tráfego das outras pessoas, e também identifiquem situações de acidentes ou grandes congestionamentos, possibilitando uma rápida tomada de decisões. Nessa simulação, a biblioteca TraCI coleta a velocidade e localização individual de cada veículo, além de enviar esses dados para o CDPO, que verifica se os veículos estão na via e eventualmente calcula suas velocidades médias.

4.2 Cenário 1 - chegada e atraso de ônibus nas paradas

O objetivo dessa simulação foi realizar a execução de um cenário de mobilidade urbana utilizando o SUMO, na qual ônibus públicos seguem uma rota controlada e fazem paradas nos pontos de ônibus. A partir disso, é detectada a chegada e os eventuais atrasos dos ônibus às paradas através de funções da biblioteca *python* TraCI. Além disso, assim como na simulação anterior, os dados são enviados via MQTT[1] - Message Queuing Telemetry Transport, ideal para comunicação entre dispositivos IoT por conta de sua leveza e simplicidade. para o CDPO, de modo que ele instancia regras CEP e as aplica sobre esses dados, guardando o resultado em um arquivo de *log*. Após isso, esses arquivos são comparados de modo a verificar a similaridade entre eles.

Com os resultados obtidos nesse experimento, espera-se mostrar que o SUMO pode ser utilizado para criar cenários que simulem situações reais de transporte público urbano, podendo ajudar a encontrar soluções efetivas para problemas como insuficiência de frotas, atrasos de ônibus e melhores rotas.

4.2.1 Metodologia de condução do experimento

Primeiramente, utilizando o *OpenStreetMap* [2], foi reproduzido uma parte do mapa da Cidade Universitária da UFMA. Ao realizar a importação do mapa, alguns ajustes tiveram que ser realizados manualmente através do *netedit*, pois ocorreram alguma falhas na via, como a sobreposição de estradas, e tiveram de ser corrigidos. Além disso, as vias foram renomeadas para se ter uma facilidade maior no momento de definir a rota dos veículos. A Figura 11 mostra o mapa após importação e ajustes.

Foram definidas 8 paradas ao longo do trajeto. A posição dessas paradas na simulação foi baseada nas posições reais, obtidas através do site *Moovit*[3]. Após elencar as paradas, o próximo passo foi definir o fluxo de veículos e suas rotas no arquivo *.rou.xml*. Foram definidos 4 ônibus para compor a simulação, que simulam a rota que os ônibus reais fazem dentro da UFMA. O código abaixo apresenta a definição dos ônibus e suas respectivas rotas.

```
1 vTypeDistribution id="mixed">
2   <vType id="Onibus_monitorado" color="red"  accel="0.8" maxSpeed
      ="5" decel="2.0" sigma="0.5"  vClass="bus"/>
3 </vTypeDistribution>
4
5 <route id="Rota_onibus" edges= "k3 k2 k1 c4 j1 j2 j3 j4 c0 c1 c2
    p1 p2 k8 k7" repeat="5">
```

[1] https://mqtt.org/
[2] https://www.openstreetmap.org/
[3] https://moovitapp.com/

Figura 11 – Mapa da UFMA

```
6         <stop busStop="bs_v0" duration="10.00"/>
7 </route>
```

A partir desse recorte de código, perceba que da linha 1 à linha 3 está sendo definido um veículo do tipo `bus`, que representa um transporte público no SUMO, além de seus atributos. Na linha 6 está sendo especificada a rota que os ônibus deverão seguir. Nesse experimento, apenas uma rota foi definida, isto é, todos os veículos seguirão a mesma rota e irão repeti-la 5 vezes. Do mesmo modo, também é possível observas nas linhas 7 à 14 a definição das paradas que os ônibus deverão obedecer. Essas paradas possuem um *id* que as identificam e a duração na qual cada ônibus deverá demorar, definido com o tempo de simulação 10. É importante citar que essas paradas de ônibus já devem ter sido previamente criadas no **netedit** antes de serem definidas nesse arquivo. Posterior a esses passos, os ônibus foram instanciados:

```
1 <vehicle depart="0" id="Onibus_monitorado_1" route="Rota_onibus"
     type="Onibus_monitorado" />
```

```
<vehicle depart="10" id="Onibus_monitorado_2" route="Rota_onibus"
    type="Onibus_monitorado"  />
<vehicle depart="11" id="Onibus_monitorado_3" route="Rota_onibus"
    type="Onibus_monitorado"  />
<vehicle depart="498" id="Onibus_monitorado_4" route="Rota_onibus
   " type="Onibus_monitorado"  />
```

Foram instanciados 4 ônibus, os quais seguem a mesma rota. Contudo, o atributo `depart`, que determina o momento com a qual os ônibus irão aparecer na simulação, se diferencia. Isso ocorre para representar os diferentes horários de circulação dos ônibus em um ambiente real. Nesse sentido, ao iniciar a simulação, cada ônibus surge em seu tempo determinado e inicia seu percusso e suas paradas.

Na medida que os ônibus percorriam seus trajetos, os dados de suas posições e velocidades eram colhidos a partir da biblioteca TraCI, de modo a ser verificado se eles chegaram à alguma parada e para envio dessas informações para o CDPO. Abaixo, o código que realiza essa função.

```
if onibus_monitorado_1 in traci.vehicle.getIDList():
    position1 = traci.vehicle.getPosition(onibus_monitorado_1)
    speed1 = traci.vehicle.getSpeed(onibus_monitorado_1)
```

A função `getPosition` retorna uma tupla com as coordenadas referentes à latitude e longitude do veículo. Essa função é particularmente importante pois permite que aplicações sejam modeladas levando em consideração a localização dos elementos nos ambientes físicos reais. Após detectar a chegada de um ônibus à parada, foi gravado em um arquivo de *log* o *timestamp* desse evento e a identificação do ônibus, bem como a exata parada que erou esse evento. Abaixo, um recorte do arquivo gerado.

```
{"timestamp_left":"2023-07-27 09:33:08", "bus_id":"
   Onibus_monitorado_1", "bus_stop":"PF"}
{"timestamp_left":"2023-07-27 09:33:26", "bus_id":"
   Onibus_monitorado_2", "bus_stop":"PF"}
{"timestamp_left":"2023-07-27 09:34:27", "bus_id":"
   Onibus_monitorado_1", "bus_stop":"CCH"}
{"timestamp_left":"2023-07-27 09:34:45", "bus_id":"
   Onibus_monitorado_2", "bus_stop":"CCH"}
{"timestamp_left":"2023-07-27 09:35:24", "bus_id":"
   Onibus_monitorado_1", "bus_stop":"CCET"}
{"timestamp_left":"2023-07-27 09:35:42", "bus_id":"
   Onibus_monitorado_2", "bus_stop":"CCET"}
```

Além das chegadas, foi modelado um atraso programado nos ônibus, a fim de verificar se o SUMO conseguiria detectar esses atrasos na parada em que eles ocorressem. Um evento de atraso deveria ser disparado cada vez que uma parada ficasse 10 minutos ou mais sem receber algum ônibus. Através da biblioteca TraCI, nos intervalos de tempo simulado `400 - 1000 e 2500 - 3100`, a velocidade dos ônibus é zerada, o que faz com que eles fiquem parados cerca de 10 minutos. Abaixo, o trecho de código que altera a velocidade dos ônibus.

```
1  if tempo_simulado > 400 and  tempo_simulado < 700 :
2      for bus in lista_veiculos:
3          traci.vehicle.setSpeed(bus,0)
4
5  elif tempo_simulado > 702 and  tempo_simulado <= 1001 :
6      for bus in lista_veiculos:
7          traci.vehicle.setSpeed(bus,0)
8
9  elif tempo_simulado > 2500 and  tempo_simulado <= 2800:
10     for bus in lista_veiculos:
11         traci.vehicle.setSpeed(bus,0)
```

