UNIVERSIDADE FEDERAL DO MARANHÃO
Centro de Ciências Exatas e Tecnologia
Programa de Pós-Graduação em Engenharia Elétrica

Rayanne Silva de Oliveira

UMA ABORDAGEM BASEADA EM ENGENHARIA ORIENTADA A MODELOS PARA APOIAR O DESENVOLVIMENTO DA WEB OF THINGS

São Luís - MA
2021

Rayanne Silva de Oliveira

UMA ABORDAGEM BASEADA EM ENGENHARIA ORIENTADA A MODELOS PARA APOIAR O DESENVOLVIMENTO DA WEB OF THINGS

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica, área de concentração Ciência da Computação, ao Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Federal do Maranhão.

Orientador: Prof. Dr. Denivaldo Cicero Pavão Lopes

São Luís - MA

2021

Rayanne Silva de Oliveira

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica, área de concentração Ciência da Computação, ao Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Federal do Maranhão.

São Luís - MA, 31 de outubro de 2021:

Prof. Dr. Denivaldo Cicero Pavão Lopes
Orientador
Universidade Federal do Maranhão

Prof. Dr. José Francisco da Silva e Silva
Examinador Interno
Universidade Federal do Maranhão

Profa. Dra. Daniela Barreiro Claro
Examinador Externo
Universidade Federal da Bahia

São Luís - MA
2021

Dedicatória

Dedico este trabalho a minha família

Agradecimentos

Agradeço a Deus, por sua infinita bondade e por ter me mantido firme durante este projeto de pesquisa com forças para chegar até o final.

A todos os meus familiares. Em especial a minha avó Aristea Lopes, minha mãe Eline Lopes, meus irmãos Rayssa Oliveira e Edmilton Filho, e aos meus tios Ednalda Lopes e Mauro Lopes.

Ao meu orientador Dr. Denivaldo Cicero Pavão Lopes por todo conhecimento oferecido, pela grande paciência, pelo desafio e pelo incentivo.

Aos amigos do LESERC(Labortório de Engenharia de Software e Redes de Computadores) Lady Débora Ferreira da Silva, Éder Matheus Silveira Felix, Matheus Ribeiro Ferreira, Raimundo de Araújo Lima e Pedro Cutrim dos Santos pelo companherismo.

Aos meus amigos de longa data que me compreenderam e me incentivaram no meu processo de aprendizagem.

A Universidade Federal do Maranhão e ao Programa de Pós-Graduação em Engenharia da Elétrica, pela oportunidade de realização do curso.

A FAPEMA pelo auxílio financeiro durante o mestrado.

E a todos que direto ou indiretamente contribuíram para realização deste trabalho.

"Feliz aquele que transfere o que sabe e aprende o que ensina"

(Cora Coralina)

Resumo

Web of Things (WoT) é uma das tecnologias mais emergentes, pois fornece uma solução para suportar interoperabilidade e heterogeneidade de aplicações que podem ser encontradas nas propostas baseadas em *Internet of Things* (IoT). A fim de tornar fácil o devenvolvimento de soluções baseadas em WoT e IoT, um *framework* baseado em *Modem Driven Engineering* (MDE) é proposto. MDE suporta um processo de desenvolvimento de *software* baseado em modelos, reduzindo o tempo de desenvolvimento e erros humanos. Node-WoT é empregado para criar *Thing Description* e suportar WoT. *WebSockets* é empregado para tornar possível a comunicação entre WoT e dispositivos IoT. Uma metodolgia de utilização do *framework* é apresentado, bem como um exemplo ilustrativo.

Palavras-chave: *Web of Things*, *Model Driven Engineering*, *framework*, *Internet of Things*, Desenvolvimento de *software*.

Abstract

Web of Things (WoT) is one of the most emerging technologies because it provides a solution to support protocols interoperability and applications heterogeneity that can be found in Internet of Things (IoT) based proposals. In order to facilitate the development of WoT solutions and communication with IoT proposals, thus preventing them from becoming out of date and allowing them to benefit from WoT based solutions, a framework based on Model Driven Engineering (MDE) technologies is proposed. MDE Technologies are responsible for automating the framework's development processes, reducing development time and human errors. Node-WoT is used, which is a library responsible for creating Thing Description and supporting WoT proposals. WebSockets, which is the communication link between WoT proposals and IoT devices, is also used. The necessary methods for framework implementation are described, and finally an illustrative example.

Keywords:Web of Things; Model Driven Engineering; Framework; Internet of Things; Software Development.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
ATL	*Atlas Transformation Language*
COAP	*Constrained Application Protoco*
EMF	*Eclipse Modeling Framework*
HTTP	*Hyper Text Transfer Protocol*
IOT	*Internet of Things*
JSON	*JavaScript Object Notation*
LED	*Light Emitting Diode*
MDA	*Model Driven Architecture*
MDE	*Model Driven Engineering*
OMG	*Object Managemet Group*
PIM	*Platform-Independent Model*
PSM	*Plataform-Specific Model*
UML	*Unified Modeling Language*
URI	*Universal Resource Identifier*
URL	*Uniform Resource Locator*
WEB	*World Wide Web*
W3C	*World Wide Web Consortium*
WOT	*Web of Things*
XMI	*XML Metadata Interchange*

Sumário

1 INTRODUÇÃO

Este capítulo apresenta o contexto, a problemática, a hipótese de pesquisa, a solução proposta, os objetivos a serem atingidos, a metodologia de pesquisa adotada e uma descrição dos demais capítulos da dissertação.

1.1 Contexto

A *Internet* surgiu em laboratórios de universidades e já proporcionou uma grande revolução nos meios de comunicação. Com a *Internet of Things* (IoT) a comunicação é constituída entre os objetos por meio da *internet*, registrando outro marco na comunicabilidade (KINTSCHNER, 2017).

IoT proporcionou o aumento principalmente nos segmentos das Smart Cities. Com isso houve um conglomerado de empresas que desenvolveram produtos IoT. Os objetos produzidos por estas empresas utilizavam-se de protocolos específicos de cada um, impossibilitando que objetos diferentes pudessem se comunicar (GUINARD DOMINIQUE ; TRIFA, 2016).

A heterogenidade dos protocolos permitiu que surgisse uma nova proposta tecnológica, a *Web of Things*, que promete sanar alguns problemas encontrados na IoT. Dentre eles a eliminação de incompatibilidade entre dispositivos e protocolos diferentes promovendo interoperabilidade entre as aplicações (W3C, 2020c).

Por ser uma proposta utilizada na web, em 2016, a *Word Wide Web Consortium (W3C)* resolveu fazer a padronização da *Web of Things*. Os padrões ainda são atuais tendo como última atualização em 2021 (W3C, 2020d). A *W3C* dividiu em dois grandes grupos, *Web of Things Interest Group (IG)* (W3C, 2020d) responsável por criar um fórum para se discutir tudo que envolve o assunto da WoT em conjunção com IoT, e *Web of Things Working Group (WG)*(W3C, 2020a), responsável por padronizar tudo que *IG* construiu.

E como resultado da padronização da W3C os seguintes tópicos, *WoT Architecture* (W3C, 2020e), que descreve como a arquitetura deve ser utilizada. *WoT Thing Description* (W3C, 2020i), descreve como os dados devem ser manipulados e consumidos por meio dos protocolos. *WoT Scripting Api* (W3C, 2020g), parte opcional para facilitar o desenvolvimento de aplicações por meio de uma interface. *WoT Binding Templates* (W3C, 2020f), fornece diretrizes voltadas para vinculação de protocolo para manipular a Thing e conectar com projetos IoT e *WoT Security and Privacy* (W3C, 2020h), descreve a implementação de políticas de segurança e privacidade necessária para a interação com a Thing.

Por ser uma padronização recente, a W3C WoT poderá diminuir a implantação dos padrões da W3C em projetos WoT. Pois em sua documentação deixou claro a necessidade de referenciá-lo como trabalho em construção e que poderá ser alterado a qualquer momento (W3C, 2020e).

Para que a tecnologia IoT possa se beneficiar da padronização de Api por meio da Thing Description possibilitando maior escalabilidade de projetos e do acesso de mais de um protocolo para sua manipulação, e, ao mesmo tempo para que a tecnologia *Web of Things* possa evoluir em conjunto com atualizações da padronização da W3C utilizamos *Model Driven Engineering*. Pois, o MDE possui benefícios como portabilidade, velocidade de desenvolvimento, qualidade na implementação e facilidade de manutenção(KLEPPE ANNEKE. ; WARMER, 2003).

Portanto, utilizar abordagem MDE para implementar a padronização da *W3C*, pode proporcionar segurança ao desenvolvedor, por ter facilidade principalmente na manutenção caso haja alteração da documentação da W3C.

1.2 Problemática

A *Web of Things* vem rompendo os conceitos da *Internet of Things* diante do mercado, pois, os projetos IoT muitas das vezes utilizam apenas um protocolo para acesso aos recursos, não havendo nenhum tipo de padronização das URIs ou de modelagem de dados (GUINARD DOMINIQUE ; TRIFA, 2016).

O padrão W3C WoT fornece a utilização de APIs, flexibilidade em número de dispositivos conectados, escalabilidade por acoplar mais dispositivos em projetos IoT, interoperabilidade entre marcas, fornecedores diferentes e comunicação entre diferentes tipos de protocolos (W3C, 2020j).

Por ser um padrão novo (W3C WoT) e ainda em construção, pode impedir a implementação da proposta por desenvolvedores e consequentemente que projetos IoT não se beneficiem da WoT (W3C, 2020b).

Por meio disso, para incentivar a utilização WoT e projetos IoT, utilizaremos MDE. Pois, o MDE propõe fácil adaptação a novas propostas por meio dos modelos, automação na criação de códigos, e possivelmente a diminuição de erros humanos (SILVA, 2015).

