UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

BRIAN MARQUES VIANA

NOKIX STL - UM MIDDLEWARE PARA O GERENCIAMENTO DE DADOS SEMAFÓRICOS

QUIXADÁ

2018

BRIAN MARQUES VIANA

NOKIX STL - UM MIDDLEWARE PARA O GERENCIAMENTO DE DADOS SEMAFÓRICOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Paulo Armando Cavalcante Aguilar

Coorientador: Prof. Dr. Marcio Espíndola Freire Maia

QUIXADÁ

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

V667n Viana, Brian Marques.
NOKIX STL : Um middleware para o gerenciamento de dados semafóricos / Brian Marques Viana. – 2018.
43 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2018.
Orientação: Prof. Dr. Paulo Armando Cavalcante Aguilar.
Coorientação: Prof. Dr. Marcio Espíndola Freire Maia.

1. Middleware. 2. Internet das coisas. 3. Semáforo. 4. Mobilidade urbana. I. Título.

CDD 005.1

BRIAN MARQUES VIANA

NOKIX STL - UM MIDDLEWARE PARA O GERENCIAMENTO DE DADOS SEMAFÓRICOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: ___ /___ /____

BANCA EXAMINADORA

Prof. Dr. Paulo Armando Cavalcante Aguilar (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Marcio Espíndola Freire Maia (Coorientador)
Universidade Federal do Ceará (UFC)

Profa. Dra. Rossana Maria de Castro Andrade
Universidade Federal do Ceará (UFC)

Aos meus pais, Maria do Socorro e Raimundo Filho.

AGRADECIMENTOS

A minha família, em especial aos meus pais por me apoiarem e investirem no meu futuro, sempre acreditando em mim.

A minha namorada Alquimara, por sempre estar ao meu lado, nas horas boas e ruins, sempre me dando apoio emocional e motivacional.

A meus professores orientadores Prof. Dr. Paulo Armando Cavalcante Aguilar e Prof. Dr. Marcio Espíndola Freire Maia, por toda a ajuda, contribuição e orientação desse trabalho.

Aos demais professores do curso, em especial ao Prof. David Sena, Diana Braga, Paulyne Matthews, Carla Ilane, Carlos Diego e Aníbal Cavalcante por me ensinarem e ajudar a crescer como profissional.

A meus amigos e colegas de graduação, em particular Davi Cedraz, Samuel Alves, Ernades Junior, Leo Jaimesson, Gleydson Rodrigues e Rodrigo Barbosa, por me auxiliarem em vários momentos difíceis.

A meus amigos, Felipe Sampaio, Emerson Viera e Iago Sousa pela ajuda dentro e fora da universidade.

“O que você sabe não tem valor; o valor está no que você faz com o que sabe.”

(Bruce Lee)

RESUMO

Este trabalha fala sobre o desenvolvimento do middleware NOKIX STL, que é um middleware para o gerenciamento de dados semafóricos. O intuído o middleware é facilitar o desenvolvimento de aplicações que necessitem dos dados semafóricos, tais como os aplicativos de navegação por gps, onde o desenvolver poderá consultar o NOKIX STL para verificar o tempo de um determinado semáforo, e assim calcular a velocidade media que o veiculo deve seguir para sempre pegar os semáforos verdes, este seria um exemplo de aplicação. No decorrer do trabalho será apresentados dos os passos utilizados para o desenvolvimento do middleware assim como seus requisitos e arquitetura.

Palavras-chave: Semáforos. *Middleware*. *IoT*. Internet das Coisas.

ABSTRACT

This work talks about the development of the NOKIX STL middleware, which is a middleware for the management of traffic data. The intuition of the middleware is to facilitate the development of applications that require the traffic signal data, such as GPS navigation applications, where the development can consult the NOKIX STL to check the time of a certain traffic light, and thus calculate the average speed that the vehicle must follow to always pick up the green traffic lights, this would be an example of application. In the course of the work will be presented the steps used to develop the middleware as well as its requirements and architecture.

Keywords: *Traffic lights*. *Middleware*, *Internet of Things*.

SUMÁRIO

1 INTRODUÇÃO

Um dos maiores problema enfrentados no dia a dia de quem vive nas cidades é a falta de uma boa mobilidade urbana, provocados por diversos fatores, entre eles está o rápido crescimento populacional urbano. Segundo a ONU (Organização das Nações Unidas), mais de 54% das pessoas vivem na zona urbana, esse número deve aumentar para 70% até 2050, ONU (2013). No Brasil, de acordo com o censo demográfico de 2012, 80% dos brasileiros vivem nas cidades. Em consequência do crescimento urbano, temos a precária mobilidade urbana, ocasionada pelo enorme número de veículos causando gigantescos congestionamentos no trânsito. De acordo com o Departamento Nacional de Trânsito (DENATRAN), o Brasil terminou o ano de 2017 com uma frota de veículos total de 97.091.956. Em 2010 havia 64.817.974 de veículos. Portanto, teve um crescimento de aproximadamente 50% em sete anos. Em consequência disso, temos diversos outros problemas, tais como, a poluição, tanto do ar quanto sonora, o desperdício de tempo das pessoas paradas no trânsito, seja por conta de acidentes, vias mal dimensionadas ou por meio de semáforos mal regulados, entres outros. De forma geral temos uma redução na qualidade de vida das pessoas.

Tornar as cidades "inteligentes" tem sido uma estratégia muito utilizada para amenizar os problemas ocasionados pelo rápido crescimento populacional urbano. Uma cidade inteligente ou do inglês *Smart City*, como é conhecida, é uma cidade que utiliza da Tecnologia da Informação e Comunicação (TICs) para auxiliar nas diversas atividades do dia a dia da sociedade. Nesse sentido, as TIC's se fundem com as infraestruturas tradicionais de forma coordenadas e integradas usando tecnologias digitais como sensores, câmeras de vídeos, sistemas embarcados, entre outras. Esse conceito de cidades inteligentes aplica o paradigma de Internet das Coisas (*IoT*) para seu desenvolvimento. A *IoT* é uma rede de objetos, tais como, eletrodomésticos, veículos, casas, entre outros, que possuem algum tipo de tecnologia embarcada e sensores capazes de coletar e transmitir dados Zanella *et al.* (2014). Em sua essência, a Internet das Coisas é apenas um ambiente que concentra na nuvem, informações de diversos dispositivos do cotidiano (*smartphone*, veículos, semáforos entre outros) e aplicações (redes sociais, sistemas de produção, sistemas de controle de tráfego entre outros).

Uma cidade inteligente, possui diversas soluções para resolver o problema da mobilidade urbana precária, como estacionamentos conectados, de forma que o usuário possa saber se há vagas em um determinado estacionamento, poder reservar ou até mesmo fazer o pagamento do estacionamento por meio de um aplicativo. Outra solução seria totens conectados em pontos

de ônibus, de forma que qualquer pessoa possa interagir para ver todos os itinerários dos ônibus. Também existem aplicativos de caronas compartilhadas, aplicativos de navegação por exemplo o Waze. Além disso temos os sistemas de semáforos inteligentes (do inglês *Smart Traffic Light* - STL). Um STL é um sistema que permite a automação de um semáforo de forma inteligente, em que o ciclo do semáforo passará de estático para dinâmico conforme o fluxo de veículos ou pedestres. Por ser um sistema para internet das coisas, pode ser desenvolvido de diversas formas, entre elas está a abordagem de Hawi Roxanne e Okeyo (2017), que usou uma rede de sensores sem fio para coletar dados em tempo real e a lógica *Fuzzy* na concepção do algoritmo que define o tempo do semáforo de forma dinâmica com base no fluxo de veículos. Já Kanungo *et al.* (2014), utilizou o processamento de vídeo para capturar imagens do tráfego e assim calcular sua densidade para definir o tempo dinamicamente para cada um dos semáforos de um cruzamento.

A concepção dos STL envolve a utilização de diversas tecnologias, tanto de *hardware*, como sensores, atuadores e sistemas embarcados, quanto de *software*, como lógica *Fuzzy*, linguagens de programação (Java, C, Python entre outras) e protocolos de comunicação (HTTP, MQTT entre outros). Por conta da inúmeras tecnologias existentes para a implementação de um STL, podem existir algumas dificuldades, pois não há um padrão de desenvolvimento ainda definido, portanto, para criar um dispositivo ou uma aplicação STL, cada desenvolvedor poderá implementá-lo da maneira que lhe for mais conveniente. Em consequência disso, existe um mercado com diversos dispositivos semelhantes, mas sem interoperabilidade.