A partir da observação do código acima, é importante destacar que o SUMO só consegue deixar um veículo parado por 300 passos de simulação, pois após isso, ela altera sua velocidade de forma automática. Por esse motivo, o atraso foi definido como uma sequência de paradas isoladas, para que fosse respeitado o tempo limite. É possível notar, portanto, que é feita uma verificação do tempo da simulação, e caso a condicional seja atendida, os ônibus presentes na simulação irão permanecer parados. Quando há a ocorrência de um atraso, ele é registrado, juntamente com o *timestamp* e a parada que o atraso foi detectado, em um arquivo de *log* local. Nesse experimento, foram programados 14 atrasos, distribuído entre as paradas. A seguir, a lógica utilizada para detectar os atrasos.

```
1  if (timestamp - tempo_chegada) >= datetime.timedelta(minutes=10):
2      atrasotxt.write(f" {timestamp_str} - Atraso do Onibus para
         chegar    parada do CCSO\n")
```

Quando um ônibus chega em uma parada específica, é guardado o *timestamp* desse evento, de modo que a variável `tempo_chegada` sempre guarde o momento da última parada. Após isso, é feita uma verificação contínua entre o tempo atual e a última chegada de cada parada, de modo que se essa diferença exceder 10 minutos, um evento de atraso é gravado no arquivo de *log*. A 12 demonstra a execução da simulação, com a presença de ônibus chegando à uma parada.

Os dados dos ônibus são enviados para um tópico MQTT, a fim de possibilitar que o CDPO se subscreva e realize processamento sobre eles. São enviados os dados de

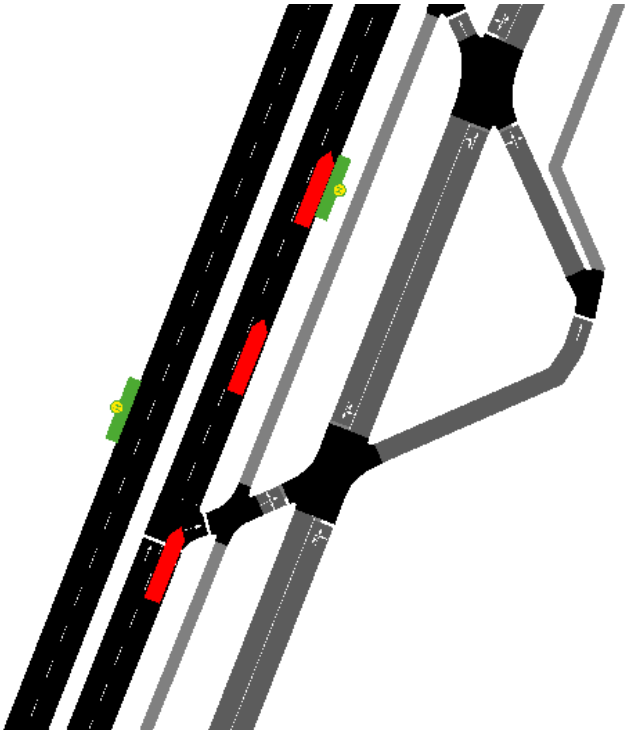

Figura 12 – Execução da simulação

id, velocidade, localização e *timestamp*.O trecho da EPN abaixo ilustra como o CDPO detecta os eventos de chegada dos ônibus à uma parada da simulação. Como já citado, a simulação contém 8 paradas, e portanto, as demais seguem a mesma lógica do exemplo mostrado.

```
1  {
2  {
3          "name":"BusArivedPF",
4          "description":"Detectar a chegada do onibus na parada do
               PF",
5          "qos":"AT_MOST_ONCE",
6          "level":"EDGE",
7          "target":"CLOUD",
8          "tag_filter":"type:bus",
9          "definition":"select a.lastOf().id as bus_id, 'PF' as
              bus_stop, a.firstOf().timestamp as timestamp_arrived,
              a.lastOf().timestamp as timestamp_left from pattern[
              every(a=Vehicle(latitude = -2.55932 and longitude = -4
              4.31212) until Vehicle(latitude != -2.55932 or
              longitude != -44.31212))] where a is not null",
10         "event_type":"Vehicle",
11         "event_attributes":{
12            "id":"string",
13            "latitude":"double",
```

```
            "longitude":"double",
            "speed":"double",
            "timestamp":"string"
        },
        "output_event_type":"BusArived"
    }
  ]
}
```

Observando a EPN acima, no campo `definition` é possível notar que essa regra utiliza a coordenada fixa da parada, e verifica quando a coordenada dos ônibus são iguais à ela, com a finalidade de identificar a chegada desses ônibus na parada, gerando, assim, o evento `BusArived`. É importante observar que essa regra atua no nível da *edge* (informação contida na tag `level`), e tem o destino da camada `cloud`, isto é, quando é gerado um evento de chegada, ele é enviado para a camada superior.

A camada *cloud*, por sua vez, ao receber os eventos `BusArived`, irá realizar o processamento para verificar se há a ocorrência de atrasos, através da seguinte EPN:

```
"webhook_url":"http://cdpo:8080/logger",
  "rules":[
    {
      "name":"LateBus",
      "description":"Padrao para detectar se o onibus esta
        atrasado",
      "qos":"AT_MOST_ONCE",
      "level":"CLOUD",
      "target":"WEBHOOK",
      "tag_filter":"block:ufma",
      "definition":"select a.bus_stop as parada_com_atraso,
        current_timestamp.format(\"yyyy.MM.dd HH:mm:ss\") as
        timestamp from pattern[every a=BusArived -> (timer:
        interval(10 min) and not BusArived(bus_stop=a.bus_stop
        ))]",
      "output_event_type":"LateBus"
    },
```

Nessa regra, o objetivo é receber os eventos `BusArived` e aplicar a primitiva de padrão para detectar uma sequência em que há mais de 10 minutos de intervalo entre um evento e outro. Quando essa condição é detectada, um evento `LateBus` é gerado. Essa EPN executa no nível *cloud* e seu resultado é direcionado para o *webhook* definido.

4.2.2 Resultados obtidos

Após a execução da simulação, foi verificado o número de eventos de chegada que o SUMO conseguiu detectar, isto é, se o simulador monitorou efetivamente os ônibus da simulação, registrando suas chegadas às respectivas paradas. Como já citado anteriormente, foram modeladas 8 paradas ao longo do trajeto, de modo que cada um dos 4 ônibus presentes realizasse uma parada a cada vez que passasse por um desses pontos. Como a simulação consistiu nos veículos realizarem, cada um, 6 voltas no total, fazendo uma conta simples, percebe-se são esperados 8(paradas) `X` 6 (voltas) `X` 4(ônibus) = 192 eventos de chegada.

Assim, após análise dos resultados, foi constatado que o arquivo de *log* continha todos os 192 registros de chegada. Contudo, também foi analisado se o SUMO conseguiu fazer a relação entre as paradas e os ônibus, ou seja, se foi possível detectar quando um ônibus específico chegava à uma parada específica. Assim, a Figura 13 apresenta a quantidade de paradas de cada ônibus.