E isso proporciona que mesmo que a documentação da W3C venha a mudar, o MDE seja fácil de se adaptar a novas propostas e a automação de códigos.

1.3 Hipótese

Empregar o MDE para apoiar o desenvolvimento na automatização de *WoT Thing Description* em conjunto com a arquitetura WoT e seus componentes para que mais de um protocolo possa manipulá-lo (HTTP, CoAP e *WebSockets*). E ao mesmo tempo fazer automatização de código para dispositivo IoT utilizando protocolo *WebSockets*. E que por fim as duas propostas possam se comunicar por meio de *WebSockets*, possibilitando que IoT tenha o benefício trazido pelo WoT e incentive a padronização de acordo com a W3C..

1.4 Motivação

A possibilidade de utilizar MDE que proporciona um alto nível de abstração, em seu desenvolvimento, manutenção e evolução de sistemas de *software* centrada em modelos (KLEPPE ANNEKE. ; WARMER, 2003) e (SILVA, 2015).

O MDE facilita a visualização do ciclo de vida do desenvolvimento de *software*, podendo adaptar novas propostas e inseri-las no ciclo novamente (KLEPPE ANNEKE. ; WARMER, 2003) e (SILVA, 2015).

O MDE possui ainda a possibilidade de automação do código podendo, se, aplicado de forma correta, diminuir os erros humanos (SILVA, 2015) e (MOHAGHEGHI, 2011). Um fácil desenvolvimento do código, que para propostas WoT com a W3C seja mais fácil de implementar e facilitar a integração com as propostas IoT.

1.5 Objetivo Geral

O objetivo geral deste trabalho consiste em propor e desenvolver um *framework* baseado em MDE para suportar WoT e dispositivo IoT de modo a apoiar a modelagem de *Thing Description* e desenvolver o WoT em conjunto com a IoT. Proporcionando interoperabilidade e empregando protocolo padrão entre eles.

1.5.1 Objetivos Específicos

A pesquisa apresenta os seguintes objetivos específicos:

- Propor um *framework* baseado em MDE para suportar o desenvolvimento de *Web of Things* em conjunto com dispositivo IoT por *WebSockets*;
- Aplicar a padronização fornecida pela W3C;
- Propor a criação de *Thing Description*;
- Sugerir a formação de código para dispositivos IoT;

- Aplicar o *framework* para integração de protocolos heterogêneos para meio da Biblioteca *Node-WoT*;
- Fazer levantamento de trabalhos relacionados ao desenvolvimento de aplicações WoT, IoT, *WebSockets* e MDE através de artigos publicados em periódicos e anais de congressos;
- Fornecer prototipagem na utilização de *framework*;
- Expor exemplos ilustrativos para validar a proposta do *framework*.

1.6 Solução Proposta

Este trabalho propõe a criação de um *framework* baseado em MDE para suportar o desenvolvimento de WoT e IoT, proporcionando interoperabilidade, empregando protocolo padrão entre eles.

1.7 Metodologia

A metodologia utilizada para o desenvolvimento dessa pesquisa pode ser vista a seguir:

1. Pesquisa bibliográfica com a finalidade de buscar informações em *sites*, teses, periódicos, dissertações, anais de congressos, jornais e revistas sobre o estado da arte e aprofundamento referente aos assuntos MDE, WoT, IoT, MDE aplicado a WoT, MDE aplicado a IoT, e *WebSockets*;
2. Estudos sobre possíveis ferramentas a serem utilizadas para construção da abordagem proposta;
    - ATL (ATL, 2020);
    - EMF (ECLIPSE, 2020a);
    - EcoreTools (ECLIPSE, 2020b);
    - Node-WoT (THINGWEB, 2020).
3. Proposta de um *framework* baseado em MDE para suportar o desenvolvimento de WoT e IoT, permitindo a interoperabilidade entre eles;
4. Proposta de uma metodologia para aplicação do *framework* proposto para suportar o desenvolvimento de WoT e IoT;

5. Proposta de abordagem para geração de *Thing Description* conforme a padronização da W3C e a utilização de *WebSockets* para fazer comunicação com dispositivo IoT utilizando abordagem MDE para o desenvolvimento da proposta;

6. Fornecer três exemplos ilustrativos para ilustrar a proposta do *framework*;

1.8 Apresentação da dissertação

Este manuscrito é composto por sete capítulos descritos como segue:

- No Capítulo 1, apresentou-se o contexto desta dissertação, sua problemática, hipótese e solução proposta, assim como seu objetivo e metodologia;
- No Capítulo 2, apresenta-se a fundamentação teórica relacionada às ferramentas e principais conceitos utilizados nesta pesquisa;
- No Capítulo 3, apresenta-se o estado da arte, bem como os trabalhos relacionados que focam em propostas da *Web of Things* e *Internet of Things*;
- No Capítulo 4, apresenta-se um *framework* para suportar o desenvolvimento de propostas WoT e IoT em conjunto com seus bem como seus metamodelos, definições de transformações e geração de códigos da *Thing Description* e dispositivo IoT a partir destes modelos;
- No Capítulo 5, apresenta-se prototipagem de utilização do *frameworks* com software utilizado e implementações de *plugins*;
- No Capítulo 6, apresentam-se exemplos ilustrativos para a validação da proposta do *framework*;
- Finalizando a dissertação, o Capítulo 7 apresenta a conclusão desta dissertação, suas limitações e trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo, conceitos são apresentados para: MDE, *Internet of Things*, *Web of Things*, comunicação de protocolo. Esses conceitossão necessários para a concepção do *framework* e posteriormente sua implementação.

2.1 Model Driven Engineering

O *Model Driven Engineering* é considerado uma abordagem que tem como maior objetivo melhorar o desenvolvimento de sistemas, de forma mais abstrata por meio de modelos que são utilizadas para construção e manutenção por todo o ciclo de desenvolvimento de *software* (SILVA, 2015) e (MOHAGHEGHI, 2011).

Os modelos proporcionam benefícios, como: flexibilidade, portabilidade, e interoperabilidade (KLEPPE ANNEKE. ; WARMER, 2003). Pois, os modelos se baseiam na abstração do sistema, permitindo análise, entendimento e manipulação de todo sistema. Isso possibilita a diminuição de erros.

O MDE também dá a possibilidade de um rápido desenvolvimento de *software* por meio da automatização de códigos ou documentação e transformação entre eles de forma automática chamada como Arquitetura Dirigida por modelos (MDA)(MOHAGHEGHI, 2011).

2.1.1 Arquitetura Dirigida por Modelos

A *Object Management Group* (OMG) é constituída por várias entidades como empresas, pessoas, instituições, responsável por discutir e padronizar novas propostas no setor de *software*. Em 2003, a OMG propôs uma nova forma de utilização de modelos para tentar solucionar os problemas que surgem em manutenção de sistemas, integração com outros sistemas, e alteração dos requisitos. Portanto, surge o MDA (KLEPPE ANNEKE. ; WARMER, 2003) e (BORGES, 2017).

Segundo (BORGES, 2017), “A proposta da MDA é promover o desenvolvimento de modelos que sejam independentes dos detalhes de implementação, criando-se, portanto, sistemas mais flexíveis e de fácil portabilidade, agregando ganhos imponentes em relação à qualidade do produto devido o foco dos desenvolvedores estar exclusivamente aplicado as regras de negócio, e em relação à produtividade, dado a automatização intrínseca a proposta da abordagem.”

Então, a MDA utiliza modelos como o centro de todo processo de desenvolvimento

de software. A função do MDA é a separação da lógica do domínio do sistema, das funções do sistema, plataforma independente ou de plataforma específica (KLEPPE ANNEKE. ; WARMER, 2003).

No processo de implementação do MDA, primeiro ocorre a utilização do modelo *Computation Independent Model* (CIM), responsável por conter toda a lógica do sistema e regras de negócio. Após a criação do CIM, conforme a Figura 1, vem o *Platform Independent Model* (PIM), gerado por meio do CIM, onde contém todas as funções do sistema sem ser específico para uma plataforma. *Platform Specific Model* (PSM), é gerado a partir do PIM, responsável por conter as características mais específicas de uma plataforma, e por último a geração de código a contar do PSM. A transformação automática é feita de PIM para PSM para código (KLEPPE ANNEKE. ; WARMER, 2003) e (SILVA, 2015).

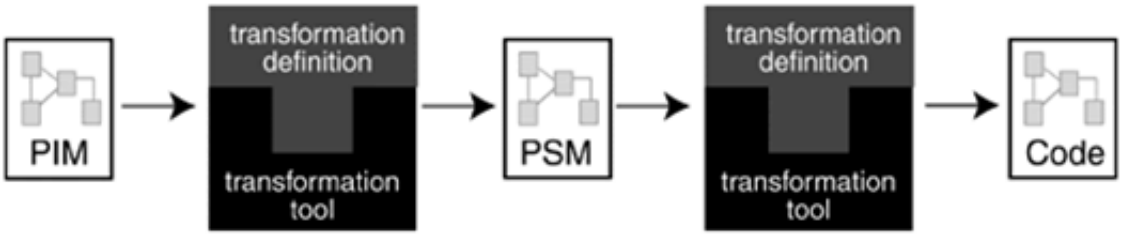

Figura 1 – Transformation definitions inside transformation tools.Fonte:(KLEPPE ANNEKE. ; WARMER, 2003).

As transformações conforme a Figura 1 são feitas por definição de transformação, que descreve o modelo fonte para o modelo alvo, compostos por regras que são necessárias para fazer o mapeamento entre modelos. Todo o processo é executado pela ferramenta de transformação necessária por fazer a transformação entre os modelos (KLEPPE ANNEKE. ; WARMER, 2003), (SILVA, 2015) e (MOHAGHEGHI, 2011).