Ao realizar um estudo domínio dos semáforos inteligentes, especialmente nos citados acima, percebeu-se diferentes formas de implementação, seja por questões de linguagens de programação, *hardware* utilizados ou protocolos de comunicação. Uma forma de solucionar tal problema, seria a utilização de um *middleware*. Um *middleware* é um *software* capaz de fazer a mediação entre a camada de aplicação (por exemplo um sistema de monitoramento e controle dos semáforos) e a camada de percepção dos dados (por exemplo dados coletados de um semáforo e do fluxo de veículos), abstraindo os diferentes tipos de protocolos de comunicação, plataformas e até mesmo sistemas operacionais. Dessa forma, para desenvolver uma aplicação que monitore e/ou controle esse semáforos sem o intermédio de um *middleware*, seria necessário o desenvolvimento de um ambiente completo de coleta, armazenamento e processamento de dados para cada modelo STL, o que dificultaria o desenvolvimento e a interoperabilidade de aplicações.

Observando a problemática da padronização dos sistemas de semáforos inteligentes,

este trabalho, propõe o desenvolvimento do *middleware* NOKIX STL para o controle, armazenamento e gestão de dados dos semáforos. Desta forma, qualquer desenvolvedor que necessitar dos dados semafóricos para sua aplicação, só precisará se comunicar com o *middleware* para obtê-los. O *middleware* ainda poderá auxiliará na concepção de novos modelos STL's, pois o mesmo poderá agir como um padronizador dos dados necessários para um sistema deste modelo. É importante ressaltar que este trabalho abstrai os meios responsáveis pela coleta das informações dos semáforos e dos fluxo de veículos.

2 TRABALHOS RELACIONADOS

Esta Seção, detalha os trabalhos relacionados acerca dos semáforos e *middlewares*, apresentando um breve resumo e os pontos de cada trabalho que terão influências na construção do *middleware* NOKIX STL.

2.1 Semáforos

O Departamento Nacional de Trânsito (DENATRAN) juntamente com o Conselho Nacional de Trânsito (CONTRAN) apresentam um conjunto de manuais sobre a sinalização no trânsito, o Volume V do Manual Brasileiro de Sinalização de Trânsito, refere-se a sinalização semafórica e trata das considerações gerais e os critérios para a implantação dos semáforos, as características gerais para o controle e a programação dos mesmos, CONTRAN e DENATRAN (2016).

A automação semafórica pode ser feita de diversas forma com tecnologias de *hardware* ou *software* diferentes. Hawi Roxanne e Okeyo (2017) apresenta uma abordagem híbrida para o desenvolvimento de um STL, em seu projeto tem como base dois componentes. O primeiro é uma rede de sensores sem fio (do inglês *Wireless Sensor Network* - WSN) para a coleta de dados do tráfego. O segundo trata-se de um conjunto de controladores, responsáveis pela tomada de decisão do tempo que o semáforo ficará verde. Tais controladores foram implementados utilizando a lógica *Fuzzy*.

Já Kanungo *et al.* (2014), utilizou o processamento de vídeo para capturar imagens do tráfego, a captura da foto era realizada para o próximo semáforo ficar verde, essa imagem é pré-processada para preto e branco e depois passa por um algoritmo que realiza uma análise para calcular a densidade do tráfego e definir o tempo dinamicamente para o sinal. Desse modo, ele garante que todas as vias, pelo menos uma vez por ciclo, ficar com o sinal verde.

Em Villalonga *et al.* (2011), foi proposto a construção de um sistema inteligente de trânsito (SIT), para automação de semáforos por meio de um mecanismos de *hardware*. A estrutura proposta para o controlador foi dividida em duas categorias, os cartões de luzes, instalado nos veículos e é responsável por ativar o sinal para a interseção que se o veículo se encontra e o cartão de controle, responsável pelo controle de todas as interseções e a comunicação com os níveis hierárquicos acima de sua camada, como sistemas de consumos de dados semafóricos. O *hardware* usado, foi uma placa baseada em um microcontrolador microchip de 8 bit com um

firmware usando PIC C + CCS.

O estado da arte desta seção foi importante para a compreensão dos conceitos chaves relacionados aos semáforos tais como a determinação das principais entradas e saídas de dados, assim como os protocolos de comunicação e as principais estratégias usadas nesse tipo de sistema.

A tabela 1, apresenta a seguir, mostra os 3 trabalhos e suas diferenças em implementação.

Tabela 1

Trabalhos	**Coleta de dados**	**Controlador**
Hawi Roxanne e Okeyo (2017)	Rede de sensores sem fio	Lógica Fuzzy
Kanungo, Sharma e Singla (2014)	Câmera de vídeo e processamento de imagens	C++ e MATLAB
Sarmiento (2011)	Não realiza coleta de dados	Hardware

Fonte: Elaborada pelo autor

2.2 Middleware IoT

No trabalho de Razzaque *et al.* (2016), foi levantado um conjunto de características para sistemas IoT e um conjunto de requisitos para a construção de um *middleware* IoT. Os requisitos de *middleware* foram dividindo-os em dois subconjuntos, são eles, requisitos de serviços (compostos por requisitos funcionais e não funcionais) e requisitos de arquitetura. Esses requisitos são diretamente influenciados pelas características dos sistemas IoT. Eles são importantes, pois definem os atributos mínimos que um *middleware* IoT deve apresentar. Com base nesse conjunto de requisitos, foi realizado um levantamento das principais abordagens para *middleware* IoT. Para cada abordagem, foram descritos os principais *middleware* utilizados, fazendo uma comparação entre os pontos forte e fracos, tanto dos *middleware* quanto da abordagem, em relação aos requisitos levantados. Mostrando com resultado que cada *middleware* apresentado suporta de forma total ou parcial no mínimos dois dos requisitos levantados, porém nenhum *middleware* citado no trabalho suporta todos os requisitos.

Já o artigo de Cruz *et al.* (2018), descreve os principais padrões de comunicação que são utilizados atualmente em um ambiente IoT. Assim como no trabalho de Razzaque *et al.* (2016), e foi realizado um levantamento de requisitos e características da internet das coisas, porém com o foco em plataformas IoT. Uma plataforma IoT é um pacote de software que integra

dispositivos, redes e aplicativos, de forma a esconder a complexidade da implementação do usuário, fornecendo um ecossistema onde as “coisas” são construídas. Ele utiliza os requisitos para dividir um conjunto com as principais plataformas IoT em três categorias: plataformas de gerenciamento de dispositivos, plataformas de desenvolvimento de aplicativos e plataformas de habilitação de aplicativos, que é o foco do seu trabalho. Essa categoria de plataforma também é conhecida como *middleware* IoT. Em cada *middleware* do conjunto de plataformas selecionadas, foi realizada uma análise para definir quais os principais padrões de comunicação utilizado e se tal *middleware* é de código aberto ou é um Paas (*Platform as a Service*, do português plataforma como serviço). Com base no que foi analisado, o autor propõe um modelo referência para um *middleware* IoT, onde ele define os principais requisitos para a concepção de um *middleware*.

Para a concepção do *middleware* NOKIX STL, foi tomado como base, os requisitos e características levantados pelos dois trabalhos apresentados nesta seção. Já a escolha do padrão de comunicação teve como base a pesquisas apresentadas no trabalho Cruz *et al.* (2018).

Além disso, realizou-se um breve estudo sobre o middleware LoCCAM,Maia *et al.* (2013), afim de entender algumas questões arquiteturas como a organização dos modudos do mesmo.

As Tabelas 2, 3 e 4, aprestadas a seguir mostram todas as informações que foram necessarias para realizar o estudo dos middleware.

Tabela 2

Trabalhos	**Levantamento de tecnologias**	**Levantamento de Middleware**
Razzaque et al. (2016)	Não	Faz um levantamento geral de middleware
Cruz et al. (2018)	Sim	Realiza o levantamento apenas de middlewares de código aberto e plataformas de middleware

Fonte: Elaborada pelo autor

Tabela 3

Trabalhos	**Levantamento de requisitos funcionais**	**Levantamento de requisitos não funcionais**
Razzaque et al. (2016)	Sim	Sim
Cruz et al. (2018)	Sim	Sim

Fonte: Elaborada pelo autor

Tabela 4

Trabalhos	**Levantamento de arquiteturas**	**Características de IoT**
Razzaque et al. (2016)	Sim	Mostra as principais características da IoT
Cruz et al. (2018)	Não	Mostra apenas algumas características

Fonte: Elaborada pelo autor

3 FUNDAMENTAÇÃO TEÓRICA

Esta seção, resume os principais conceitos chaves utilizados neste trabalho, são eles, Semáforos, Internet das Coisas e *Middleware* IoT.

3.1 Semáforos

Segundo Krauss (2014), um semáforo é um controlador de passagem para veículos e pedestres nos cruzamentos de uma via que funciona por meio de uma indicação luminosa, permitindo ou não o deslocamento dos veículos ou pedestre, por meio da comutação das luzes do dispositivo controlador.

No Brasil existem órgãos responsáveis por regulamentar esse equipamento de trânsito, entre eles estão o CONTRAN e o DENATRAN. Esses órgãos define um conjunto de requisitos básicos que um semáforo deve ter. Esse requisitos serão explicados nos tópicos seguintes.