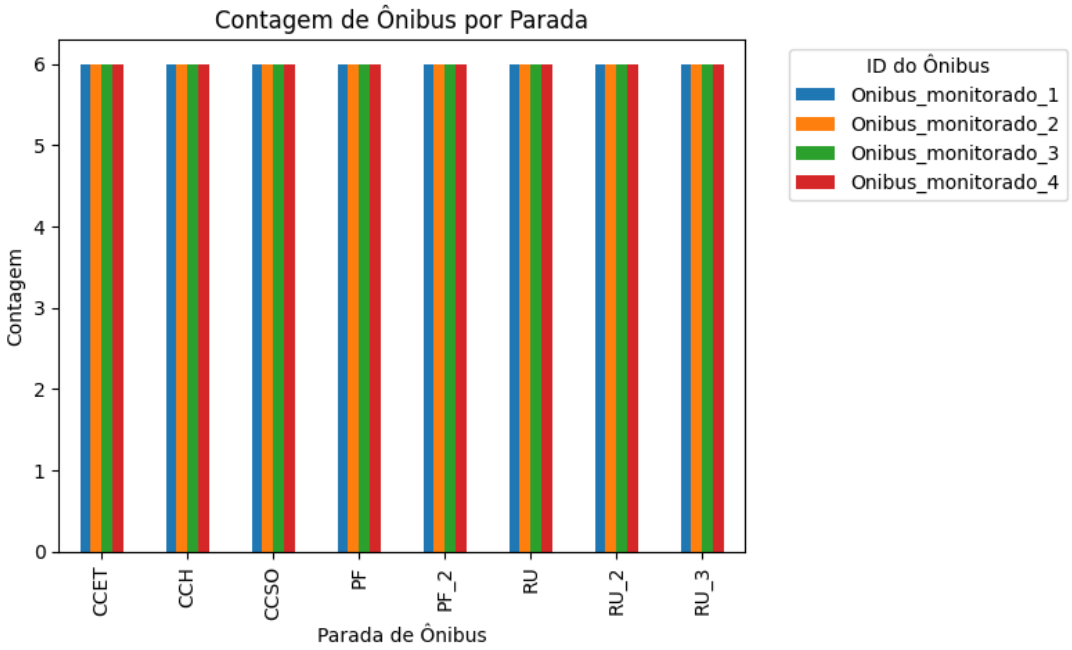

Figura 13 – Quantidade de eventos de chegada

É possível, a partir da Figura 13,notar que cada parada da simulação recebeu cada ônibus 6 vezes, referente às 6 voltas que estavam programadas, ou seja, em cada percusso que um veículo realizou, ele obedeceu a regra especificada. Desse modo, pode-se afirmar que o SUMO pôde processar a informação de cada ônibus de forma individual, não havendo perca de informações ou falhas na execução do cenário proposto.

Além dos eventos de chegada, foram programados ao todo 14 eventos de atraso para os ônibus. De forma similar, foi necessário verificar se o simulador foi capaz de detectar tais atrasos, que deveriam ser gerados cada vez que a ausência de veículos nas

paradas excedesse o tempo de 10 minutos. Como já demonstrado anteriormente, ao ocorrer um atraso, o SUMO registrou esse evento em um arquivo de *log*, e assim, após análise desse arquivo, ratificou-se sua capacidade de modelar e identificar uma situação conforme desejado. A Tabela 3 apresenta a quantidade de eventos de atraso identificados em cada parada.

Tabela 3 – Contagem de Atrasos SUMO

Parada	**Contagem**
PF	2
CCH	2
CCET	2
RU	2
RU_3	2
CCSO	2
RU_2	1
PF_2	1

Assim, com base na Tabela 3, é observável que todos os 14 eventos de atrasos foram detectados, bem como a parada responsável por gerar cada evento. Com isso, o SUMO se apresenta como uma boa alternativa para a modelagem e execução de aplicações de mobilidade urbana que tenham foco no transporte público, pois se mostrou capaz de oferecer suporte à criação de rotas controladas, implementação e controle de paradas de ônibus, e possibilitar a programação de eventos inesperados como atrasos, além de registrar efetivamente a ocorrência desses eventos.

Adicionalmente, após o processamento CEP aplicado sobre os dados através do CDPO, foi analisado se os resultados obtidos foram similares aos resultados gerados pelo SUMO. Novamente, buscou-se verificar se o simulador foi efetivo ao enviar os dados para o *middleware*. Realizando uma primeira análise sobre os arquivos gerados, foi possível perceber que o CDPO registrou a mesma quantidade de eventos de chegadas e de atrasos que o SUMO, totalizando **192** eventos de chegada e **14** eventos de atrasos. Isso significa que as EPNs foram eficientes, e assim, conseguiram registrar todos os eventos ocorridos. Contudo, é preciso verificar a acurácia do CDPO, ou seja, se ele conseguiu ter uma alta taxa de acerto ao identificar eventos de chegada e de atraso. Para realizar tal verificação, primeiramente foi analisado o *timestamp* dos eventos de chegada, para analisar se os tempos de detecção foram similares. A Figura 14 faz a comparação entre o horário de chegada em cada parada no SUMO e no CDPO.

O gráfico 14 indica que quando um ônibus chega a uma parada no SUMO, o CDPO prontamente realiza a detecção. Isso pode ser percebido pois os pontos azuis e laranjas se sobrepõem em quase todo o gráfico, o que indica que não houveram perca de informações. Nota-se, ainda, que o CDPO registrou a chegada dos ônibus a cada parada em um tempo bem próximo à ocorrência desse evento, destacando, assim, sua acurácia e precisão. Tais

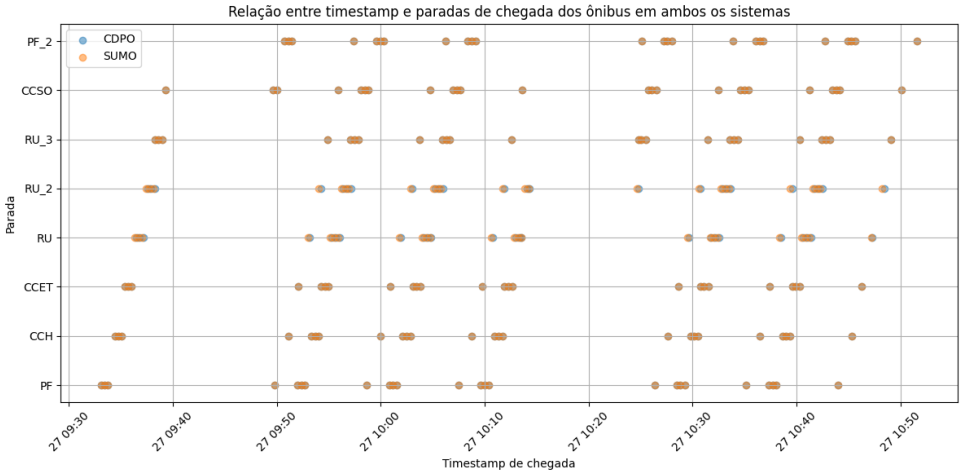

Figura 14 – Chegada dos ônibus por parada

fatores ratificam que além do SUMO ser eficaz no envio correto dos dados, isso é feito em um tempo bem rápido, e nesse exemplo, onde há um processamento de dados em mais de uma camada, essa característica é fundamental para a aplicabilidade do simulador.

Para facilitar a visualização dos dados de *timestamp*, foram calculadas algumas estatísticas entre os dois arquivos de chegada. A Tabela 4 demonstra esses valores.