Uma das ferramentas de transformação, são *atlas transformation language*, que são responsáveis por fazer a transformação de modelos. Os modelos podem ser criados por uma ferramenta chamada *EcoreTools*, que facilita todo o processo MDA e consequentemente os benefícios tragos por eles.

2.1.2 *Ecore Tools*

EcoreTools foram apresentadas primeiro em meados de 2007, e possui várias mudanças no projeto até *EcoreTools* 2, onde permitia a criação de classes tipos de dados, referência de *ecore* (ECLIPSE, 2020b).

EcoreTools tem um editor de metamodelos criado por *Eclipse Modeling Framework* (EMF). Um modelo *'Ecore'* representa os modelos EMF. Portanto, *EcoreTools*, possibilita um ambiente para criar, editar e manter modelos *Ecore* (ECLIPSE, 2020b).

2.1.3 ATL

Atlas Transformation language é uma linguagem de transformação de modelo, permitindo fazer o mapeamento entre os modelos fonte para modelo alvo (ATL, 2020).

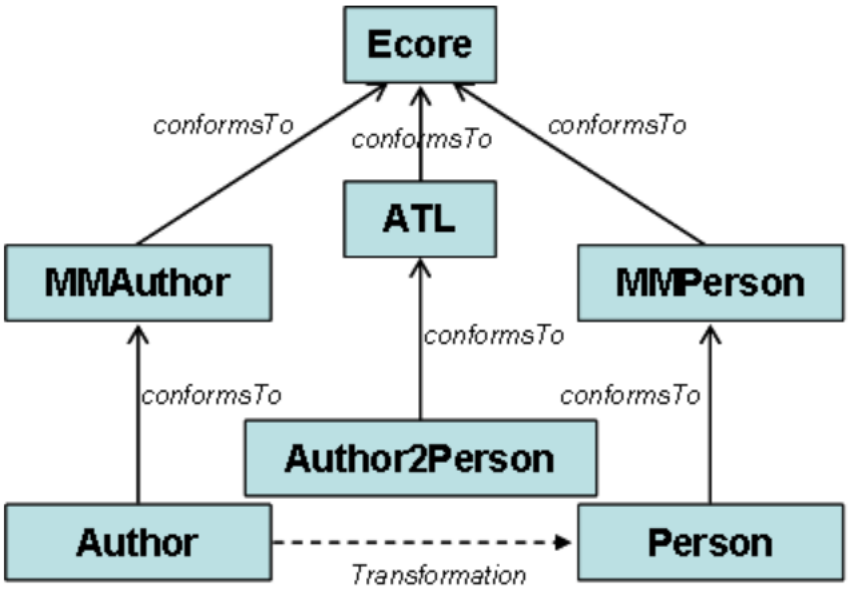

Figura 2 – Transformation ATL.Fonte: (ATL, 2020).

Neste contexto, um exemplo da transformação em ATL se destaca a Figura 2, onde há transformação do modelo *Author* para modelo *Person*. Os modelos são criados conforme seus metamodelos que consequente são baseados na semântica do metamodelo *Ecore*. Para compreender melhor, *Author* e *Person* são modelos, *MMAuthour* e *MMPerson* são seus metamodelos e o Ecore é o meta-metamodelo. *Autor2Person* é o que permite a transformação, que é projetada pela linguagem ATL que está conforme o metamodelo ATL (ATL, 2020).

As transformações são compostas por regras, recursos de consulta de modelos e uma biblioteca que permite a fatoração de código assim facilitando o processo do MDA.

2.2 *Internet of Things*

A *Internet of Things* tem como principal objetivo conectar objetos que a princípios não são conectados à *internet*, que possam comunicar e interagir com as coisas ou pessoas. E que proporciona a capacidade de controlar objetos inteligentes conectados em uma rede inteligente (HANES D. ; SALGUEIRO, 2017).

O termo *Internet of Things* surgiu em laboratórios de pesquisa acadêmica em 1999, focada em desenvolvimento de rede RFID (*Radio Frequency Identification*). Desde então

o termo tem sido amplamente utilizado com a revolução da *internet* e em conjunto o impacto social (KINTSCHNER, 2017) e (GUINARD DOMINIQUE ; TRIFA, 2016).

Conforme a Figura 3, o impacto social em conjunto com a *internet* destaca as fases (HANES D. ; SALGUEIRO, 2017):

- *Connectivity*, utilização de e-mails como meio de comunicação, *web services* e pesquisa na *internet*;
- *Networked Economy*, negócio de forma digital como, e-commerce, gerenciamento da cadeia de suprimentos e colaboração;
- *Immersive Experiences*, interações sociais de forma que envolva a internet, mídias sociais sempre conectada com a mobilidade e armazenamento sendo inserido em nuvens;
- *Internet of things*, um mundo conectado, pessoas, processos computacionais, dados, coisas. Conectar o não conectável.

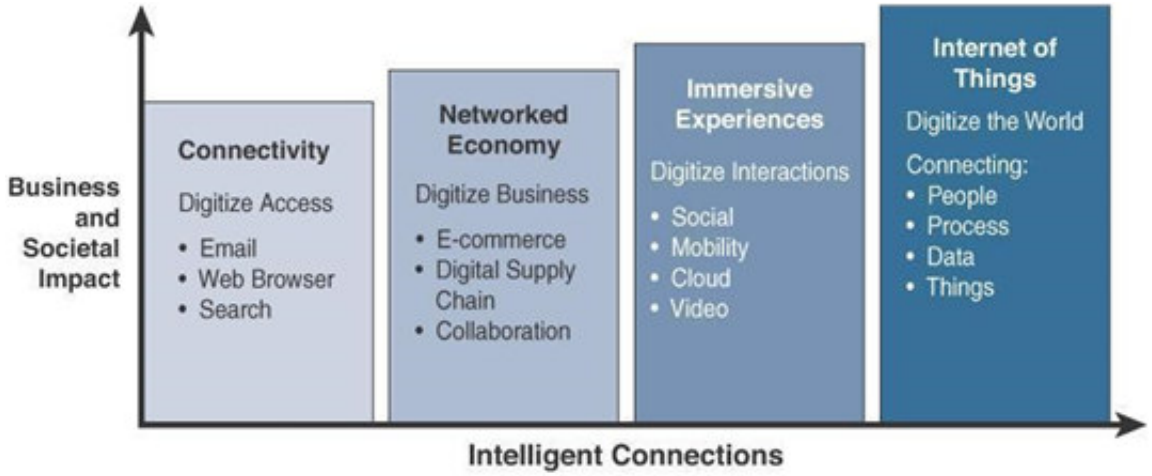

Figura 3 – Fases evolutivas da internet .Fonte:(HANES D. ; SALGUEIRO, 2017).

E conforme a última fase da Figura 3, tem como consequência um alto número de dispositivos conectados à rede. Um relatório do Reino Unido estima que o número de objetos conectados seja maior que a especulação da Cisco em 2020, era de 50 milhões, para 100 bilhões de objetos na rede. Principalmente em cenário de *smart cities* (HANES D. ; SALGUEIRO, 2017).

Um dos problemas da IoT, é que apesar dos números altos de dispositivos, os *softwares* são na maioria, para problemas específicos, e com pouco reuso. Havendo problemas com interoperabilidade entre outros projetos e sistemas, privacidade, armazenamento e heterogeneidade entre dispositivos (KON F; SANTANA, 2016).

Mas existem propostas que preveem melhora para projetos IoT, dentre elas a *Web of Things*, que em conjunto da W3C, promete sanar alguns problemas da *Internet of Things*.

2.3 IoT vs WoT

A intenção da WoT é fazer com que todos os projetos da *Internet of Things* migrem para a camada de aplicação, conforme a Figura 4. Segundo (GUINARD DOMINIQUE ; TRIFA, 2016), os projetos IoT são difíceis de integrar pois a maioria utiliza de protocolos da camada OSI (um até a seis), e isso dificulta na hora de integrar com os mais derivados projetos. Já a WoT utiliza os protocolos da camada de aplicação (HTTP, Json, Websocket etc), muito mais fácil de integrar com a Web. É a camada de consumo dos dados e interação homem-máquina. E a web quebra o conceito "Um protocolo, um dispositivo, uma aplicação".

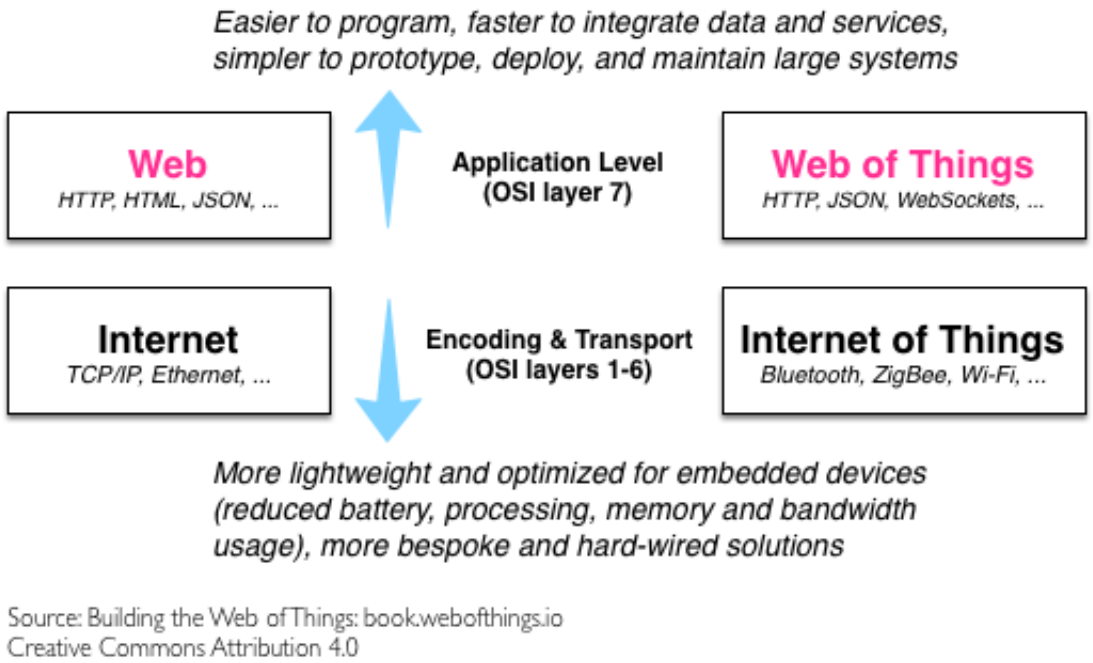

Figura 4 – *The Web of things is concerned with only the highest OSI layer(7), wich handles applications, services, and data. Working with such a high level of abstraction makes it possible to connect data and services from many devices regardless of the actual transport protocols they use. In contrast, the internet of things doesn't advocate a single application-level protocol and usually focuses on the lower layers of the OSI stack* .Fonte:(GUINARD DOMINIQUE ; TRIFA, 2016).