3.1.1 Tipos de Controle

De acordo com o DENATRAN, existem dois tipos de controles para sinalização semafórica, sendo eles:

a) **Controle em Tempo Fixo:** Esse modelo trabalha com o tempo definido estaticamente no semáforo, geralmente em *hardware*, com base no plano semafórico obtido por meio de estudos do fluxo de veículos obtido por meio de contagens volumétricas e outros levantamentos de campo.

b) **Controle Atuado pelo Tráfego:** Esse padrão tem a alteração do tempo por meio de algum tipo de atuador e se divide em dois modelos, sendo eles:

- **Semi atuado:** Este tipo de controle precisa da interação humana para realizar a alteração no tempo verde definido a cada um dos sinais, podendo ser por exemplo, acionado por meio de uma botoeiras, que interrompe o ciclo da via principal de veículos dando prioridade a travessia de pedestre.
- **Totalmente atuados:** O controle totalmente atuado usa de monitores e detectores do fluxo de veículos e assim ajusta o tempo do semáforo verde, esses equipamentos pode ser sensores de tráfego ou câmeras de vídeo, dependendo da estratégia usada para monitorar o tráfego. Esse tipo de

controle permite o ajuste em tempo real da duração do ciclo. Para que o semáforo seja dinâmico, deve-se definir um limite mínimo e máximo de tempo em que a luzes devem ficar acesas em um ciclo.

3.1.2 Tipos de Estratégia

O DENATRAN ainda define duas estratégias para a implantação dos controles semafórico citados anteriormente, sendo elas:

a) **Controle isolado:** Nesta estratégia, cada grupo de semáforos se comportam de forma independentes uns dos outros, ou seja, não existe coordenação semafórica. Sendo assim, o semáforo leva em conta apenas o fluxo local, podendo ser por meio de um histórico de um estudo ou a demanda atual do tráfego.

b) **Controle em rede:** Esta estratégia, visa o aumento de desempenho do tráfego por meio de uma rede aberta ou rede fechada, sendo dividida em duas maneira, são elas:

- **Rede aberta:** Visa melhorar a circulação do tráfego ao longo de uma via/corredor, estabelecendo um tempo que permita um fluxo contínuo entre os cruzamentos adjacentes.
- **Rede fechada:** Visa melhorar o desempenho do tráfego de forma geral em um região, de forma que os cruzamentos devem ser agrupados e coordenados para permitir um fluxo contínuo.

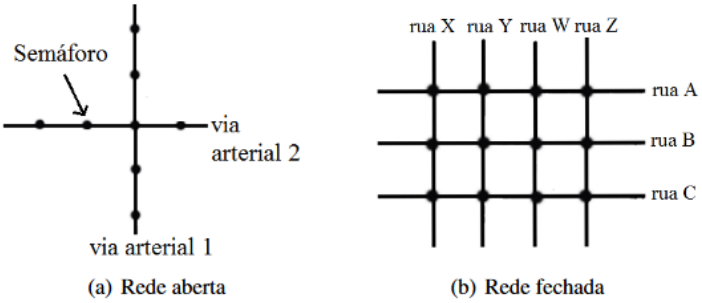
Na Figura 1, mostrada a seguir, (a) tem-se a representação de uma rede aberta com semáforos ao longo de uma via arterial. Já em (b) tem-se a representação de uma rede fechada em que nos pontos que as vias arteriais se cruzam tem-se semáforos

3.1.3 Modo de Operação

Ainda conforme o DENATRAN, o modo de operação de um semáforo pode ser realizado de duas formas, sendo elas:

a) **Controle local (descentralizado):** Cada cruzamento semafórico tem um dispositivo controlador que recebe o programa para seu controle, qualquer alteração no ciclo do semáforo deve ser feito diretamente no controlador, utilizando o recurso disponíveis nele.

Figura 1 – Exemplos de estrategia de controle de rede



Fonte: CONTRAN e DENATRAN (2016)

b) **Controle centralizado:** Nesse modo de operação, os controladores eletrônicos de tráfego são ligados a um computador central que gerencia a operação conjunta dos equipamentos. O controle centralizado é utilizado para agilizar a operação do sistema de interseções semaforizadas.

3.1.4 Ciclo semafórico

É obrigatório que todos os semáforos de um cruzamento tenham passado uma vez pela luz verde, para a realização de um ciclo. Por conta disso, é necessário definir um tempo mínimo e máximo que a luz verde pode esta ativa em um ciclo.

3.2 Internet das Coisas (IoT)

A Internet das Coisas é um novo paradigma definido pelo conceito de ambientes inteligentes que abre espaço para o desenvolvimento de um grande número de aplicações com grande impacto na vida cotidiana futura, Giusto *et al.* (2010). A ideia básica é a presença na vida cotidiana, de um número variado de objetos que serão equipados com micro controladores, transmissores e receptores para comunicação na rede e um conjunto de protocolos que permitirão a comunicação entre si, e também a comunicação com os usuários, de forma cooperativa para alcançar seus objetivos, tornando-os assim parte integrante da internet Atzori *et al.* (2010).

Esse conceito, visa tornar a Internet mais imersiva e abrangente, possibilitando o fácil acesso e interação com uma grande variedade de objetos do dia-a-dia, tais como eletrodomésticos, câmeras de vigilância, sensores de monitoramento, atuadores, veículos, semáforos de trânsito entre outros. Além disso, a rede de “coisas” produzirá uma grande variedade de

dados possibilitando a implantação de diversas aplicações que forneceram serviços à comunidade Zanella *et al.* (2014).

Figura 2 – Principais domínios de aplicações IoT

Fonte: Al-Fuqaha *et al.* (2015)

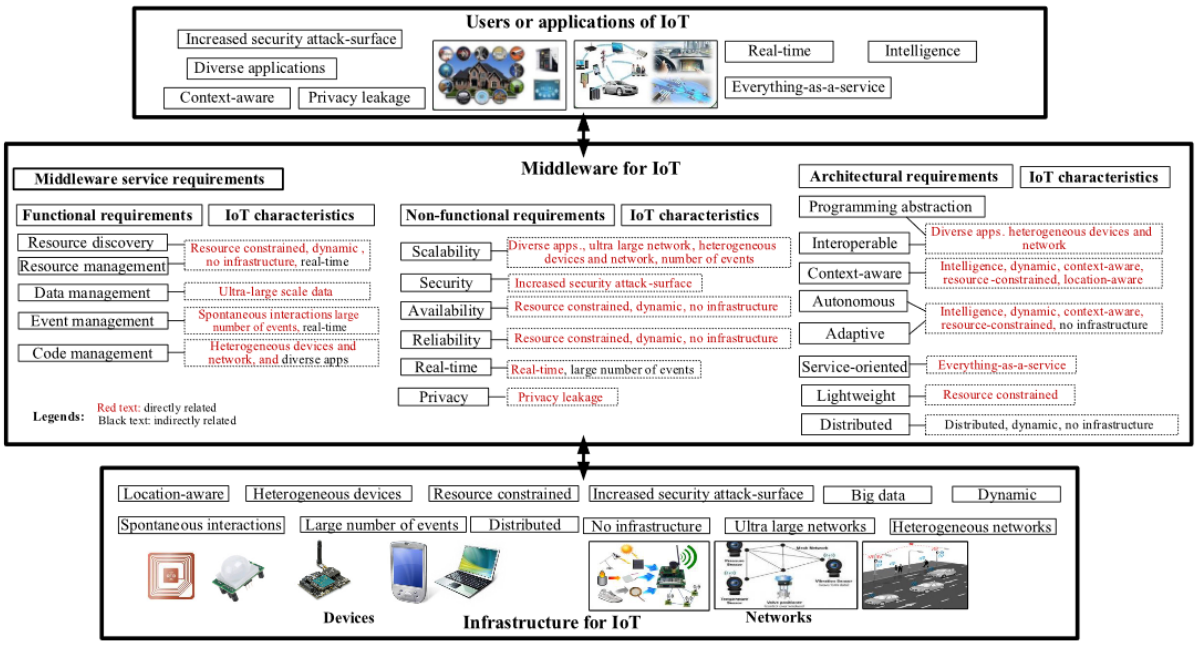
Nesse contexto, automação residencial/industrial, gerenciamento inteligente de energia, assistência médica móvel, logística, sistemas inteligentes de transporte (SIT), entre outros são apenas alguns exemplos de aplicações existentes nos diferentes domínios da IoT.

3.3 *Middleware* para IoT

Um *middleware* fornece uma camada de software entre aplicativos, o sistema operacional e as camadas de comunicação da rede, o que facilita e coordena alguns aspectos do processamento cooperativo. Normalmente, ele abstrai as complexidades do sistema, permitindo que o desenvolvedor de aplicativos concentre todos os esforços na tarefa a ser resolvida sem a distração das preocupações ortogonais no nível do sistema Neely *et al.* (2006).