Tabela 4 – Latência da detecção dos eventos de chegada

Estatística	**Valor**
Latência Média	0.625 s
Latência Máxima	5.0 s
latência Mínima	0.0 s
Desvio Padrão	1.09017 s

A partir da Tabela 4, é possível perceber que a média de diferença entre o tempo de ocorrência do evento e sua detecção no CDPO foi muito baixa, pouco mais de meio segundo. Entretanto, foi necessário observar se o CDPO também conseguiu detectar todos os eventos de atraso, pois eles foram gerados de forma controlada e apenas em alguns momentos da simulação. Novamente, o CDPO gerou bons resultados. A Figura 15 realiza essa verificação, relacionando o timestamp do atraso e a parada em que ele foi detectado, tanto no *log* do SUMO quanto no CDPO.

A partir desse gráfico, é possível notar que os pontos estão quase completamente sobrepostos, e assim, constata-se a similaridade entre os pontos do SUMO (pontos laranjas) e entre os pontos do CDPO (pontos azuis). Isto significa que o CDPO foi muito eficiente ao detectar o atraso dos ônibus com uma baixa latência. Todos os 14 atrasos programados para o experimento foram detectados pelo CDPO, nas mesmas paradas em que eles de fato ocorreram.

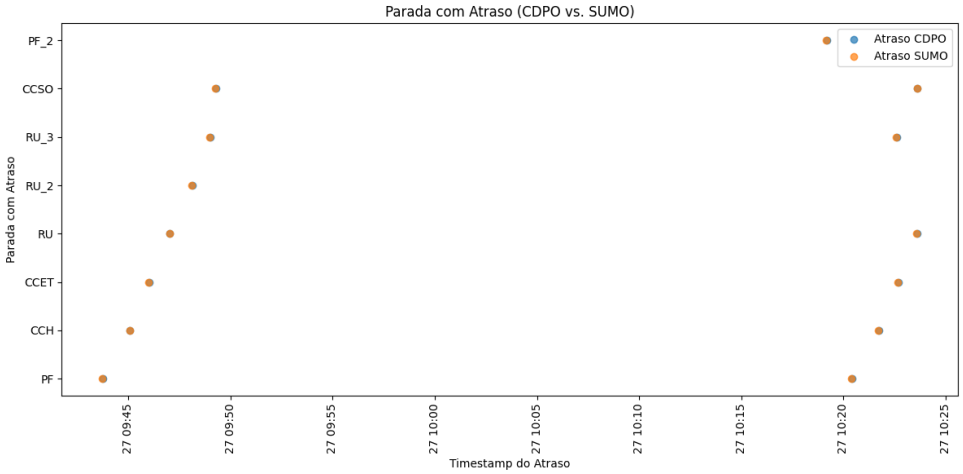

Figura 15 – Atraso dos ônibus por parada

4.2.3 Conclusão

Após a execução da simulação e da análise das informações, observou-se que o SUMO disponibilizou todas as ferramentas e funções necessárias para a criação do cenário e execução da aplicação, pois executou de forma correta todos os eventos programados previamente. Isso demonstra o potencial do simulador para contextos de mobilidade urbana onde se deseja desenvolver e avaliar soluções que envolvam o transporte público, sendo útil para a análise de problemas existentes, oferecendo possibilidades de se testarem soluções onde o cenário de simulação corresponde à realidade, diminuindo a possibilidade de erros, custos e poupando tempo, uma vez que permite a criação de elementos e regras encontradas em ambientes reais.

4.3 Cenário 2 - velocidade média dos veículos

O objetivo principal desse experimento foi desenvolver um ambiente de mobilidade urbana a partir do SUMO, em que veículos trafegam por uma avenida e tem suas localizações e velocidades coletadas a partir da biblioteca TraCI. O intuito dessa coleta de dados é verificar se o veículo está em uma determinada via, e caso esteja, é calculada a velocidade média desse veículo naquele trecho, bem como calcular a velocidade média total da via. Além disso, a biblioteca TraCI também envia os dados do veículo para o *middleware* CDPO, de modo que um processamento dessas informações seja realizada em uma arquitetura de múltiplas camadas, através de regras CEP.

Através desse experimento, busca-se mostrar que o SUMO pode ser utilizado para criar aplicações do mundo real que possuem a finalidade de monitorar a velocidade de veículos em áreas urbanas, seja a partir de câmeras inteligentes ou sensores de velocidades localizados em pontos estratégicos das vias, seja a partir de sensores de velocidades instalados nos próprios veículos.

4.3.1 Metodologia de condução do experimento

Como o mapa utilizado nessa simulação foi o mesmo do cenário 1, o processo de importação e edição das vias não precisou ser refeito. O segundo passo foi definir o fluxo de veículos e a rota que eles iriam percorrer ao longo da simulação. Para esse experimento, foi definido o tipo de veiculo *passenger*, que é um tipo padrão que o SUMO disponibiliza para simular carros comuns de passeio. A escolha desse veículo se baseia no fato de que o trânsito interno da UFMA é composto majoritariamente por esses carros. Posteriormente, uma rota foi estabelecida para esses veículos, de modo que seja conhecido o caminho que eles estão percorrendo durante a execução da simulação. O número de veículos foi definido como 10 para se ter um número maior de velocidades coletadas, e o número de voltas que cada veículo realizará foi de 16. Os trechos abaixo demonstram o arquivo *rou.xml* contendo as definições de veículos e rotas.

```
1 <vType id="Carro_comum"  accel="3.0" decel="6.0"
2     minGap="2.5" maxSpeed="50.0" sigma="0" vClass="passenger"/>
```

Nessa parte do código, é especificado o tipo de veículo que estará contido no ambiente de simulação. A classe do veículo é o valor contido no atributo `vClass`, *"passanger"*. Além disso, são definidos outros atributos como `id, accel, decel, minGap, maxSpeed e sigma`. Como já citado em seções anteriores, o valor de cada atributo irá definir o comportamento do veículo na simulação, e podem ser alterados de acordo com a necessidade do problema. É importante citar que mais de uma classe veículos poderiam ter sido definidas, contudo, para a modelagem desse cenário, apenas um tipo foi necessário. Após a definição dos veículos, foi criada a rota que eles iriam seguir no mapa.

```
1 <!-- Definindo rotas -->
2 <route id="Rota_controlada" edges= "k6 k5 k55 k4 k3 k2 k1 c4 j1 j
     2 j3 j4 j5 j6 j7 j8 j9 k6" repeat="15" >
3 </route>
```

O trecho de código acima contém o *id* da rota, as *edges*, isto é, as vias que compõem o caminho a ser percorrido, e o número de repetições que devem ser realizadas pelos veículos. É importante destacar que caso a simulação contenha mais de uma rota, cada uma deverá conter um *id* próprio. Adicionalmente, os valores presentes no atributo *edges* devem ser o nome das vias que foram definidos no *netedit*, e cada via subsequente deve estar fisicamente conectada na simulação, caso contrário, um erro de execução será gerado. Por fim, definidos os veículos e rotas, criou-se o fluxo de trânsito. O código abaixo demonstra essa etapa.

```
1 <!-- Criando fluxo -->
2 <interval begin="0" end = "3600">
```

```
3         <flow id="car_" route="Rota_controlada" number="10" type
              ="Carro_comum" period="0"/>
4 </interval>
```

É possível perceber, a partir do trecho acima, que foi estabelecido um fluxo que inicia no tempo 0 e vai até o tempo 3600 da simulação. Esse tempo é referente ao tempo simulado, já citado anteriormente. Isso significa que irão haver 3600 passos de tempo a serem executados. O atributo `route` contém o nome da rota que o fluxo irá seguir, e deve ser uma rota já definida anteriormente. Há também o número de veículos que irão ser criados, além do tipo de veículos e do perído de surgimento entre eles. É necessário observar que o valor no atributo `type` também já deve ter sido definido anteriormente.