2.4 *Web of Things*

A *Web of Things* tem como princípio facilitar o desenvolvimento de projetos IoT. E tentar solucionar problemas como interoperabilidade, heterogeneidade entre protocolos, etc.

A WoT é um assunto que surgiu nos meados dos anos 2000 e desde então teve uma influência bastante crescente sobre o assunto. Em 2016 a W3C, decidiu padronizar a WoT para possuir um padrão que todos pudessem implementar (W3C, 2020d). Portanto, é necessário perceber os pontos de IoT e WoT e cada módulo citado para pela W3C.

2.4.1 Arquitetura WoT

Segundo (GUINARD DOMINIQUE ; TRIFA, 2016), a Web of Things possui uma arquitetura de quatro camada como podemos perceber na Figura 5:

- *Networked Things*: É a camada onde os dispositivos físicos irão se comunicar com a camada 1. Responsável por captar as informações de sensores e atuadores. Utilizando ainda os protocolos das camadas 1 e 7 do modelo OSI;
- *Layer 1 Access*: É a primeira camada, que conecta os dispositivos com web, onde o desenvolvedor não precisa se preocupar com os protocolos usados na *Networked Things*. Os dispositivos que não tem acesso a internet, normalmente sua comunicação é feita por meio do *gateway*;
- *Layer 2 Find*: A proposta dessa camada é para que as informações fiquem modeladas para o cliente, é a forma como as Things vão ser descritas e seus serviços entregues para as outras camadas. Apesar da camada 1 proporcionar a heterogeneidade, como não se possui um único modelo, dificulta a integração em larga escala;
- *Layer 3 Share*: Essa camada dedicada ao compartilhamento de dados para big data ou banco de dados não relacional. Também é necessário a preocupação de como os dados serão compartilhados principalmente em relação a segurança;
- *Layer 4 Compose*: Última camada onde a intenção é fazer a composição de todas as outras camadas para possibilitar a criação de aplicações, e poder aderir a outras propostas.

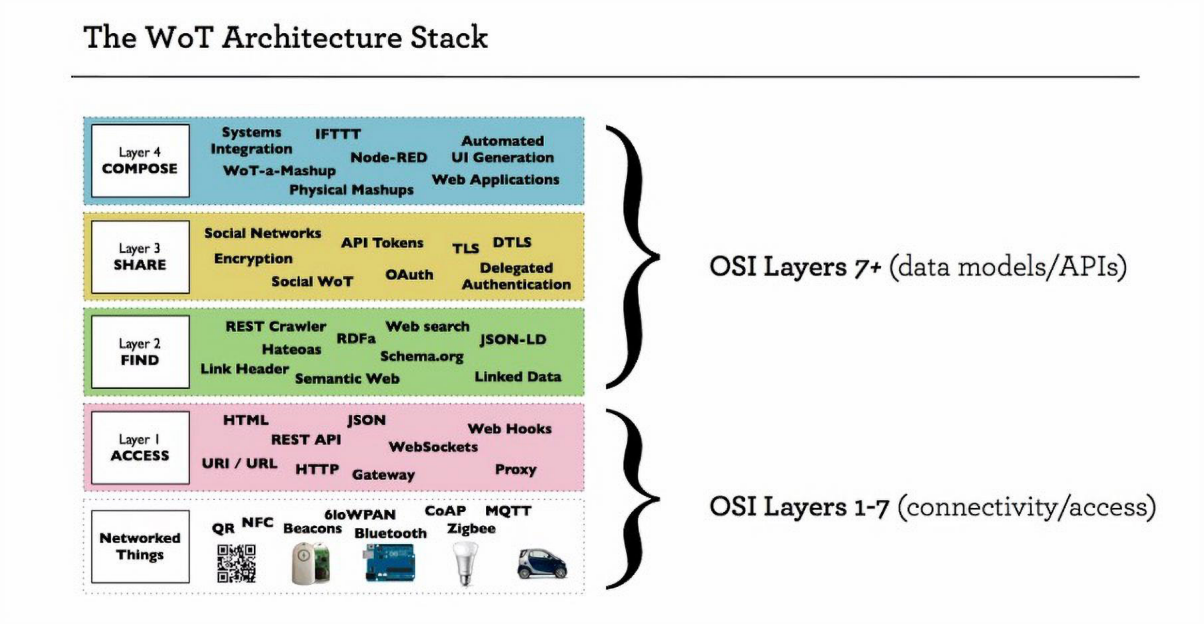

Figura 5 – *The WoT Architecture Stack*. Fonte: (LAB, 2021).

A arquitetura do (GUINARD DOMINIQUE ; TRIFA, 2016) descreve como construir a *Web of Things* ou iniciar na proposta, mesmo não sendo considerado padrão para implementar como outras arquiteturas.

Outras argumentações sobre arquitetura, é a necessidade de padronização para cada camada, e assim proporcionar maior integridade e heterogeneidade. Por isso o surgimento da padronização da W3C para WoT, na intenção de unificar mais os padrões e aumentar a interoperabilidade.

2.4.2 Padrão WoT W3C

A *World Wide Web Consortium* (W3C) como principal organização de padrões na *World Wide Web*, em 2016 começou o processo de padronização da WoT (KINTSCHNER, 2017) e (W3C, 2020d).

Para criação da padronização existem 2 grandes grupos *Interest Group*, responsável por reunir, criar e estudar sobre o assunto abordado dentro de um fórum e o *Working Group* responsável por padronizar tudo que a IG criou. *Working Group* fica responsável na criação dos rascunhos da documentação criada no *GitHub* (W3C, 2020a).

2.4.2.1 *WoT Architecture*

WoT Architecture se descreve como a arquitetura deve ser utilizada para que os recursos inteligentes possam ser consumidos e proporcionar maior usabilidade, interoperabilidade principalmente em projetos IoT.

Para compreender melhor a composição da arquitetura destacamos *Servient* da arquitetura abstrata utilizando *WoT Scripting Api* na Figura 6. A arquitetura se destaca em 4 blocos (W3C, 2020e):

- *Application Script*: Responsável por compor toda a lógica da *Thing*, sendo sensor ou atuador. E o modelo de interação composto por propriedade, eventos e ações;
- *Scripting WoT Runtime*: Em tempo de execução é responsável por expor, consumir e criar a TD. *Private Security Data*, responsável por possuir uma chave que é compartilhada sempre que há interação com a *Thing*;
- *Protocol Stack Implementation*: Onde está inserida a *interface* para que as interações de consumo e de exposição da *Thing* possam ser realizadas de forma remota por meio dos protocolos. Podendo fazer interação com vários protocolos e com projetos IoT;
- *System*: Implementação de *WoT Runtime* na utilização de serviços de sistema ou de *hardware*.

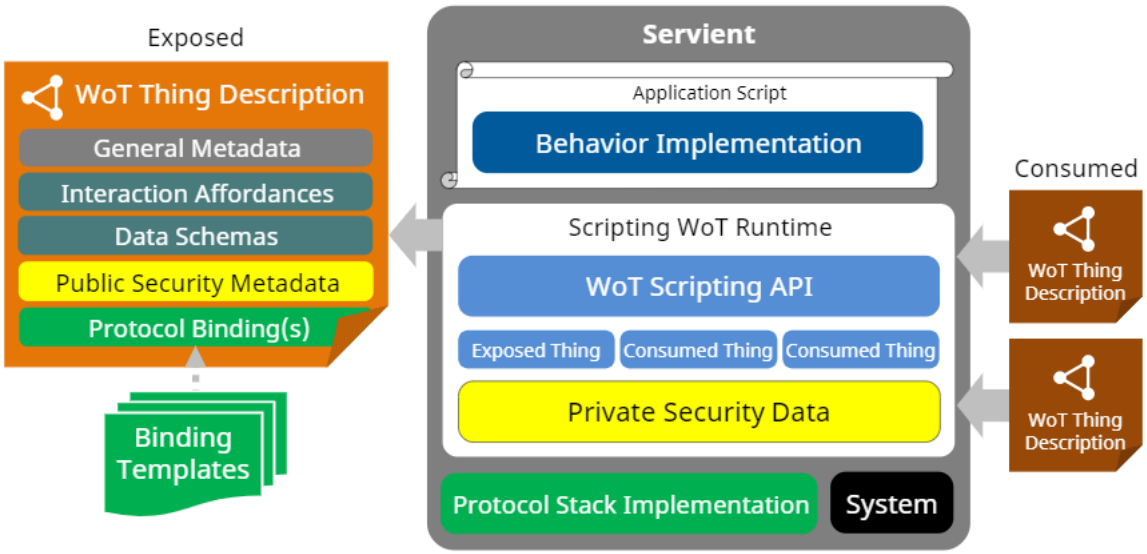

Figura 6 – Implementation of Servient using the WoT Scripting. Fonte:(W3C, 2020e).