Para a concepção de um *middleware* para IoT, existem alguns requisitos fundamentais, são eles, os requisitos de serviços do *middleware* e os requisitos da arquitetura que o sistema deve suportar.

Figura 3 – Relação entre os requisito de *middleware*, as aplicações e a infra estrutura



Fonte: Razzaque *et al.* (2016)

3.3.1 Requisitos de Serviço do middleware

Os requisitos de um *middleware* podem ser divididos em duas categorias, requisitos funcionais (RF), que trata das funcionalidade do sistema, ou seja, o que o *middleware* deve fazer, quais tarefas ele pode executar, já os requisitos não funcionais (RNF) levam em consideração a qualidade que os serviço deve ter, e por isso tão são conhecidos como atributos de qualidade Razzaque *et al.* (2016), Cruz *et al.* (2018). É importante ressaltar que requisitos levantados nesse

tópico, não pertencem a um domínio específico da IoT, mas sim a qualquer *middleware* projetado para IoT, por esse fato que não foram levantados os requisito de negócio nesse ponto.

Os requisitos funcionais serão mostrados a seguir.

a) **Descoberta de recursos:** Uma das principais características da IoT é a diversidade de recurso, esses recursos podem ser dispositivos conectados à rede (por exemplo sensores e atuadores) ou os serviços ofertados por esses dispositivos, além do mais esse recursos deve estar na rede de forma dinâmica, pois seria inviável a interferência humana para descobrir e adicionar cada um deles a rede, ou seja, é de grande importância que toda a descoberta desse recurso seja automatizada. A descoberta de recursos é o processo usado por um dispositivo para procurar os recursos desejados, onde toda a rede é sondada para serviços.

b) **Gerenciamento de recursos:** Em um ambiente como a IoT, onde os recursos têm muitas restrições por conta do tamanho dos dispositivos, o que limita seu processamento, memória e capacidade de comunicação, é necessário que haja um serviço para o gerenciamento dos mesmo. Para isso essencial que o uso desses recursos seja monitorado e alocado de maneira justa e os conflitos devem ser resolvidos.

c) **Gerenciamento de dados:** Uma parte de grande importância em IoT são os dados, os mesmos pode ser informações de infraestrutura da rede ou dados capturados pela rede de interesse para os aplicativos. Por isso um *middleware* precisa fornecer um serviço de gerenciamento de dados aos aplicativos, atuando desde o processo de aquisição até o processamento e armazenamento desses dados.

d) **Gerenciamento de eventos:** Aplicativos IoT pode gerar um grande número de eventos. O gerenciamento de eventos é uma extensão do gerenciamento de dados. Ele deve fornecer análise em tempo real de dados em alta velocidade para que os aplicativos de recebimento de dados sejam orientados por informações precisas e em tempo real.

e) **Gerenciamento de código:** A atualização de código no ambiente IoT e impraticável se todos os dispositivos fossem atualizados pessoalmente. Sendo assim, o *middleware* devem facilitar as operações de atualização de código, pois elas possuem uma conexão com os dispositivos.

Os principais requisitos não funcionais são os seguintes:

a) **Escalabilidade:** Um *middleware* IoT precisa ser escalável, pois a quantidade de "coisas" conectadas a rede pode crescer exponencialmente. Com isso, o *middleware* deve fornecer uma qualidade de serviços mesmo com a adição de novos dispositivos ou aplicações a rede. A melhora da escalabilidade pode influenciar positivamente na ocultação da complexidade do *hardware* ou da lógica da implantação de serviços subjacentes.

b) **Tempo real ou pontualidade:** Grande parte das aplicações IoT precisa dos dados em tempo real, portanto, e essencial que o *middleware* forneça esse serviço. Na ciência da computação, o termo tempo real significa que o usuário mal percebe o atraso entre o envio de dados e a quantidade de tempo que o computador leva para receber e processar os dados Cruz *et al.* (2018). Informações ou serviços atrasados em alguns aplicativos podem tornar o sistema inútil e até perigoso.

c) **Confiabilidade:** É confiabilidade de um *middleware* está relacionada com o fato que o mesmo deve permanecer operacional durante toda a sua vida, mesmo com a presença de falhas. Isso implica diretamente no nível de confiabilidade do sistema, ou seja, todos os componentes e serviços precisam ser confiáveis para que um *middleware* tenha a confiabilidade geral Razzaque *et al.* (2016).

d) **Disponibilidade:** Mesmo com falha em algum lugar no sistema, um *middleware* IoT deve estar sempre disponível. O tempo de recuperação e a frequência de falhas deve ser o mínimo possível para atingir a disponibilidade idealizada. Portanto, o requisito de confiabilidade deve funcionar lado a lado com o requisito de disponibilidade para garantir a maior tolerância a falhas que o sistema exige.

e) **Segurança e privacidade:** Assim como em qualquer sistema, a segurança é algo fundamental, especialmente no ambiente IoT esse aspecto é ainda mais crítico, pois o comprometimento da segurança permite a realização de diversos tipos de ataques, como DoS (*Denial of Service*), ou a até mesmo a divulgação de dados dos usuários, por exemplo a localização em tempo real comprometendo assim a privacidade do usuário. No *middleware* IoT, a segurança precisa ser considerada em todos os blocos funcionais e não funcionais, incluindo o aplicativo no nível do usuário.

f) **Facilidade de implantação, manutenção e uso:** Como um *middleware* IoT

é gerenciado por seus usuários, que em geral, não tem conhecimento técnico, portanto, o *middleware* deve fornecer uma procedimento descomplicado quanto a sua instalação, configuração, uso e manutenção.

3.3.2 Requisitos de Arquitetura de middleware

Assim como qualquer software, um *middleware* precisa de uma arquitetura bem definida tem um bom empenho em seu papel. Questões arquiteturais envolvem a organização e estrutura geral de controle, protocolos de comunicação, sincronização, alocação de funcionalidade a componentes e seleção de alternativas de projeto. Porém os requisitos que serão mostrados neste tópico, tratam do dilema abstrações de programação e outras preocupações no nível da implementação, visando auxiliar os desenvolvedores de aplicações.

Os principais requisitos de arquitetura serão mostrados a seguir:

a) **Abstração da programação:** Um requisito essencial para qualquer *middleware* é o fornecimento de uma API (*Application Programming Interface*) para que os desenvolvedores de aplicativos ou serviços, pois as interfaces de programação de alto nível precisa ser isoladas do desenvolvimento de aplicações ou serviços fornecidos pelo ambiente heterogêneo da IoT. Na construção de uma API deve-se levar em conta o nível de abstração, que trata de como o desenvolvedor vai visualizar os dados fornecidos pelo sistema, o paradigma de programação é outro ponto crucial, ele é responsável por lidar com o modelo de desenvolvimento ou programação do aplicativo ou serviços (por exemplo *Publisher/Subscriber*, programação orientada a objetos (POO), entre outros), e por fim temos o tipo de interface, que define a forma como os dados serão transferidos (alguns exemplos seriam o JSON, XML e até mesmo SQL).

b) **Interoperabilidade:** Um *middleware* deve ser compatível com diferentes tipos de dispositivos, tecnologias e aplicativos, para que o desenvolvedor tenha o mínimo de esforço possível. Um maneira de alcançar a interoperabilidade e fornece a comunicação por diferente tipos de protocolos, como HTTP e MQTT. Além disso, o uso de uma API permite que o sistema exponha funcionalidades e dados aos aplicativos sem que o desenvolvedor precise conhece a implementação do *middleware*.

c) **Adaptabilidade:** Em um ambiente IoT onde a mudanças constantemente, pois a

rede de objetos é dinâmica e a demanda de requisições de um aplicativo pode variar frequentemente, um *middleware* precisa ser adaptativo para garantir a qualidade de seus serviços em todas as variações do ambiente.

d) **Autônomo:** A rede IoT é dinâmica, em consequência disso, novos dispositivos podem ser adicionados a rede ou até mesmo realocados. O *middleware* deve ser autônomo para auxiliar aos dispositivos a se comunicar uns com os outros.

e) **Distribuído:** Os dispositivos, aplicações e usuários provavelmente estarão devidos geograficamente, portanto o *middleware* deve fornecer uma solução que suporte tais condições. Além disso o *middleware* deve suportar funcionalidades que estão distribuídas na rede da IoT.

3.3.3 Abordagens de Arquitetura de Middleware

Além dos requisitos, a concepção de um *middleware* pode apresentar diversas abordagens. Essas abordagens lidam com o comportamento de *middleware* e está diretamente ligada a arquitetura do sistema.