Após isso, o próximo passo foi a integração da simulação com a biblioteca TraCI, uma vez que é possível utilizar as funções da biblioteca para recuperar valores do ambiente, como velocidade e posição dos veículos. O trecho abaixo mostra o código *python* referente à essa integração. Observa-se que é uma integração sem grandes complexidades.

```
1 traci.start(["sumo-gui", "-c", "teste.sumocfg"])
2 traci_connection = traci.connect(port=8815)
```

Na execução dessa simulação, é importante citar que o veículo não mantém a velocidade constante ao longo de todo o percurso, pois foi criada uma lógica para que periodicamente essa velocidade seja alterada aleatoriamente. Isso se deu pelo fato de a velocidade não ser uma variável previsível para o cálculo da velocidade média. Abaixo, a demonstração da tática utilizada.

```
1 if tempo_simulado % 100 == 0 :
2     num_aleatorio = randint(5,30)
3     traci.vehicle.setMaxSpeed(vehicle, num_aleatorio)
```

Através desse código, percebe-se que a cada 100 passos de simulação executados, um número aleatório entre 5 e 30 era calculado e definido como a velocidade máxima do veículo, ou seja, sua velocidade poderia variar entre 1 e o valor limite. A escolha desses valores se deu pelo fato de que a métrica utilizada pelo simulador é `metros/segundos`.

Para a execução do experimento, foi considerado que se desejava calcular a velocidade média dos veículos que passavam por uma das vias da instituição. Dessa forma, quando um veículo adentrava essa via, sua velocidade passava a ser coletada e adicionada a um dicionário de velocidades. Quando o veículo sai da via, a média dessa velocidade é calculada. A cada volta, o cálculo da média é registrada em um *log* da simulação, associada ao *id* do carro que a gerou, bem como o *timestamp* em que a informação foi registrada. Esses procedimentos estão presentes no código abaixo

```
1         via = "j4"
```

```
2         if vehicle in traci.edge.getLastStepVehicleIDs(via):
3             if vehicle in velocidades:
4                 velocidades[vehicle].append(speed * 3.6)
5             else:
6                 velocidades[vehicle] = [speed * 3.6]
7
8         if vehicle in traci.edge.getLastStepVehicleIDs("j6") and
              len(velocidades[vehicle]) > 0:
9             velocidadeMedia = sum(velocidades[vehicle]) / len(
                  velocidades[vehicle])
10            logging.info(f"Velocidade_Media {vehicle} : {
                  velocidadeMedia}")
```

Na linha 1 do código, é definida a via na qual se deseja monitorar a velocidade média dos veículos. Na primeira condicional, é verificado se o veículo está presente na via, e caso esteja, é criado um dicionário para guardar suas velocidades ao longo dessa via. Na segunda condicional, é realizada o cálculo da média de velocidade dos veículos, bem como a escrita dessa informação em um arquivo de *log*. A Figura 16 demonstra a execução da simulação.

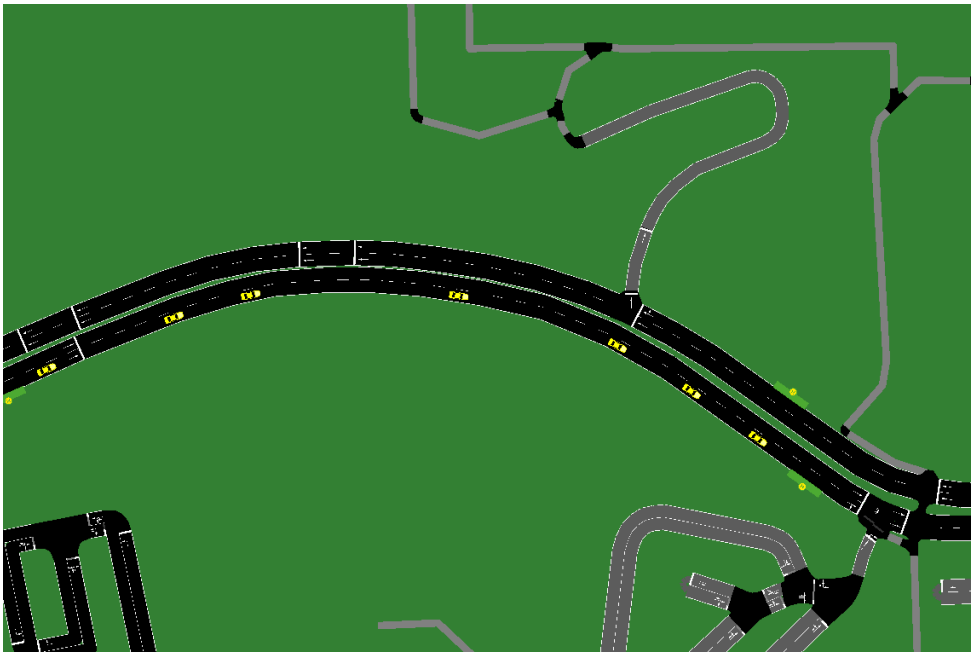

Figura 16 – Execução da simulação

Os passos acima descrevem toda a execução da simulação no software SUMO e o registro dos resultados no arquivo de *log*, presente na máquina local. Contudo, além desse processamento realizado pelo próprio SUMO, foi feito um processamento através do CDPO para verificar se é possível utilizar os dados desse simulador em outras infraestruturas. Nesse sentido, foi realizado o envio dos dados para o CDPO por meio do protocolo de

comunicação MQTT. Através desse protocolo, os dados de velocidade de cada veículo, juntamente com sua localização e seu *id*, foram publicados em um tópico específico, na qual o CDPO se inscreveu e pôde coletar essas informações para realizar o cálculo de velocidade média de cada veículo. O trecho de código abaixo demonstra o envio desses dados para um tópico MQTT.

```
# Inicializar o cliente MQTT
client = mqtt.Client()
client.on_connect = on_connect

# Conectar ao broker MQTT
client.connect("localhost", 1883, 60)

client.publish("dadosVehicle/{}".format(vehicle), dados)
```

Do outro lado, o CDPO se subscrevia nesse tópico e passava a receber esses dados, e então, utilizava uma EPN para calcular a velocidade média desses veículos. A EPN utilizada foi a seguinte:

```
"webhook_url":"http://cdpo:8080/webhook",
   "rules":[
      {
         "name":"CarMedia",
         "description":"Media de cada carro que passa pela
            avenida",
         "qos":"AT_MOST_ONCE",
         "level":"EDGE",
         "target":"WEBHOOK",
         "tag_filter":"type:vehicle",
         "definition":"select a.firstOf().id as car_id, a.average
            (i => i.speed) as media, current_timestamp.format(\"
            yyyy.MM.dd HH:mm:ss.SSS\") as timestamp from pattern[
            every (a=Vehicle(isLocationInArea(latitude,longitude,-
            2.557047,-44.309050,0.14) = True) until Vehicle(
            isLocationInArea(latitude,longitude,-2.557047,-44.3090
            50,0.14) = False))] where a.firstOf().id is not null",
         "event_type":"Vehicle",
         "event_attributes":{
            "id":"string",
            "latitude":"double",
            "longitude":"double",
            "speed":"double"
         },
```

```
18            "output_event_type":"CarSpeedAverage"
19        }
20    ]
21 }
```

A partir dessa EPN, é possível observar que essa regra utiliza a primitiva de padrão para verificar se o veículo está dentro de uma área circular. Para realizar essa verificação, o CDPO utiliza os dados contextuais de localização enviados pelos veículos via MQTT, e os aplica à função java externa `isLocationInArea`, que é baseada na fórmula de *haversine*, que através de um ponto geográfico e um raio, é capaz de verificar se um determinado objeto está em dentro de uma área circular. Enquanto o veículo está nessa área circular, sua velocidade é acumulada, e após ele sair, a cláusula `average`, do esper, é utilizada para calcular a média dessas velocidades coletadas. Essa EPN é executada no nível *Edge* e o evento de saída contendo a velocidade média é enviado para o *webhook* definido, além de serem gravados em outro arquivo de *log*, para permitir comparações posteriores.