A comunicação do *Servient* pode ser realizada como *Thing* para expor *(Exposed Thing)* ou sendo usada para consumir a *Thing (Consumed Thing)* ou a junção dos dois como *Intermediary* (W3C, 2020e).

Todos os pontos da arquitetura e os tipos de comunicação ajudam a entender como melhor construir e implementar melhor a WoT, mas para se aprofundar ainda mais na Figura 6 os módulos que completam o *Servient* são *Thing Description, Binding Templates e Security Data*.

2.4.2.2 *WoT Thing Description*

O módulo da *Thing Description* é destacada como uma das partes mais importantes da *Web of Things* da W3C (W3C, 2020i). Pois, é o ponto principal para a interação com a *Thing*. TD é composta por um vocabulário composto de metadados, modelo de interação, que descreve e manipulam uma *Thing* sendo virtual ou não.

Os modelos de interação são compostos por (KINTSCHNER, 2017) e (W3C, 2020i):

- Propriedade: Expõe as características da *Things*, seus valores e seu estado;
- Ações: Representa as funções da *Thing* e podem mudar o estado das propriedades;
- Eventos: Representa a mudança de estado e consequentemente criando um evento causado pela *Thing*.

Por meio dos modelos de interação, podem acontecer a manipulação da *Thing*. A TD gera metadados para diferentes tipos de protocolo de ligação por meio das URIs (HTTP, CoAP, etc), tipos de midia (JSON, XML, etc) e mecanismo de segurança (autenticação, autorização, etc) (W3C, 2020i). A serialização da TD é baseada em JSON, para que tanto máquina quanto humano pudessem ler.

Para criar a TD de forma correta, na documentação W3C possuem um modelo UML para poder modelar um sensor ou atuador, conforme na Figura 7. Portanto, destacamos suas classes (W3C, 2020i):

- *Thing*, a parte principal do modelo, contendo os atributos necessários para construção da *Thing*;
- *Interaction Affordances*, é uma superclasse que acopla três classes herdeiras utilizadas para interagir com a *Thing* (*Properties, Action e Events*);
- *VersionInfo*, versão de quando a *Thing* foi criada ou modificada;
- *Link*, promove a ligação entre a *Thing* e a *interaction affordances*;
- *Security Scheme*, descreve a metadata de mecanismo de segurança;
- *Forms*, pode ser utilizado para controlar hipermídia na web, recursos e o estado da thing como método ou destino de envio, na TD especificamente *Properties, Action, Events*;
- *Data Schema*, definição de metadados, especificamente *data types*.

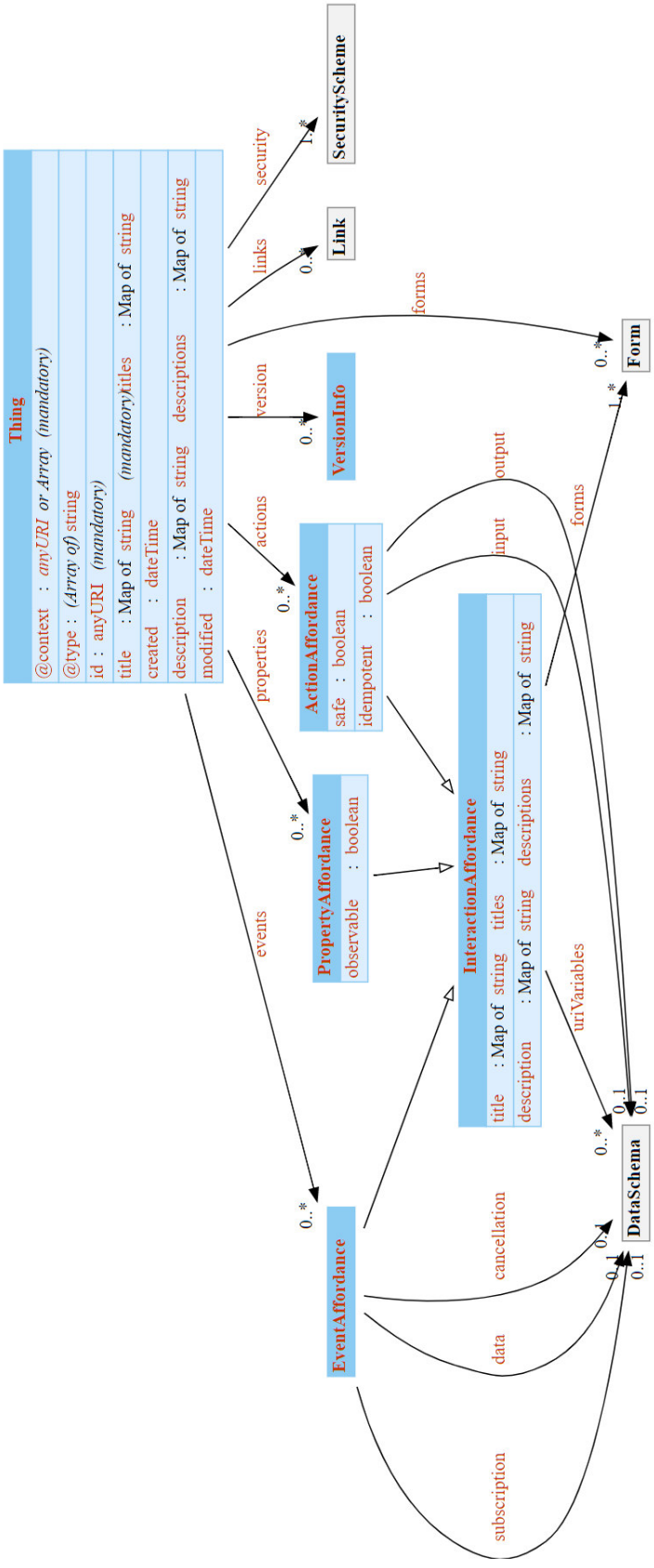

Figura 7 – Diagrama UML Thing Description. Fonte:(W3C, 2020i).

Após a modelagem da criação de uma *Thing Description*, cada API da Web na TD é identificada com uma URI, o resultado deverá ser um *JSON* serializado como podemos perceber na Figura 8. É possível ver a descrição de um dispositivo WoT chamado *MyLampThing* cuja única *properties* é de ligar ou desligar (*On Off*). Para obter o status basta acesssar o link com o protocolo *HTTP* (https://mylamp.example.com/ status).

Action é para mudar a propriedade da lâmpada podendo ser acessada pelo *link* (https://mylamp.example.com/toggle), utilizando o metodo *POST*. Para o *event*, que está descrito como *overheat*, em caso de aquecimento além do limite normal da lâmpada, mensagens assíncronas são disparadas podendo ser acessada pelo *link* (https://mylamp.example.com/oh).

Os modelos de interação ou *interaction Affordances* também podem ser manipulados por outros protocolos além do *HTTP*, a documentação da W3C padronizou três protocolos principais em *Binding Templates* para a manipulação da TD.