As principais abordagens são:

a) **Baseado em eventos:** Nesse modelo, os componentes, aplicativos e qualquer outro dispositivo da rede, interagem entre si por meio de eventos. Um evento possui um tipo e um conjunto de parâmetros, que descrevem as alterações no estado do produtor. O fluxo de propagação dos evento inicia no componente de envio (produtor) e termina no componente de recebimento (consumidor). Em geral, se implementa um *middleware* com esse modelo utilizando a arquitetura *Publisher/Subscriber*, pois essa abordagem trata melhor alguns requisitos não funcionais, como confiabilidade, disponibilidade, desempenho em tempo real, escalabilidade e segurança.

b) **Orientada a serviços:** Esse paradigma visa a construção de um software em forma de serviço. Essa abordagem é baseada na Arquitetura Orientada a Serviço (SOA) e tem com suas principais características a neutralidade da tecnologia, o baixo acoplamento, a capacidade de reutilização do serviço, a capacidade de composição do serviço e a capacidade de descoberta do serviço, isso permite um grande benefício para as aplicações IoT.

c) **Orientado a banco de dados:** Nesta arquitetura cada rede de sensores é vista

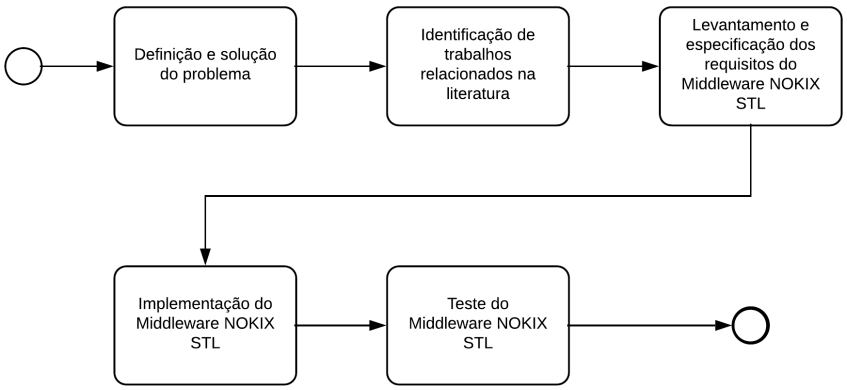
como um banco de dados relacionar, porém de forma virtual. Um aplicativo pode realizar consultas na rede (banco de dados virtual) por meio de uma linguagem semelhante ao SQL (*Structured Query Language*, ou Linguagem de Consulta Estruturada), permitindo a formulação de consultas complexas. O principal requisito dessa abordagem se encontra em uma solução de banco de dados distribuída para interoperar sistemas.

d) **Específico da aplicação:** Essa abordagem de *middleware* foca no suporte ao gerenciamento de recursos para um aplicativo específico ou domínio de aplicativos, implementando uma arquitetura que melhore o desempenho da rede ou infraestrutura baseado nos requisito de domínio do aplicativo.

4 PROCEDIMENTOS METODOLÓGICOS

Esta Seção, tem como objetivo apresentar os passos realizados para a concepção deste trabalho. O fluxograma apresentado na Figura 4 representa as etapas identificadas do processo metodológico

Figura 4 – Fluxograma de procedimentos



Fonte: Elaborado pelo autor

4.1 Definição e solução do problema

Inicialmente, foi realizado um estudo sobre mobilidade urbana com o objetivo de identificar e entender os domínios da aplicação, além de determinar as soluções atualmente aplicadas a esse domínio. Essa pesquisa mostrou um conjunto de soluções, nas quais, algumas podem ser melhoradas, um exemplo seria os semáforos inteligentes, que embora apresentem uma melhora na mobilidade urbana, ainda podem ser aprimorados. É justamente nesse ponto que a proposta deste trabalho está focada, já que se trata do desenvolvimento de um *middleware* para melhorar a gestão dos dados gerados pelos STLs, visando a melhora na qualidade dos semáforos e no desenvolvimento de aplicativos para os mesmos.

4.2 Identificação de trabalhos relacionados na literatura

Nesse passo, realizou-se um estudo para buscar, na literatura, trabalhos relacionados aos temas identificados no estudo do domínio. Utilizou-se as base do *google scholar* e *researchgate* com as seguintes strings de buscas:

- semáforos inteligentes
- internet das coisas
- iot
- middleware

Os trabalhos relacionados aos semáforos, sejam eles inteligentes ou não, foram selecionados com intuito de compreender quais são os principais requisitos para o desenvolvimento, além das tecnologias utilizadas na implementação e as diversas abordagens do mesmo. Já os artigos relacionados ao domínio de *middlewares* foram selecionados com intuito de compreender esse domínio, em especial a fase de desenvolvimento e as tecnologias utilizadas, tanto na implementação quanto nos protocolos de comunicação, além disso, os objetivos de tais artigos selecionados assemelham-se aos objetivos do presente trabalho. Todos os trabalhos selecionado foram apresentados na seção 2.

4.3 Levantamento e especificação dos requisitos do *Middleware* NOKIX STL

Um processo de software é um conjunto de atividades relacionadas que levam à produção de um produto de *software*, Sommerville (2011). A atividade inicial é no estágio de requisitos, que consiste da elicitação, análise, especificação e validação dos requisitos do sistema.

No desenvolvimento do projeto NOKIX STL, por se tratar de um projeto de software, foi necessário realizar a especificação dos requisitos que serão implementados no *middleware*, além de definir quais recursos serão usados e qual o publico alvo do sistema.

Com base nos trabalhos relacionados, foi realizado uma análise para definir quais os principais requisitos do sistema sejam eles funcionais ou não funcionais.

4.4 Implementação do *Middleware* NOKIX STL

O *Middleware* NOKIX STL foi implementado usando a linguagem *JavaScript* por meio da plataforma **Node JS**. Essa ferramenta de desenvolvimento foi selecionada para a codificação do sistema, por ser uma tecnologia que permite a produção de aplicações em tempo real, além de permitir a programação baseada em eventos, Pereira (2013).

Para a persistência dos dados foi utilizado **MongoDB**. Por se tratar de um banco de dados NoSQL que permite a realização de consulta com uma alta velocidade, além de ser flexível e escalável. Além disso ele ainda fornece maneiras de analisar e acessar os dados de

forma simplificada, Pereira (2013).

Como protocolos de comunicação, foram selecionados **HTTP** e **MQTT**. **HTTP** é utilizado no principalmente no desenvolvimento de aplicações para *WEB*. Já o **MQTT** e geralmente utilizado no desenvolvimento de aplicações para *IoT*, em especial, aplicações com arquitetura *Publisher/Subscriber*.

Para a troca de mensagens entre o *middleware* e as aplicações, utilizou-se **JSON**. Tal padrão foi escolhido por ser um formato de texto, que independente da linguagem de programação, sendo assim qualquer sistema deve ser capaz de interpretar ou gerar uma mensagem em **JSON**.

Como editor de código, utilizou-se o ***Visual Studio Code***, por ser um editor simples e de familiaridade e preferência do desenvolvedor.

Usou-se o **Git** e o **GitHub** o controle de versão e hospedagem de código-fonte.

4.5 Teste do *Middleware* NOKIX STL

Para realizar o teste do *middleware*, foi desenvolvido um simulador com a plataforma **Arduino** para simular um semáforo inteligente, onde o simulador comunica-se com o *middleware* por meio do protocolo **MQTT**, simulando um semáforo real.

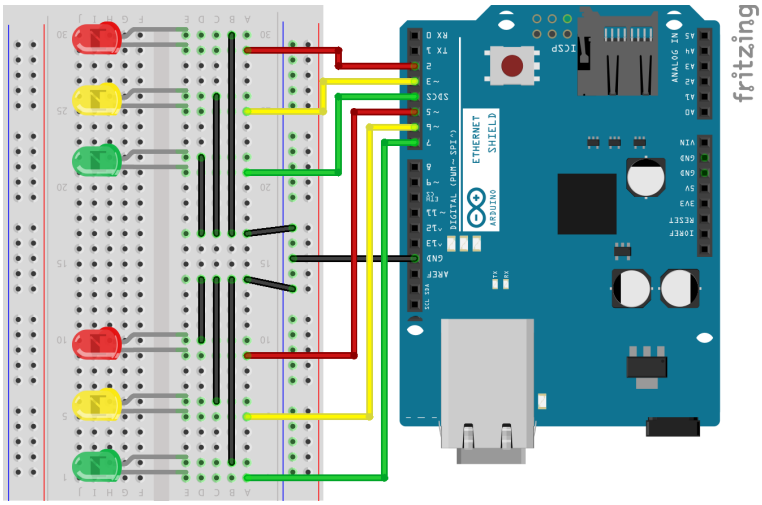
4.5.1 Experimento

Foi utilizado a plataforma ***Arduino***, para implementar o código de um semáforo, baseado no manual brasileiro de sinalização semafórica. Além disso, foi construído em uma *protoboard* o protótipo dos semáforos que foram utilizados para a realização dos testes.

O projeto e o código desenvolvido pode ser encontrado no repositório do projeto no ***GitHub***[1].