4.3.2 Resultados Obtidos

Após a modelagem e execução da aplicação de velocidade média, o SUMO gerou um arquivo de *log* na qual registrou o *id* dos veículos, juntamente com a velocidade média e o *timestamp* de cada volta realizada. Como demonstrado anteriormente, essa simulação foi composta por 10 veículos, cada um realizando 16 voltas ao longo da via, e portanto,se a cada vez que esses veículos saíssem da via suas velocidades médias fossem calculadas, era esperado que houvessem 160 medições no total. Foi verificado, através do arquivo gerado, que o SUMO conseguiu realizar todas as medições esperadas, ou seja, o simulador conseguiu coletar a velocidade de cada veículo individualmente e calcular suas velocidades médias. A seguir, um recorte dos registros presentes no arquivo de *log* gerado.

```
1 "media":41.55547911097275, "car_id":"car_.5", "timestamp":"2023.0
    8.18 18:48:33.979"}
2 {"media":50.52368514512781, "car_id":"car_.2", "timestamp":"2023.
    08.18 18:48:36.002"}
3 {"media":47.85064756613061, "car_id":"car_.0", "timestamp":"2023.
    08.18 18:48:37.995"}
4 {"media":46.897968149441944, "car_id":"car_.8", "timestamp":"2023
    .08.18 18:48:42.017"}
5 {"media":52.29476249214077, "car_id":"car_.6", "timestamp":"2023.
    08.18 18:48:56.031"}
6 {"media":53.84743468837412, "car_id":"car_.3", "timestamp":"2023.
    08.18
```

Percebe-se, logo, que o simulador pôde representar um sistema de monitoramento que pode ser implementado em um ambiente real, na qual se faz a coleta da velocidade de veículos a partir de sensores externos ou internos, e gravam-se informações de identificação do veículo, bem como suas respectivas velocidades e a data de medição.

Para a avaliação dos resultados do processamento de dados realizado pelo CDPO, foram comparados os resultados presentes nos dois arquivos de *log*, com a finalidade de verificar a similaridade entre eles, e assim, verificar a acurácia e a latência do processamento CEP sobre os dados enviados pelo SUMO. Com essa avaliação, é possível dizer se o simulador consegue ser efetivo ao realizar o envio de seus dados para outras plataformas para possibilitar o tratamento e processamento externo.

Primeiramente, houve a verificação de quantos registros de velocidade média cada sistema conseguiu detectar. A partir dessa análise, é possível detectar se houveram perdas de informações no processo de coleta do CDPO. Como já havia sido constatado que o SUMO havia realizado 160 medições de velocidade média, foi observado quantas medições o CDPO conseguiu processar. Foi checado que houveram 160 registros de velocidade média também no arquivo de *log* do *middleware*, divididos igualmente entre os 10 veículos que compuseram a simulação, ou seja, 16 registros por *id*.

Percebe-se, logo, que o CDPO detectou todos os eventos ocorridos na simulação, não havendo perca de informações no momento da coleta. Contudo, é necessário observar se a EPN instanciada pelo *middleware* teve uma boa taxa de acertos no cálculo das velocidades médias, isto é, se conseguiu encontrar os mesmos valores calculados pelo SUMO, pois dessa forma, pode-se verificar se o simulador conseguiu enviar os dados de forma correta. A Figura 17 realiza uma comparação entre a média dos valores de velocidade calculadas em cada sistema.

A partir desses resultados, é possível observar que o CDPO obteve valores bem próximos em comparação aos valores calculados pelo SUMO, ou seja, o *middleware* conseguiu ter uma boa acurácia no cálculo da média das velocidades para todos os veículos da simulação. Para entender melhor as similaridades entre os resultados, a Tabela 5 demonstra alguns valores estatísticos de cada sistema.

Tabela 5 – Estatísticas de Velocidade

Estatística	**CDPO**	**SUMO**
Média de Velocidade	39.914548452131505	39.93211286563352
Média Máxima	45.12394143121509	45.20728330272618
Média Mínima	37.55980451825037	37.57389440979114
Desvio Padrão	2.2262476435062815	2.2370684266077356

Observando os valores da Tabela 5, é possível notar que a média de velocidade do SUMO e do CDPO foram bem similares, com diferenças apenas nas casas decimais.

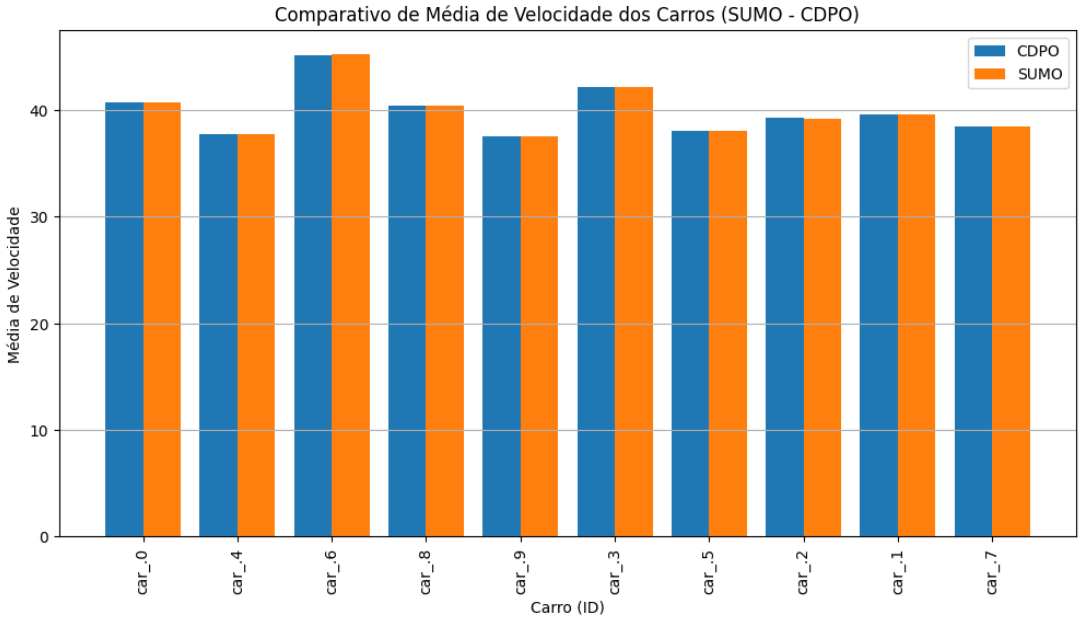

Figura 17 – Média de velocidades

O mesmo se repete para os valores de Velocidade Máxima e Mínima, bem como para o Desvio padrão. Desse modo, nota-se que o simulador foi efetivo ao realizar o envio dos dados de velocidade e localização via MQTT para o *middleware*, uma vez que ambos encontraram os mesmos resultados.