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "id": "urn:dev:ops:32473-WoTLamp-1234",
    "title": "MyLampThing",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in":"header"}
    },
    "security": ["basic_sc"],
    "properties": {
        "status" : {
            "type": "string",
            "forms": [{"href": "https://mylamp.example.com/status"}]
        }
    },
    "actions": {
        "toggle" : {
            "forms": [{"href": "https://mylamp.example.com/toggle"}]
        }
    },
    "events":{
        "overheating":{
            "data": {"type": "string"},
            "forms": [{
                "href": "https://mylamp.example.com/oh",
                "subprotocol": "longpoll"
            }]
        }
    }
}
```

Figura 8 – Thing Description MyLampThing. Fonte: (W3C, 2020i).

2.4.2.3 *Bind Templates*

A forma que os protocolos se comunicam pela TD é por meio dos métodos através da URI. A *Binding Templates* fornecem diretrizes e extensões que permitem que clientes possam usar TD com diferentes tipos de protocolos (KINTSCHNER, 2017) e (W3C, 2020f).

Especificamente os protocolos bases como (HTTP, CoAP e MQTT), pois, possuem métodos padrões da arquitetura *REST* e *PubSub* em comum como GET, PUT, POST, PUBLISH and SUBSCRIBE (W3C, 2020f), por meio das URI acessada por esses protocolos, permitindo o acesso aos modelos de interação (propriedades, ações e eventos) autorizando a leitura e gravação.

Os métodos de operação são caracterizados conforme os modelos de interação, as propriedades utilizam *readproperty, writeproperty, observepropert, unobserveproperty, readallproperties, writeallproperties, readmultipleproperties e writemultipleproperties.* Ações utilizam *invokeaction.* E por último os eventos utilizam *subscribeevent e unSubscribeevent* (W3C, 2020f).

Para que se utilizem os métodos de forma correta com os protocolos de comunicação, existe um mapeamento entre métodos de operação com protocolos de comunicação como mostra a tabela 1 para facilitar a utilização dos protocolos com a *Thing Description.*

Tabela 1 – Mapeamento em métodos de operação e protocolos de comunicação

Métodos de Operação	HTTP	MQTT	CoAP
Readproperty	GET	SUBSCRIBE	GET
Writeproperty	PUT	PUBLISH	PUT
Observproperty	Não possui	SUBSCRIBE	GET E OBSERVE
Unobservproperty	Não possui	UNSUBSCRIBE	GET E OBSERVE
Invokeaction	POST	PUBLISH	POST
Subscribevent	Não possui	SUBSCRIBE	GET E OBSERVE
Unsubscribevent	Não possui	UNSUBSCRIBE	GET E OBSERVE
Readallproperties	GET	SUBSCRIBE	GET
Writeallproperties	PUT	PUBLISH	PUT
Readmultipleproperties	GET	SUBSCRIBE	GET

Após os mapeamentos torna-se muito mais fácil a implementação e utiização da *scripting Api.*

2.4.2.4 Scripting API

WoT Scripting Api é uma parte opcional, para facilitar o desenvolvimento de aplicações através da *interface* WoT, que permite que os *scripts* possam identificar clientes,

servidores, e descoberta de "coisas", proporcionando interoperabilidade entre eles. Em sua documentação se destacam em três pontos (W3C, 2020g):

- Consumir a *Thing Description*: Responsável por ler e observar as propriedades, invocar as ações e observar eventos. A nomenclatura se descreve como *CosumedThing interface*, possuindo os métodos (*getThingDescription(), readProperty(), readMuiltiple-Properties(), readAllProperties(), writeProperty(), writeMuiltipleProperties(), observe-Property(), invokeAction() e subscribeEvent()*);
- Expor a *Thing Description*: Criar primeiramente a *Thing* a ser exposta, gerar protocolo, adicionar e remover (propriedades, ações e eventos). A nomenclatura se descreve como *ExposedThing interface*. Herdando todos os métodos citados em *ConsumedThing* e implementam outros como *setPropertyHandler(), setProperty-ObserveHandler(), setPropertyNnobserveHandler(), emitPropertyChange(), setProperty-WriteHandler(), setActionHandler(), setEventSubscribeHandler(), setEventUnsubscribe-Handler(), setEventHandler(), emitEvent(), expose() e destroy()*;
- Descoberta: Executa por rede, *WoT Runtime*, é feita por aproximação, utilizando *Thing Dicrectory*. Também pode parar a descoberta e especificar o tempo de processamento. A nomenclatura se descreve como *ThingDiscovery interface*. Utilizados de métodos *start(),next() e Stop()*.

WoT Scripting Api, foi criada para que várias linguagens de programação pudessem implementá-la. Apesar da documentação utilizar *Typescript* e fazer indicações da biblioteca *Node-WoT* para implementar a *WoT Scripting Api* (W3C, 2020g).

2.5 Protocolo de comunicação WebSockets

WebSockets (ws) surgiu com *HTML* 5, permitindo a comunicação em tempo real entre cliente e servidor de forma bidirecional através de somente um *socket* (LOMBARDI, 2015) e (WANG V.; SALIM, 2013).

Os *WebSockets* possuem baixa latência, pois, uma vez que a comunicação entre cliente e servidor se estabelece poderá haver troca de mensagem sempre que possível, não havendo mais espera de solicitação, e assim diminuindo a latência (WANG V.; SALIM, 2013).

A proposta inicialmente beneficiou as funcionalidades em tempo real para jogos *multiplayer online*, conversas estilo chat e edição de documentos (WANG V.; SALIM, 2013). Obtendo WS tanto como em protocolo como API, permitem que respondam eventos acionados pelo servidor. *WebSockets* Api é baseada em quatro eventos que são destacados por(LOMBARDI, 2015):

- Aberto: Servidor responde a solicitação de conexão com *handshake*, que utiliza HTTP como transporte, assim a conexão é estabelecida, estar pronta para enviar as mensagens possibilitando a comunicação bidirecional;
- Mensagem: Após a conexão estabelecida, estará pronta para o envio e recebimento das mensagens;
- Erro: Quando ocorre a falha na comunicação, a conexão será encerrada, um evento é disparado e possivelmente uma explicação do erro;
- Fechado: É disparado sempre que a conexão for encerrada, não havendo mais a possibilidade de comunicação entre cliente e servidor.

WS também está sendo utilizado em propostas IoT, que necessitam de comunicação em tempo real, tornando algumas soluções mais escaláveis [25]. Portanto, se torna bastante cabível a utilização de WS, para conectar soluções *Web of Things*, com proposta de IoT *devices*.

2.6 Proposta *Mozilla Web of Things*

A proposta do *Mozilla* foi iniciada em 2017 e é derivado das propostas trazidas pela W3C. Em seu site existem duas grandes propostas, um *framework* responsável por fazer a programação de sensores, atuadores e expor com API da biblioteca *WebThing*, e *gateway*, criado para fazer o monitoramento, controle e identificação da *Thing* na rede (MOZILLA, 2020b) e (MARTINS A. J.; MAZAYEV, 2017).

O *framework* permite que consumidores possam manipular a TD. A biblioteca permite ser programada em várias linguagens como *Node. Js, Python, Java, Arduino (C e C + +) e Rust*, que permite que o *gateway* possa identificar cada componente e controlá-lo (MOZILLA, 2020b).