[1] <https://github.com/brianmviana/semaforo-inteligente-arduino-mqtt>

Figura 5 – Modelo do protótipo de semáforo



Fonte: Elaborado pelo autor

5 O *MIDDLEWARE* NOKIX STL

O NOKIX STL, é um *middleware* para o gerenciamento de semáforo, que visa facilitar a comunicação entre os sistemas semafóricos e as aplicações que necessitem desses dados. Por meio do NOKIX STL é possível coletar as informações geradas pelos semáforos de forma simples, apenas consultando uma *API Rest*.

Esta seção contém o detalhamento dos requisitos e funcionalidades do *middleware* assim como as especificações da comunicação do *middleware*, tanto da API de tópicos do MQTT quanto da *API Rest*, além da descrição de sua arquitetura.

5.1 Requisitos e Funcionalidades

Para realizar o levantamento e elicitação dos requisitos, foi realizado um estudo na literatura, sobre sistemas similares e quais eram os principais requisitos de cada um deles. O estudo resultou em uma base de requisitos apresentados na Seção 3. Além dos requisitos encontrados para o contexto de um *middleware*, existem os requisitos necessários para o contexto da aplicação. Tais requisitos estão listados a seguir:

- Requisitos Funcionais
  - Cadastrar um Semáforo
  - Cadastrar um Cruzamento
  - Cadastrar uma Via
  - Adicionar um semáforo a um cruzamento
  - Adicionar um cruzamento a uma via
  - Listar informações de um semáforo
  - Listar informações de um cruzamento
  - Listar informações da via
  - Listar todos os semáforos de um cruzamento
  - Autenticar um semáforo
  - Autenticar uma aplicação
  - Permitir comunicação pelo protocolo MQTT
  - Permitir comunicação pelo protocolo HTTP
- Requisitos Não Funcionais
  - Facilidade de implantação, manutenção e uso

- Interoperabilidade
- Abstração da programação
- Tempo real ou pontualidade
- Adaptável
- Específico da aplicação
- Seguraça

5.2 API de Tópicos

O protocolo MQTT funciona sobre a arquitetura *Publish / Subscribe*. Essa arquitetura trabalha sobre um conjunto de tópicos para realizar a comunicação entre dois ou n dispositivos. Esse protocolo será utilizado para realizar a comunicação entre os semáforos e o *middleware*, pois o mesmo foi projetado para dispositivos restritos e redes de baixa largura de banda, alta latência ou não confiáveis. Os princípios do protocolo são minimizar a largura de banda da rede e os requisitos de recursos do dispositivo, ao mesmo tempo em que tentam garantir a confiabilidade e um certo grau de garantia de entrega. Além disso, este protocolo é bastante usado na comunicação de dispositivos *IoT*.

Sendo assim, foi necessário definir uma *API* de tópicos no *middleware* para que seja possível realizar a comunicação com os semáforos. Os tópicos existentes no *middleware* estão listados abaixo.

- **/semaforo** : Esse tópico permite que um semáforo cadastrado no *middleware*, publique suas informações.

Como o *middleware* ainda esta em sua versão inicial, foi desenvolvido apenas esse tópico para que os semáforos enviem suas informações para o *middleware*. Novos tópicos serão adicionados em versões futuras. Tais como:

- **/semaforo/auth** : Usado para autenticar um semáforo no middleware.

5.3 API Rest

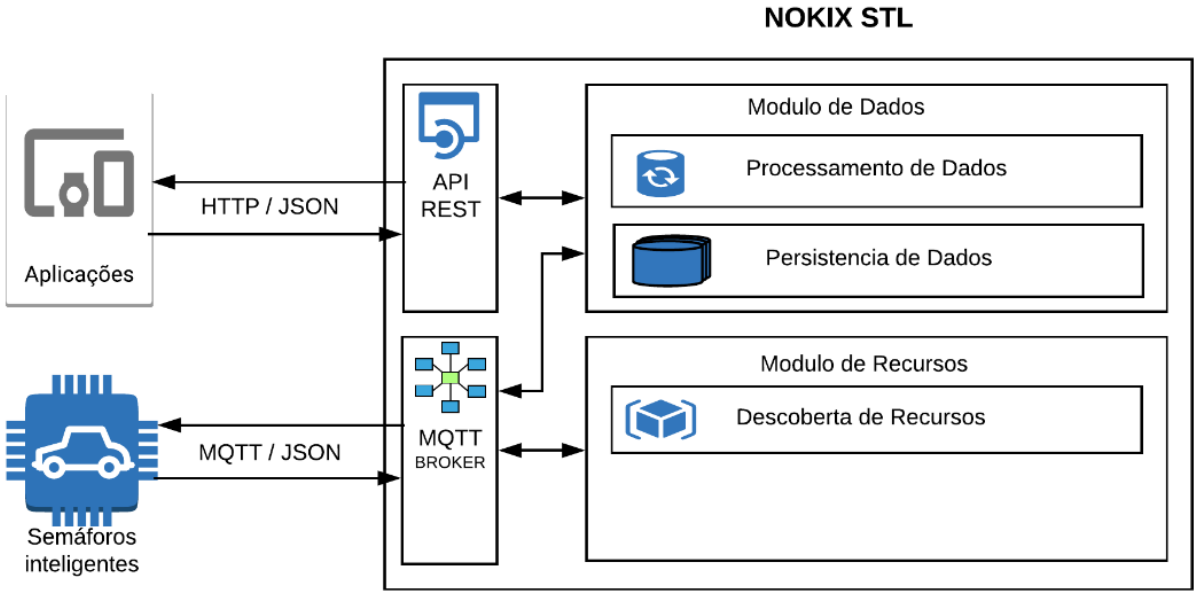
Para que outros sistemas ou aplicativos possam se comunicar com os semáforos através do *middleware*, foi desenvolvido uma *API Rest*. O *Rest* é um estilo de arquitetura que define um conjunto de restrições e propriedades baseados em HTTP. O *Rest* trabalha sobre os métodos do HTTP para realizar ações sobre os recurso, também conhecidos como URI.

É comum que um serviço REST suporte múltiplos formatos para representar seus recursos, tais como XML, JSON e HTML. Porém a versão inicial do *middleware* ira trabalhar sobre o formato JSON. Esse formato foi selecionado pois é o padrão mais adotado nas aplicações recentes.

5.4 Arquitetura do NOKIX STL

Ainda na etapa de requisitos, foi realizado um estudo sobre as diversas arquiteturas aplicadas no desenvolvimento de *middlewares IoT*. Com base nessas arquiteturas e no domínio da aplicação, foi concebido o desenho arquitetônico da primeira versão do *middleware* NOKIX STL. A figura 6 mostra a arquitetura na qual o *middleware* foi desenvolvido:

Figura 6 – Arquitetura do *Middleware* NOKIX STL



Fonte: Elaborado pelo autor

A figura 6 mostra todos os módulos implementados no middleware. O modulo de Dados é responsável persistir as informações em um banco de dados e tratar essa informações quando requisitadas por algum sistema. Já o modulo de recurso é responsável por saber quando um semáforo está conectado ao middleware e procurar novos semáforos na rede. Ainda é possível perceber que existem dois módulos de entrada e saída de dados, esse dois módulos utilização os seguintes padrões arquiteturais:

- Arquitetura *Publish/Subscribe*

- Arquitetura Rest

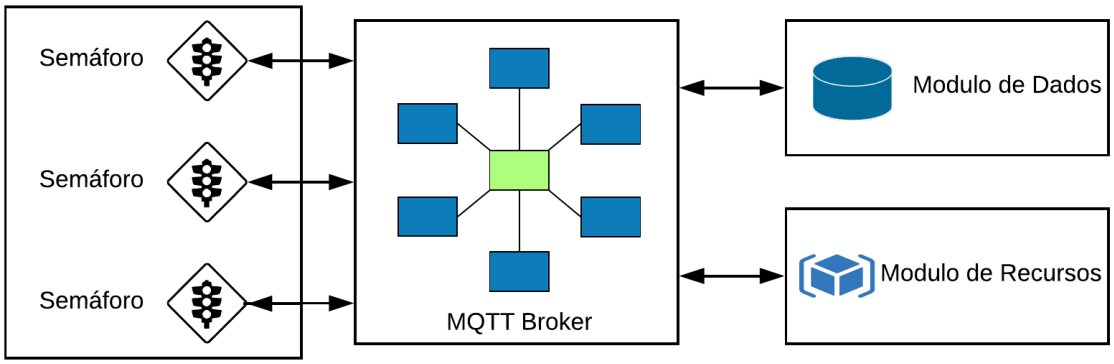
As arquiteturas base utilizadas para o devolvimento assim como o motivo de sua escolha serão detalhadas nas próximas seções.

5.4.1 Arquitetura Publish/Subscribe

Existem diversos protocolos de comunicação para Internet da Coisas, entre eles esta o MQTT, que atualmente, é o mais utilizado para comunicação entre sistemas/dispositivos IoT. Por conta disso, ficou decidido de implementar esse protocolo no *middleware* para realizar a comunicação com os semáforos inteligentes. Esse protocolo trabalha sobre o padrão arquitetural *publish/subscribe*, logo esse modelo de arquitetura tornou-se um requisito do *middleware*.