Por fim, uma última análise realizada consistiu em avaliar o *timestamp* com a qual o CDPO registrou a velocidade média dos veículos. Essa análise é necessária para verificar se houve uma latência muito grande entre a ocorrência do evento no SUMO e a captação desse mesmo evento no CDPO. Essa análise se faz necessária pois em um contexto real de cidades inteligentes, muitas aplicações de mobilidade exigem que o processamento dos dados seja feito em um tempo próximo ao real, para que decisões possam ser tomadas de forma eficiente. Para realizar essa verificação, foi observado o horário em que a velocidade média de cada veículo foi registrada nos arquivos de *log* do SUMO e do CDPO, e após isso, calculada a diferença entre o registro de cada volta. Desse modo, foi possível encontrar alguns dados estatísticos que relacionam os resultados. A Tabela 6 os apresenta.

Tabela 6 – Latência da detecção de eventos

Estatística	**Valor**
Latência Média	1,8167 segundos
Latência Máxima	3.984 segundos
Latência Mínima	0.903 segundos

A partir dos dados da Tabela 6, é perceptível que não houve uma grande atraso entre a ocorrência do evento e sua detecção pelo CDPO, ficando a média em menos de dois segundos. É importante destacar que a comunicação via protocolo MQTT está sujeita às variações de rede, contudo, a maior diferença detectada não ultrapassou os 4

segundos. Dessa forma, nota-se que o CDPO detectou os eventos no momento de sua ocorrência, sem uma grande latência. Tal característica é ratifica que o SUMO possui uma boa aplicabilidade para modelar aplicações que necessitam de uma baixa latência de resposta.

4.3.3 Conclusão

Os resultados obtidos nesse experimento demonstraram que o SUMO disponibiliza ferramentas e funções que possibilitam a modelagem e a execução de aplicações que visem monitorar a velocidade média de veículos que trafegam por uma determinada via em tempo real, realizando cálculos de maneira rápida e eficiente. Além disso, os testes realizados entre o simulador e o *CDPO* mostraram que é possível utilizar os dados da simulação para realizar o processamento em plataformas externas ao SUMO, e que o resultado desse processamento tem uma alta acurácia e baixa latência de resposta, uma característica essencial para aplicações de mobilidade urbana.

5 Conclusão

O acelerado crescimento da população urbana torna cada vez mais necessário o desenvolvimento de soluções eficientes para lidar com os problemas oriundos desse fenômeno. Nesse contexto, as cidades inteligentes buscam proporcionar alternativas tecnológicas e sustentáveis para proverem melhores serviços e para melhorar a qualidade de vida de seus habitantes.

Nesse sentido, a mobilidade urbana constitui um dos pilares das cidades inteligentes,visto que há um crescente número de pessoas que precisam se locomover diariamente, e problemas como congestionamentos, acidentes, atrasos de ônibus, assim como outros imprevistos, afetam negativamente questões como pontualidade, segurança, humor e custos. Assim, o conceito de *smart mobility* visa mensurar a capacidade que uma cidade tem de proporcionar aos cidadãos facilidades para uso de transportes públicos, veículos menos poluentes e mais segurança no trânsito.

Sabendo disso, é necessário que os gestores busquem encontrar soluções que auxiliem no gerenciamento do fluxo de trânsito, tornando-o mais dinâmico e seguro. No entanto, desenvolver aplicações para um contexto de cidades inteligentes não é uma tarefa fácil, pois o custo e tempo do planejamento e teste dessas aplicações pode ser muito alto, se tornando, portanto, inviável de ser executado em um ambiente real. Por conta disso, são usados os simuladores, que são uma espécie de *software* que permitem o planejamento, modelação e execução de cenários controlados em um ambiente virtual, de modo que tempo e custo sejam poupados, pois antes da implementação real de uma aplicação, ela pode ser executada e testada quantas vezes se desejar.

Nessa perspectiva, o SUMO (*Simulation of Urban Mobility*) é um software de código aberto cuja finalidade é permitir o planejamento, execução e testes de aplicações voltadas para a área de mobilidade urbana. Esse simulador possui diversos elementos e funções que possibilitam a criação de cenários que envolvam fluxos de trânsitos, transporte público, monitoramento de veículos e velocidade, dentre outros. Esses cenários reproduzidos através do SUMO são fiéis à realidade, poupando a necessidade de testes em ambientes reais, flexibilizando as possibilidades de desenvolvimento e avaliação prévia.

Para atingir o objetivo principal desse trabalho, que foi investigar o uso do SUMO para o desenvolvimento de aplicações de mobilidade, foram criados dois cenários controlados a partir do SUMO, para a avaliação de aplicações de mobilidade no contexto de cidades inteligentes. O primeiro cenário teve a finalidade de executar um ambiente com a presença de transporte público, para detectar a chegada e o atraso dos ônibus às paradas. Do mesmo modo, o segundo cenário buscou demonstrar as etapas necessárias para a criação de um

ambiente em que se desejava monitorar a velocidade dos veículos ao trafegarem por trechos específicos.

Esses dois cenários são exemplos de como o SUMO pode ser utilizado para inferir diversas situações de interesse, como engarrafamentos, excessos de velocidade, acidentes e atrasos. Desse modo, essas aplicações podem servir como base para que desenvolvedores e gestores avaliem seu uso como ferramenta de melhora do fluxo de trânsito, uma vez que são flexíveis em relação ao número de veículos trafegantes, cenário físico que estão inseridos e demais particularidades que podem ser encontradas nessa área.

Após análise dos resultados, foi verificado que o simulador disponibilizou elementos necessários para a execução correta de ambos os cenários, isto é, as ferramentas e funções disponibilizados por ele foram suficientes para a se criar aplicações de mobilidade consistentes, com as características e atributos que seriam encontrados em um cenário real. Desse modo, ratifica-se que o SUMO é aplicável para a finalidade proposta, podendo auxiliar o desenvolvimento de aplicações de mobilidade para cidades inteligentes, tendo o potencial de poupar o uso de recursos financeiros e tempo no planejamento de soluções para ambientes reais.

Por fim, neste TCC foi também implementado a integração do SUMO com uma plataforma externa, o CDPO. Através desta integração foi possível avaliar o CDPO com relação a algumas métricas como latência e acurácia. Novamente, os resultados obtidos após a execução da simulação foram satisfatórios, pois o *middleware* teve resultados próximos aos resultados gerados pela simulação.

Em trabalhos futuros, é possível realizar a implementação de novos cenários de simulação, de modo a testar novas formas de utilização do SUMO em um contexto de cidades inteligentes, sendo possível, também, testar outras características do simulador, como desempenho e escalabilidade.

Referências

ALVAREZ, D. A. C. O automovel nas cidades e o planejamento deteriorado. *InterEspaço: Revista de Geografia e Interdisciplinaridade*, v. 2, n. 7, p. 45–60, jul. 2017. Disponível em: <http://periodicoseletronicos.ufma.br/index.php/interespaco/article/view/7364>. Citado na página 12.

BORN, M.; ADAMATTI, D.; AGUIAR, M. de; SOUZA, W. de. Gerenciamento de semáforos e dispersão de poluentes utilizando o simulador sumo: Estudo de caso do centro de rio grande/rs. In: SBC. *Anais do VI Workshop de Computação Aplicada a Gestão do Meio Ambiente e Recursos Naturais*. [S.l.], 2015. p. 47–54. Citado na página 25.