Em sua documentação, *API da Thing Description format* segue os mesmo princípiosde modelo de interação da W3C seguindo por propriedades, ações e eventos.

Porém, que se diferenciam pois a interação com *Mozilla* é por meio de links e a W3C, por meio de formulários. O *Mozilla* também não possui descrição, título, data de criação ou modificação. E o *@context* da TD, se difenciam por W3C *(https://www.w3.org/2019/wot/td/v1)* e *Mozilla (https://iot.mozilla.org/schemas)* (MOZILLA, 2020b)

Outro ponto de diferença entre *Mozilla* e a W3C. E que a W3C depende da Api e de protocolos para fazer a comunicação com a TD, já o *Mozilla* se utiliza do *gateway* que faz comunicação e o controle da *Thing* (MOZILLA, 2020a) de forma simples por uma interface de um site.

2.7 Síntese

Neste capítulo, todos os conceitos e tecnologias necessárias para a construção do *framework* foram apresentadas, permitindo um melhor entendimento do seu desenvolvimento.

Portanto, a leitura dos próximos capítulos será feita de forma clara e objetiva dos tópicos abordados como, MDE, MDA, *EcoreTools*, IoT, *Web of Things*, *WebSockets* e *Mozilla Web of Things*.

No próximo capítulo, pretende-se explorar os trabalhos relacionados aos assuntos. MDE com WoT, *WebSockets*, IoT e padrão W3C, para posteriormente ser comparada com o *framework* proposto.

3 Estado da Arte

Neste capítulo, trabalhos são apresentados com o objetivo de destacar as principais abordagens para suportar o desenvolvimento de propostas WoT em conjunto com W3C, propostas com abordagens de MDE para desenvolver sistemas WoT e sistemas de IoT. Destacando suas características, metodologias e ferramentas específicas para este tipo de aplicações.

3.1 Abordagens para o desenvolvimento de aplicações *Web of Things* com W3C

Nesta seção, alguns trabalhos relacionados a padronização da W3C para *Web of Things*.

3.1.1 *Virtual-Thing: Thing Description based Virtualization*

(HASSINE HB. ; KORKAN, 2019) destaca que as aplicações IoT normalmente utilizam dispositivos que contém seu próprio *software* e sua *interface* para manuseá-los, o que dificulta o gerenciamento e interoperabilidade dos projetos IoT.

Portanto, utilizaram a proposta da *Thing Description* da W3C para solucionar os problemas encontrados nas propostas IoT, por descreverem os metadados, *interface* e consequentemente proporcionar menos complexidade na hora do desenvolvimento. Onde os desenvolvedores podem possuir dificuldade em implementar um *software* e logo manipular a TD.

Então, o problema identificado é que se utilizaram da criação de uma TD *virtual* para que programadores pudessem manipular e testar a *Thing* sem necessitar de um *software* para manipulá-la ou de dispositivo físico. Na figura 9, demonstra a arquitetura criada pelos autores (HASSINE HB. ; KORKAN, 2019), destacadas nos pontos:

- *Original TD*: Descrita conforme a documentação da W3C de um sensor ou atuador em *TypeScript*;
- *Node-WoT Servient*: Cliente e servidor, promove a criação da TD e manipulação de protocolos como *HTTP, MQTT e CoAP*;
- *Virtual Thing*: Simula as ações feitas pela TD;
- *Thing Description*: TD disponível por uma URI para manipulação e teste;

- *IoT Controller*: Um dispositivo físico para manipular a TD.

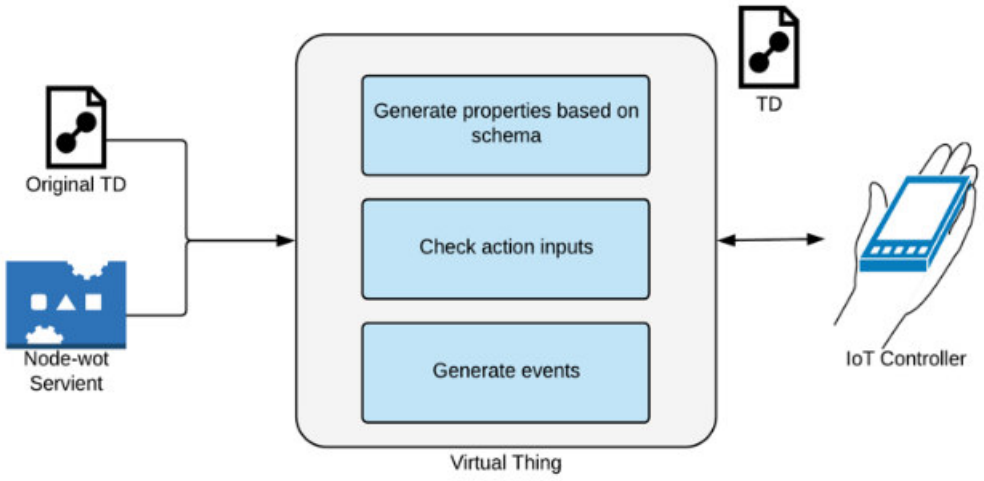

Figura 9 – *Virtual Thing architecture*. Fonte:(HASSINE HB. ; KORKAN, 2019).

Também, destacaram algumas limitações na utilização das *virtuais things*, como URL, não são arbitrárias, sendo sempre criadas pela biblioteca. Não sendo possível o cancelamento de inscrições de propriedade e eventos. E também não há identificação de dados escritos de forma incorreta, apesar de proporcionar a simulação dos sensores, atuadores e proporcionar o teste.

3.1.2 *Dynamically Exposing and Controlling Physical Devices by Expanding Web of Things Scheme*

(SAKAMOTO T. ; ITO, 2017) destaca os benefícios trazidos pela IoT para dispositivos em serviços de nuvens, e a promessa da IoT ser utilizada também na camada da web pela WoT. A proposta, é que a abordagem permita expor as *Things* de forma dinâmica gerenciando os dispositivos em nuvens usando API baseada em WoT. É uma *interface* que se adapta ao usuário sempre que utilizar uma API diferente, para controlar os dispositivos.

O problema identificado é desenvolver uma plataforma flexível, pois, os *Web Services* para *Web of Things* se diferenciam muito de uma web service normal, pois, precisa se adaptar e sincronizar com os dispositivos. Portando desenvolveram uma arquitetura na Figura 12 baseada nos padrões da W3C para *Web of Things* que funciona da seguinte forma:

- *Device-With Conversion*: Responsável por baixar e instalar os *drivers* de dispositivos descobertos, fazendo chamada para expor API do *script* e por registrar a TD em um repositório onde o *gateway* acaba fazendo a função de *WoT Servient*;

- *User permission change*: Onde há a confirmação da TD registrada, e assim podendo ser associado a um identificador, onde o usuário terá acesso;
- *Device observation*: Verifica as regras do JS e, se a TD confirma com o ID do usuário de acesso;
- *Dynamic exposing*: Para uma exposição dinâmica é necessário um verificador de regras, para verificar as URIs na TD de cada dispositivos estão acessíveis pelos usuários e seus *scripts*. Cada *script* possui suas próprias regras que são adequadas para cada usuário, e as regras incluem o local do *gateway* e do servidor onde os *scripts* são executados. Também são responsáveis por expor a TD;
- *Server access by client*: Para o acesso do cliente se faz necessário possuir o ID, para ter acesso a TD;
- *User interface composing*: A aplicação do usuário possui o acesso ao armazenamento das *interfaces*;
- *Device control*: Os usuários que possuem o controle da API da *web*. O *script* no servidor controla o dispositivo. Portanto, no fluxo o *gateway* comunica com o dispositivo por meios dos *drives* e retorna o resultado, ao servidor que enviar para aplicação do usuário.

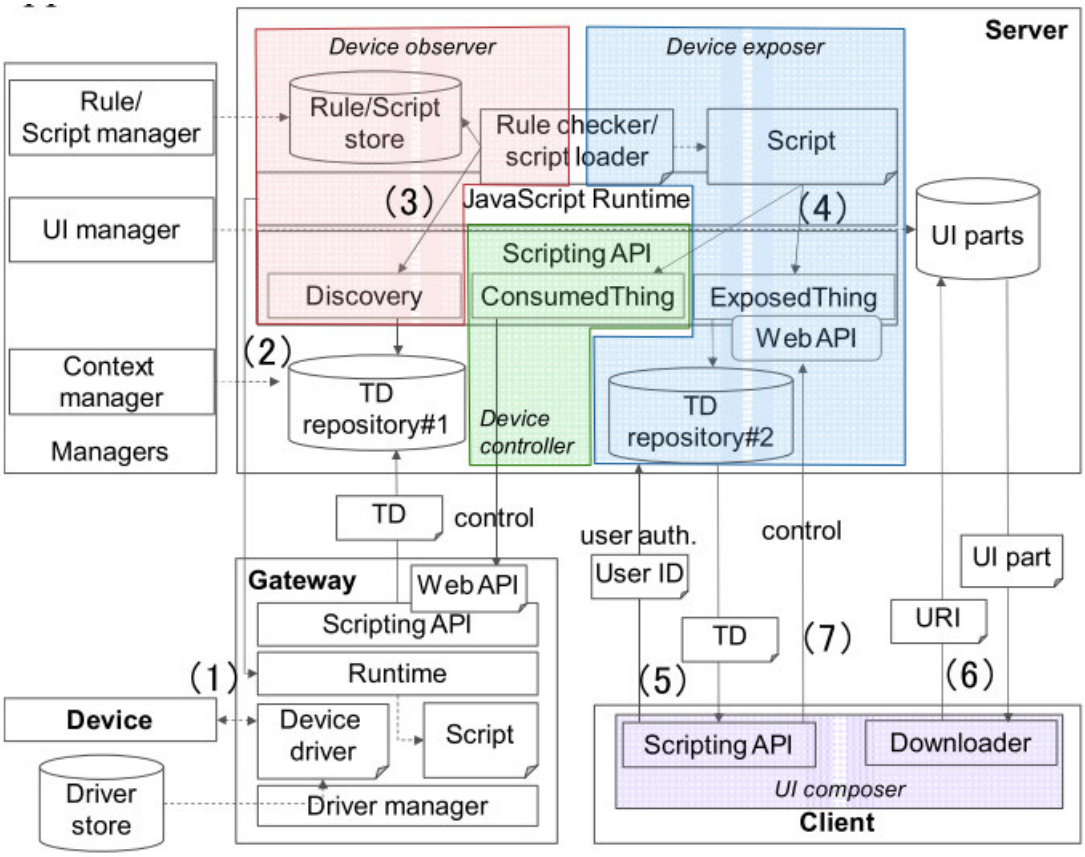

Figura 10 – *Architecture design of cloud service that exposes Web API dynamically and client with composed user interface. Fonte: (SAKAMOTO T. ; ITO, 2017).*

Portanto, a proposta consiste em desenvolver a plataforma que expõe as *Things* de forma dinâmica, por vários dispositivos IoT para ambientes de contexto de forma flexível e se beneficiando das propostas trazidas pela W3C.

3.2 Abordagens baseadas em MDE para o desenvolvimento de sistemas *Web of Things*

Nesta seção, alguns trabalhos relacionados a padronização da *Web of Things* utilizando a abordagens de *Model Driven Engineering*.

3.2.1 *A model-based approach to generate reactive and customizable user interfaces for the Web of Things*