O padrão *Publish/Subscribe* cria uma entidade central, muitas vezes chamada de Broker, que recebe mensagens dos *publishers* e as reencaminha para *subscribers* cadastrados. Um dos pontos fortes desse protocolo é o desacoplamento entre o *publish* e o *subscribe*. Os dois elementos têm conhecimento zero um do outro, permitindo uma independência muito grande entre os elementos que populam as mensagens no *broker* e os elementos que irão consumir essa informação.

Figura 7 – Arquitetura *Publish/Subscribe*



Fonte: Elaborado pelo autor

A Figura 7, mostra como está implantada a arquitetura no *middleware* NOKIX STL. Nesse caso, os semáforos estão fazendo o papel de publicadores, enviado assim suas informações para o *Broker* implementado no *middleware*. O *Broker*, ao receber os dados de um semáforo, envia para o módulo de dados, que é responsável por persistir os dados e envia-los para qualquer

aplicativo que os requisite pela *API Rest*. Além disso, o *Broker* avisa o módulo de recursos que um dispositivo está conectado. Até o momento, nenhum subscrito foi implantado, pois não havia necessidade.

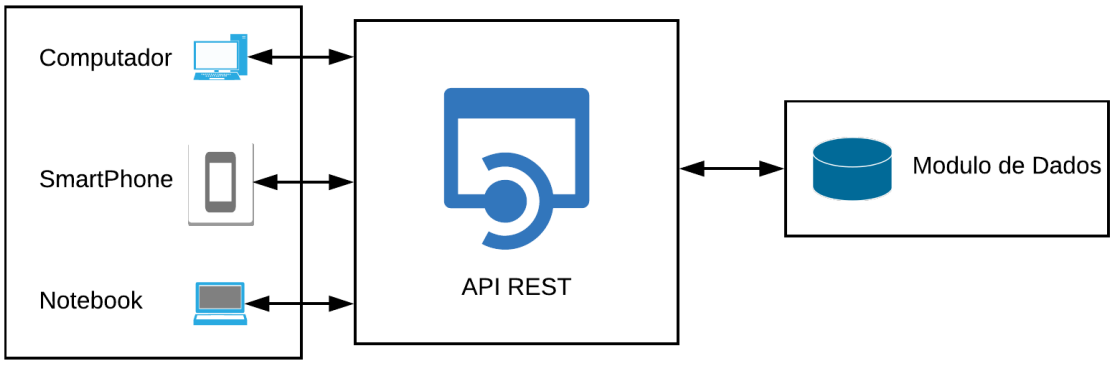
5.4.2 Arquitetura Rest

O Rest, é um modelo arquitetural que define um conjunto de restrições e propriedades baseadas no protocolo HTTP. Sistemas que seguem esse padrão de arquitetura, também conhecidos como sistemas *RESTful*, fornecem a interoperabilidade com outros sistemas.

Os *web services* compatíveis com REST permitem que os sistemas solicitantes acessem e manipulem representações textuais de recursos da *Web* usando um conjunto uniforme e predefinido de operações sem estado.

Esse modelo ignora os detalhes da implementação de componente e a sintaxe de protocolo com o objetivo de focar nos papéis dos componentes, nas restrições sobre sua interação com outros componentes e na sua interpretação de elementos de dados significantes.

Figura 8 – Arquitetura *REST*



Fonte: Elaborado pelo autor

Na Figura 8, é possível visualizar como esse modelo de arquitetura foi implantada no *middleware* NOKIX STL. Essa representação arquitetural, mostra os dispositivos acessando a API do middleware para consultar as informações presentes no banco de dados.

6 TESTES E RESULTADOS

Essa seção apresenta os testes realizados e seus resultados na avaliação do *middleware*, afim de validar o mesmo. Os testes executados foram os seguintes:

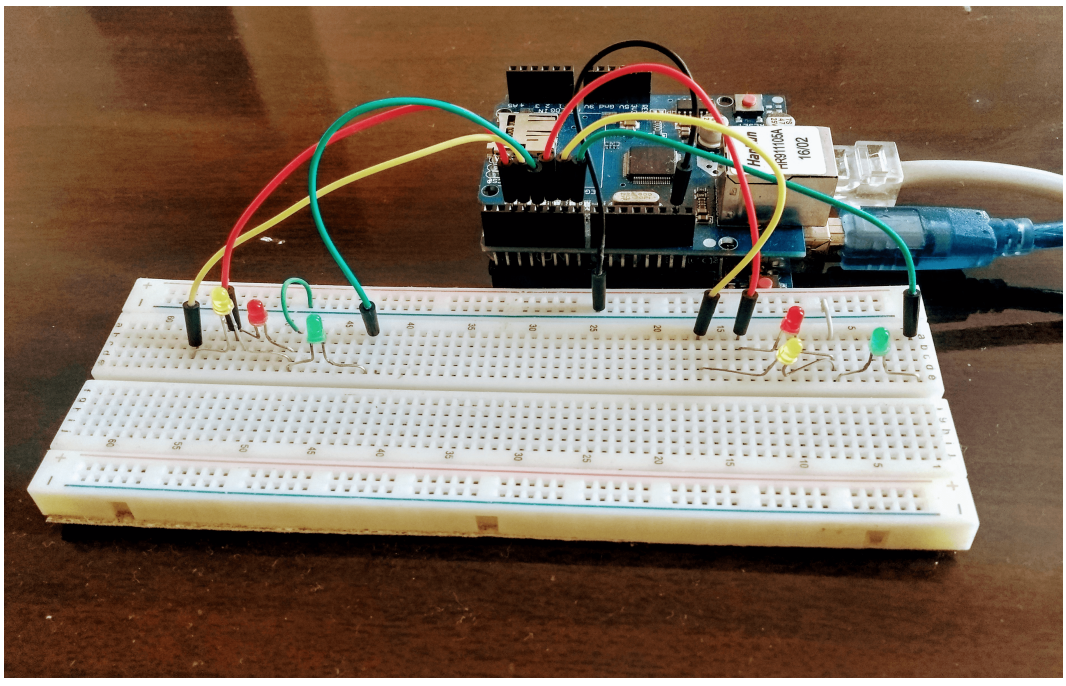
- Teste com simulador e aplicação
- Teste de carga com Jmeter

O objetivo desses primeiro teste era validar o middleware quanto a suas funcionalidades e usabilidade. Já o segundo teste tem como objetivo validar a capacidade de carga em acesso que o middleware suporta. Tais teste são detalhados na seções seguintes.

6.1 Teste com simulador e aplicação

Por falta de recursos e equipamentos semafóricos para realizar os testes do *middleware* em um ambiente real, foi desenvolvido um controlador semafórico por meio da plataforma Arduíno, com intuito de simular um controlador semafórico real, como mostra a figura 9. O código implementado no Arduíno, tem como base os requisitos obrigatórios para a concepção e implantação de um semáforo descrito pelo Volume V do Manual Brasileiro de Sinalização de Trânsito. O modelo do protótipo desenvolvido pode ser visto na figura 4 situado no capitulo 4.

Figura 9 – Controlador semafórico construído com Arduíno

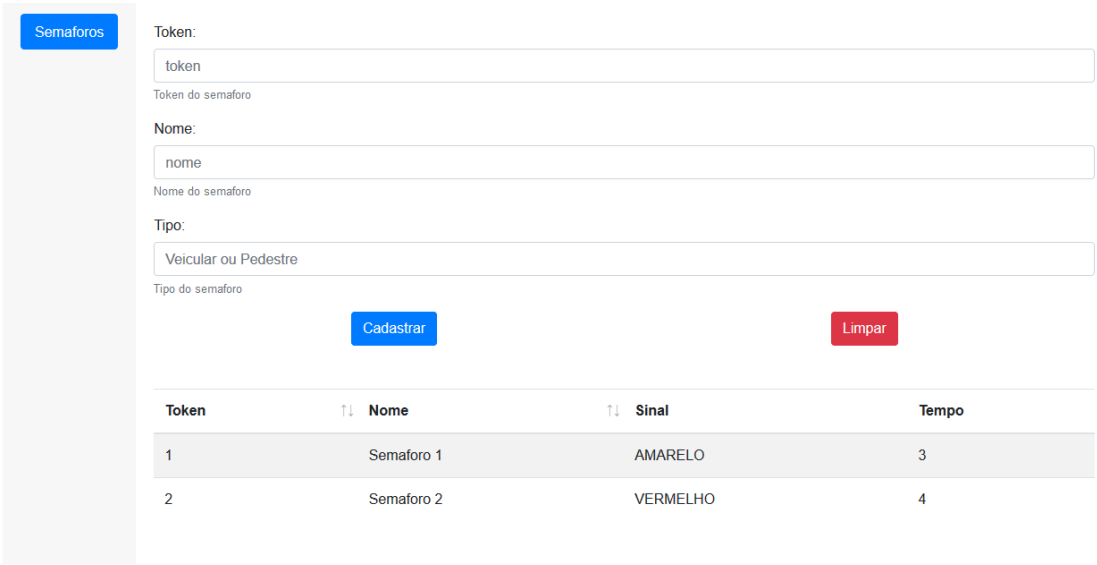


Fonte: Elaborado pelo autor

Para complementar os testes com o semáforo simulado no Arduíno, foi desenvolvido uma aplicação *web* simples feita em ***VueJS*** (*Framework JavaScript*), que lista todos os cadastrados assim como suas informações. Além disso, a aplicação estava responsável por cadastrar os semáforos.