CIDADE, L. C. F. Urbanização, ambiente, risco e vulnerabilidade: em busca de uma construção interdisciplinar. *Cadernos Metrópole*, Pontifícia Universidade Católica de São Paulo, v. 15, n. 29, p. 171–191, 2013. Citado na página 10.

COUTINHO, A.; CARNEIRO, E. O.; GREVE, F. G. P. Computaçao em névoa: Conceitos, aplicaç oes e desafios. *Minicursos do XXXIV SBRC*, p. 266–315, 2016. Citado na página 21.

DAMERI, R. P. et al. Searching for smart city definition: a comprehensive proposal. *International Journal of computers & technology*, v. 11, n. 5, p. 2544–2551, 2013. Citado na página 10.

EsperTech. *Esper Documentation*. 2023. <https://www.espertech.com/esper/esper-documentation/>. Acessado em 4 de julho de 2023. Citado 3 vezes nas páginas 18, 19 e 20.

ESPOSTE, A. d. M. d.; KON, F.; COSTA, F. M.; LAGO, N. Interscity: A scalable microservice-based open source platform for smart cities. In: *Proceedings*. [S.l.: s.n.], 2017. Citado na página 12.

FERREIRA, J. S. W. Globalização e urbanização subdesenvolvida. *São Paulo em Perspectiva*, Fundação SEADE, v. 14, n. 4, p. 10–20, Oct 2000. ISSN 0102-8839. Disponível em: <https://doi.org/10.1590/S0102-88392000000400003>. Citado na página 11.

FILHO, M. F. *Internet das Coisas (Internet of Things)*. [S.l.: s.n.], 2016. ISBN 9788550601113. Citado na página 11.

GENARI, D.; COSTA, L. F. da; SAVARIS, T. P.; MACKE, J. Smart cities e o desenvolvimento sustetável: revisão e perspectivas de pesquisas futuras. *Revista de Ciências da Administração*, p. 69–85, 2018. Citado na página 11.

GIFFINGER, R.; FERTNER, C.; KRAMAR, H.; KALASEK, R.; MILANOVIć, N.; MEIJERS, E. *Smart cities - Ranking of European medium-sized cities*. [S.l.: s.n.], 2007. - p. Citado 2 vezes nas páginas 10 e 12.

GOMES, B. d. T. P. *Uma Abordagem de Middleware com Suporte à Qualidade de Contexto voltada para Aplicações de Internet das Coisas*. Tese (Doutorado). Citado na página 18.

HALL, P. Creative cities and economic development. *Urban studies*, Sage Publications Sage UK: London, England, v. 37, n. 4, p. 639–649, 2000. Citado na página 10.

JUNIOR, M. R.; GUEDES, Á. L.; MAGALHÃES, F. B.; COLCHER, S.; ENDLER, M. Introdução ao processamento de fluxo de dados: uma abordagem orientada a eventos complexos. *Sociedade Brasileira de Computação*, 2019. Citado 4 vezes nas páginas 17, 18, 19 e 20.

LOPEZ, P. A.; BEHRISCH, M.; BIEKER-WALZ, L.; ERDMANN, J.; FLÖTTERÖD, Y.-P.; HILBRICH, R.; LÜCKEN, L.; RUMMEL, J.; WAGNER, P.; WIESSNER, E. Microscopic traffic simulation using sumo. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2018. p. 2575–2582. Disponível em: <https://elib.dlr.de/127994/>. Citado 4 vezes nas páginas 25, 29, 30 e 31.

LUCKHAM, D. The power of events: An introduction to complex event processing in distributed enterprise systems. additions-wesley. *Reading*, 2001. Citado 3 vezes nas páginas 17, 18 e 19.

MALDONADO, J. C.; VITERBO, J.; DELAMARO, M.; TEDE, P. C. d. A. R.; KON, F.; ZAMBOM, E.; SILVA, T. H.; LOUREIRO, A. A.; FIGUEIREDO, C. de; MARCZAK, S. dos S. et al. Jornadas de atualização em informática 2016. *Sociedade Brasileira de Computação*, 2016. Citado 2 vezes nas páginas 11 e 12.

MANIKA, E.; ALVES, J.; WILLE, E.; FONSECA, K.; VENDRAMIN, A. Um esquema automatizado de mapeamento de mapas com importação de dados do transporte público para o sumo. In: *Anais do Simpósio Bras. de Telecomunicações e Processamento de Sinais.* [S.l.]: SBRT, 2019. Citado na página 26.

MEDEIROS, F. S. B.; COLPO, I.; SCHNEIDER, V. A.; CARVALHO, P. S. de. Internet of things. *Revista Eletrônica de Administração e Turismo-ReAT*, v. 12, n. 7, p. 1652–1674, 2018. Citado na página 11.

MOURA, D. d. M. A. Otimização do transporte público visando a integração entre zonas central e sul de porto alegre utilizando o software de simulação sumo. 2022. Citado na página 26.

NETO, J. d. A. R.; FONSECA, J. Uma solução para alerta de preço de combustível em tempo real baseada em processamento de eventos complexos. *Revista de Engenharia e Pesquisa Aplicada*, v. 5, n. 1, p. 31–39, 2020. Citado na página 17.

ÖBERG, C.; GRAHAM, G.; HENNELLY, P. Smart cities: A literature review and business network approach discussion on the management of organisations. *Imp Journal*, Emerald Publishing Limited, 2017. Citado na página 10.

PASETO, L.; MARTINEZ, M. R. M.; PRZEYBILOVICZ, E. Cidades inteligentes e indústria 4.0: a influência das tecnologias da informação e comunicação. *Revista Científica e-Locução*, v. 1, n. 17, p. 22–22, 2020. Citado na página 10.

RESENDE, P. d. T. V.; SOUSA, P. R. d. Mobilidade urbana nas grandes cidades brasileiras: um estudo sobre os impactos do congestionamento. *Fundação Dom Cabral, Caderno de ideias CI*, v. 910, 2009. Citado na página 12.

RODRIGUES, I.; SILVA, F.; COUTINHO, L.; MARQUES, J.; TELES, A. S. Detectando padrões de sociabilidade de seres humanos através do processamento de eventos complexos. *Sociedade Brasileira de Computação*, 2020. Citado 3 vezes nas páginas 18, 19 e 20.

RONDON, L.; ROCHA-FILHO, G.; VILLAS, L. Protocolos eficientes para comunicação v2v em redes veiculares de dados nomeados. In: *Anais Estendidos do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2021. p. 177–184. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc_estendido/article/view/17169>. Citado na página 25.

SCHENFELD, M. C. *Fog e edge computing: uma arquitetura híbrida em um ambiente de internet das coisas*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2017. Citado na página 21.

SILVA, C. A. D. *Uma plataforma extensível para a transformação de fluxo de dados heterogêneos em cidades inteligentes*. Dissertação (Mestrado) — Brasil, 2015. Citado na página 12.

UN-Habitat. *World Cities Report 2022: The Value of Sustainable Urbanization*. Nairobi, Kenya: UN-Habitat, 2022. Citado na página 10.

YOU, E. F. *Edge and Fog Computing: Practical Uses*. 2023. Acessado em: 31 de outubro de 2023. Disponível em: <https://iot.electronicsforu.com/content/tech-trends/edge-and-fog-computing-practical-uses/>. Citado na página 22.

ZANELLA, A.; BUI, N.; CASTELLANI, A.; VANGELISTA, L.; ZORZI, M. Internet of things for smart cities. *IEEE Internet of Things journal*, Ieee, v. 1, n. 1, p. 22–32, 2014. Citado na página 16.