A aplicação *web* requisita para o *middleware* por meio da *API Rest* as informações sobre os semáforos a cada 1 segundo, e o middleware responde com o tempo satisfatório máximo de um segundo de atraso do controlador semafórico implementado com o Arduíno. Além disso, os testes realizados mostram que o *middleware* estava funcionando como os requisitos esperados, seja pela facilidade de uso em consumir os dados gerados pelos semáforos ou pelos tempo de resposta que o mesmo tinha quando era requisitado por uma aplicação.

Figura 10 – Aplicação Frontend feita em VueJS



Fonte: Elaborado pelo autor

6.2 Teste de carga com Jmeter

Além dos teste com o simulador e aplicação, realizei um teste de carga, com a ferramenta **Jmeter**, para saber quantas requisições o *middleware* suporta por segundo. O teste foi executado 10 vezes, variando apenas no número de usuários simultâneos em cada um deles. Cada um dos usuários, realizavam 5 requisições com intervalo de 1 segundo entre elas. O numero de usuário por teste foram de:

- 100 para o 1º teste

- 200 para o 2º teste
- 300 para o 3º teste
- 400 para o 4º teste
- 500 para o 5º teste
- 1000 para o 6º, 7º, 8º, 9º e 10º teste

Todos os teste foram executados em um notebook, o que pode influenciar no resultados dos teste, pois as configurações de um notebook em geral são mais fracas de as configurações de um servidor. As configurações da maquina na qual os testes foram executados, são as seguintes:

- Processador Intel core i5 de 4ª geração de 1.60GHz
- Memoria RAM de 12 GBs DDR3
- Sistema Operacional *Windows* 10 Pro x64

O resultado dos testes pode ser visto nas tabelas 5 e 6, logo apos as tabelas, os resultados serão discutidos.

Tabela 5

TESTES	**1**	**2**	**3**	**4**	**5**
Amostra	500	1000	1500	2000	2500
Media de requisições	66	273	353	425	762
Tempo min	7	6	5	4	22
Tempo max	148	472	1062	1886	2596
% de erro	0%	0%	0%	0%	2.16%

Fonte: Elaborada pelo autor

Tabela 6

TESTES	**6**	**7**	**8**	**9**	**10**
Amostra	5000	5000	5000	5000	5000
Media de requisições	1142	1106	1111	1256	1099
Tempo min	6	3	5	9	5
Tempo max	2663	2568	2848	2749	2659
% de erro	18.3%	13.52%	17.8%	21.06%	16.2%

Fonte: Elaborada pelo autor

Como pode ser observado nas tabelas, o sistema de comporta normalmente para um total de até 2000 requisições no tempo total de 5 segundos. Em média são respondidas 425 requisições a cada segundo. No teste de número 5, com total de 500 usuários simultâneos, o sistema não respondeu corretamente apenas 54 requisições de um total de 2500, o que não está

fora do resultado esperado.

Já entre os testes 6 e 10, o *middleware* apresentou em alguns momentos, mais falhas que o esperado na hora de responder as requisições corretamente. O único teste que o sistema atingiu o esperado foi o teste de número 7. Porém nos testes 6 e 8 o sistema operacional da máquina apresentou uma perca de desempenho, pois o nível de processamento chegou ao máximo.

No geral, o middleware se comportou como o esperado para o total de requisições, porem algumas funcionalidades precisão ser aprimoradas, para que o middleware possa responder com uma velocidade maior.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho surgiu como uma proposta para o desenvolvimento de uma aplicação que visa facilitar o gerenciamento dos dados gerados pelos semáforos inteligentes. Afim de criar uma maneira simples e fácil de disponibilizar tais dados, com o propósito de facilitar o desenvolvimento de novas aplicações para melhorar a mobilidade urbana.

O *middleware* NOKIX STL, foi desenvolvido utilizando a linguagem *JavaScript*, sobre a plataforma *NodeJS*, pois o mesmo tinha o requisito de ser assíncrono. Após a finalização do *middleware* está finalizado, foram realizados testes afim de validar seus requisitos.

Para trabalhos futuros, é proposto aplicar melhorias na *middleware*, tanto nas suas funcionalidades, seja, ampliando o número de recursos disponíveis na API REST, quanto na sua arquitetura, adicionando novos módulos, como um gerenciador de serviços e um serviço de autenticação de usuários. Além disso, novos protocolos de mensagens poderiam ser agregados ao middleware, como XML. Além dessas melhorias, no middleware, poderão ser realizadas melhorias na aplicação web, como adição um mapa para a visualização dos semáforos.

REFERÊNCIAS

AL-FUQAHA, A.; GUIZANI, M.; MOHAMMADI, M.; ALEDHARI, M.; AYYASH, M. Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE Communications Surveys & Tutorials**, IEEE, v. 17, n. 4, p. 2347–2376, 2015.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer networks**, Elsevier, v. 54, n. 15, p. 2787–2805, 2010.

CONTRAN; DENATRAN. **Manual Brasileiro de Sinalização de Trânsito, Volume V - Sinalização Semafórica**. [S.l.], 2016. Disponível em: http://www.denatran.gov.br/images/Educacao/Publicacoes/Manual_VOL_V_(2).pdf. Acesso em: 3.27.2018.

CRUZ, M. A. da; RODRIGUES, J. J. P.; AL-MUHTADI, J.; KOROTAEV, V. V.; ALBUQUERQUE, V. H. C. de. A reference model for internet of things middleware. **IEEE Internet of Things Journal**, IEEE, v. 5, n. 2, p. 871–883, 2018.

GIUSTO, D.; IERA, A.; MORABITO, G.; ATZORI, L. **The Internet of Things**. [S.l.]: Springer, 2010.

HAWI ROXANNE E OKEYO, G. e. K. M. Controle inteligente de semáforos usando lógica difusa e rede de sensores sem fio. In: **Conferência de Computação, 2017**. [S.l.]: IEEE, 2017. p. 450–460.

KANUNGO, A.; SHARMA, A.; SINGLA, C. Smart traffic lights switching and traffic density calculation using video processing. In: IEEE. **Engineering and computational sciences (RAECS), 2014 recent advances in**. [S.l.], 2014. p. 1–6.

KRAUSS, M. **Automação de sistema semafórico**. Dissertação (Trabalho de Conclusão de Curso (Graduação)) — Universidade Tecnológica Federal do Paraná, Curitiba, 2014.

MAIA, M. E. F.; FONTELES, A. S.; NETO, B. J. A.; GADELHA, R.; VIANA, W.; ANDRADE, R. M. C. Loccam - loosely coupled context acquisition middleware. **Symposium on Applied Computing, Proceedings of the 28th Annual ACM Symposium on Applied Computing**, p. 534–541, 2013.

NEELY, S.; DOBSON, S.; NIXON, P. Adaptive middleware for autonomic systems. In: SPRINGER. **Annales des télécommunications**. [S.l.], 2006. v. 61, n. 9-10, p. 1099–1118.

ONU. **ONU: mais de 70% da população mundial viverá em cidades até 2050**. [S.l.], 2013. Disponível em: https://nacoesunidas.org/onu-mais-de-70-da-populacao-mundial-vivera-em-cidades-ate-2050/. Acesso em: 3.27.2018.

PEREIRA, C. R. **Aplicacoes web real-time com Node-js**. [S.l.]: Casa do Código, 2013. 141 p. ISBN 9788566250145.

RAZZAQUE, M. A.; MILOJEVIC-JEVRIC, M.; PALADE, A.; CLARKE, S. Middleware for internet of things: a survey. **IEEE Internet of Things Journal**, IEEE, v. 3, n. 1, p. 70–95, 2016.

SOMMERVILLE, I. **Software engineering**. Boston: Pearson, 2011. ISBN 9780137035151.

VILLALONGA, A. R.; MORALES, E. O.; SARMIENTO, A. C. Controlador inteligente de transporte. **Revista Ingeniería Electrónica, Automática y Comunicaciones**, v. 31, n. 3, p. 1–7, 2011. ISSN 1815-5928. Disponível em: http://rielac.cujae.edu.cu/index.php/rieac/article/view/63.

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.