Segundo (BEZERRA, 2019) a *Internet of Things*, se destaca no cenário de sistemas embarcados de cidade inteligentes e consequentemente um alto nível de dispositivos conectado a *Internet*, mas com essa ascensão também surgiram alguns problemas entre heterogeneidade de protocolos, serviços e *hardware*.

Existe a proposta da *Web of Things* que promete sanar alguns problemas, mas os autores destacam que mesmo que existisse a padronização ainda haveria alguns problemas em interagir com WoT. Isso é porque alguns usuários precisam conhecer a interface para interagir com a *Thing* sendo específicas de cada.

O artigo propõe uma implementação de estudo de caso com conceitos de *Model Driven Engineering* (MDE) combinado com uma linguagem específica de domínio implementado em tempo de execução, mostrando que é possível criar uma interface reativa que interaja com qualquer WoT de acordo com a TD.

Portanto, criaram um componente de diagrama capaz de implementar o consumo de TD e adaptação de interface para manipular WoT. A linguagem de programação *JavaScript* utilizando *Vue Js* de modelos em tempo de execução. Podemos perceber o diagrama *TD UIComponent* na Figura 13.

TD UIComponent, só precisa receber uma TD, pelo *component Factory* e *Component Aggregator*, é responsável para mostrar e criar e gerenciar os eventos, mostrando aos desenvolvedores qualquer evento e passando para a *actuation manager* quando existe uma interação que precisa fazer *request*. O *actuation manager*, que cria chamada de acordo com URL específico de TD, notifica o *Component Aggregator* se a resposta envolver qualquer mudança de *interface*. Essas chamadas também estão disponíveis por programadores se eles quiserem gerenciar a lógica externa.

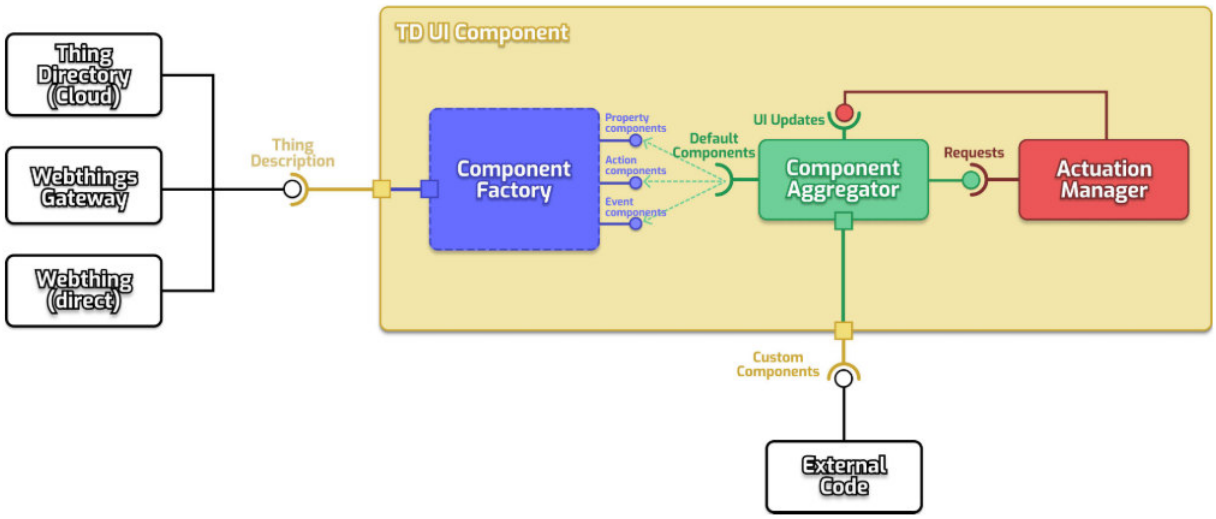

Figura 11 – *TD UI Component diagram depicting each component and their relations. Fonte: (BEZERRA, 2019).*

Mas os autores (BEZERRA, 2019) estacam algumas limitações em relação às *Vues JS*, pois, não é possível alterar os componentes em tempo de execução, sobrecarregando na criação de componentes diferentes. Então para trabalhos futuros encontrar uma forma de implementar a hierarquia dos componentes.

3.2.2 *A development methodology to facilitate the integration of Smart Spaces into the Web of Things*

O trabalho (CORREDOR, 2012), tem como objetivo a integração de sensores e atuadores em *Smart Spaces*. Para isso utilizaram arquitetura orientada a serviços em conjunto com as ontologias dirigidas por desenvolvimento (ROOD), uma metodologia baseada em arquitetura orientada por modelos. A metodologia tem como principal objetivo que as ferramentas MDE possam facilitar a prototipação de *Smart Spaces* de acordo com o conceito WoT.

Portanto, criaram uma linguagem de modelagem para espaços inteligentes (SSML), com objetivo de modelar, recursos, serviços e plataformas. A Metodologia ROOD gerencia os modelos que são criados pela SSML: modelo de objeto inteligente e modelo de contexto de ambiente.

Na Figura 14, demonstra as ontologias e as transformações utilizando MDA. MDA se destaca pela transformação de CIM, para PIM, para PSM, para código utilizado nas metodologias como diretrizes.

No trabalho (CORREDOR, 2012), destacam que apesar da implementação da metodologia foi possível fazer simulações para cenários de um hotel, em relação à qualidade do ambiente, em informações simplificadas para turistas e monitoramento de cuidados com a saúde, ainda é um recurso visto de um ambiente muito abstrato, onde os fluxos dos dados são modelados. E que para o futuro do trabalho a melhoria da metodologia ROOD. Infelizmente, apesar da arquitetura baseada na arquitetura REST, não estão baseadas na nova padronização da W3C para *Web of Things*.

3.2.3 *Enabling easy web of things compatible device generation using a model-driven engineering approach*

O trabalho (URKIA, 2019), tem como foco proporcionar a interoperabilidade de proposta IoT com WoT, e reuso de recursos. Então eles implementaram abordagem MDE com *Web of Things* W3C permitindo uma fácil implementação e fácil reutilização de design e código baseado na linguagem C + + para os dispositivos, com o protocolo de comunicação CoAP.

Portanto, os autores desenvolveram o *framework* na Figura 15. Primeira parte é um metamodelo *Ecore* representando o WoT *Model*. O *metamodel* na transformação cria a extensão de arquivo *specific model*. WoT e por último a transformação do código.

Infelizmente apesar de implementar o padrão de *Thing Description*, na utilização da *interface*, implementa apenas *consumedThing interface*, e sendo uma proposta estrita para a linguagem C ++.

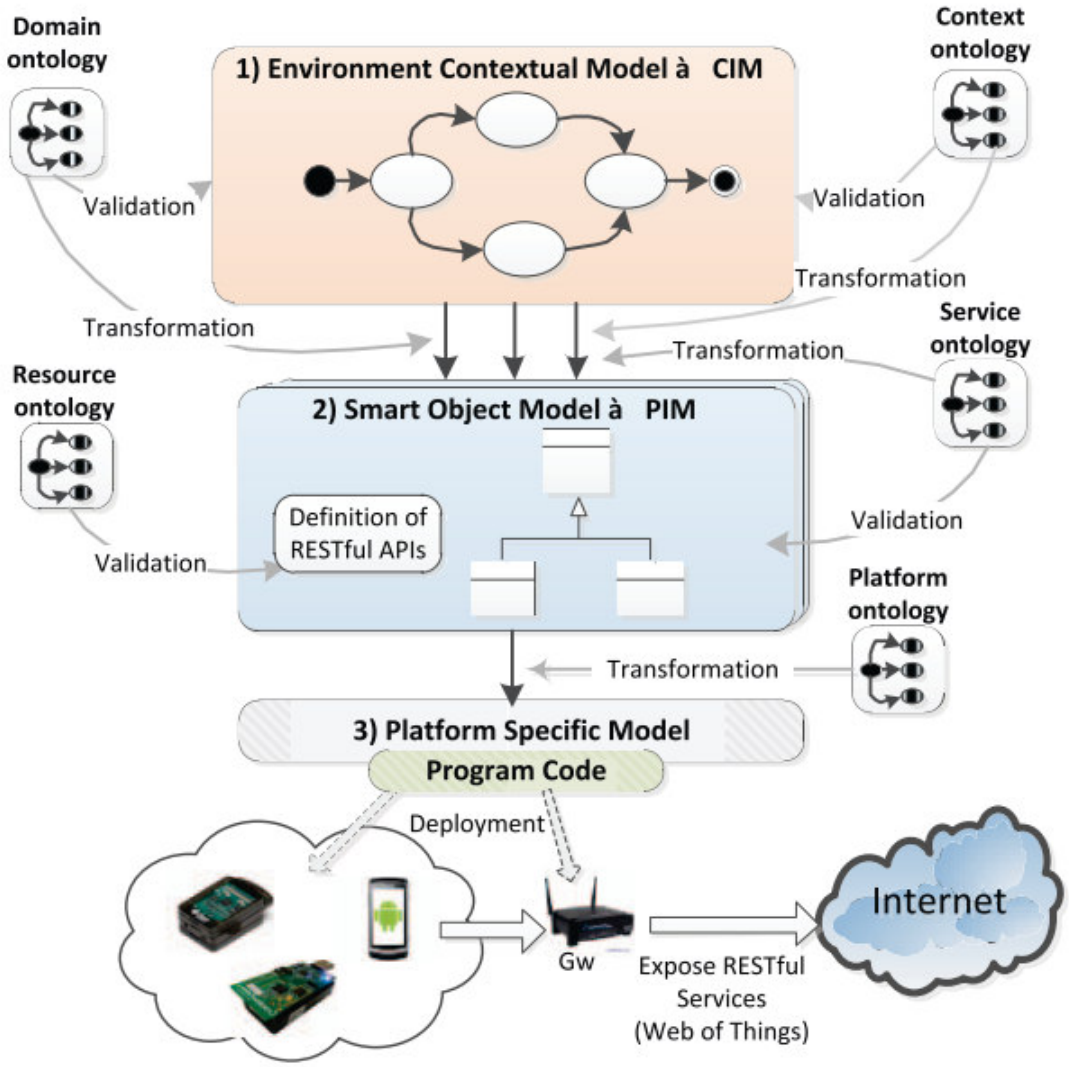

Figura 12 – *The Resource-Oriented and Ontology-Driven Methodology. Fonte: (CORREDOR, 2012).*

3.3 Abordagens baseadas em MDE para o desenvolvimento de sistemas *Internet of Things*

Nesta seção, alguns trabalhos relacionados utilizando *Internet of Things* e abordagens de *Model Driven Engineering*.

3.3.1 *A model-driven approach for the integration of hardware nodes in the IoT*

A *Internet* está crescendo em conjunto com os dispositivos e suas complexidades principalmente pelo número de bilhões de dispositivos em uma rede de *Internet* e

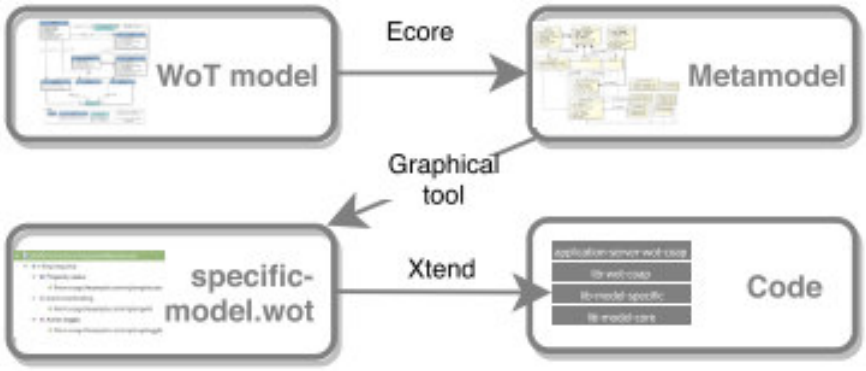

Figura 13 – *Workflow and the used tools, with intermediate results before the final code.Fonte: (URKIA, 2019).*

consequentemente a heterogeneidade de *hardware* e *software* para manipulá-lo (ALULEMA D. ; CRIADO, 2019).

Portanto, os autores (ALULEMA D. ; CRIADO, 2019) destacam a necessidade de possuir habilidade na programação de baixo nível, relacionada a plataforma de hardware em implementar esses componentes.

Para solucionar a complexidade, criaram uma metodologia utilizando abordagem MDE em conjunto à linguagem de domínio específico (DSL) para desenvolver nós de hardware para se adaptar em novos cenários. A metodologia utiliza *Eclipse Modeling Framework* para o desenvolvimento, utilizando o metamodelo que descreve os nós de *hardware* como *arduino* e *Raspberry Pi*, que possuem conectividade (*USB, Bluetooth, WiFi e Ethernet*) para serem associados com outros componentes. Também permite controlar sensores e atuadores.

Na representação do *hardware*, existe o editor gráfico na Figura 16, que possui duas seções: primeiro editar os diagramas, segundo ferramentas que desenham e editam o visual.

Incluindo serviços suas conexões dos nós de IoT e seus componentes. Os controladores exibem a placa em desenvolvimento, as comunicações são feitas pelo protocolo de comunicação. As entradas e saídas são de sensores e atuadores.

Essa abordagem permite padronizar as descrição de aplicações para definir novas arquiteturas sem precisar um conhecimento profundo da aplicação. Mais para trabalhos futuros, aplicações de metodologia e novas plataformas.

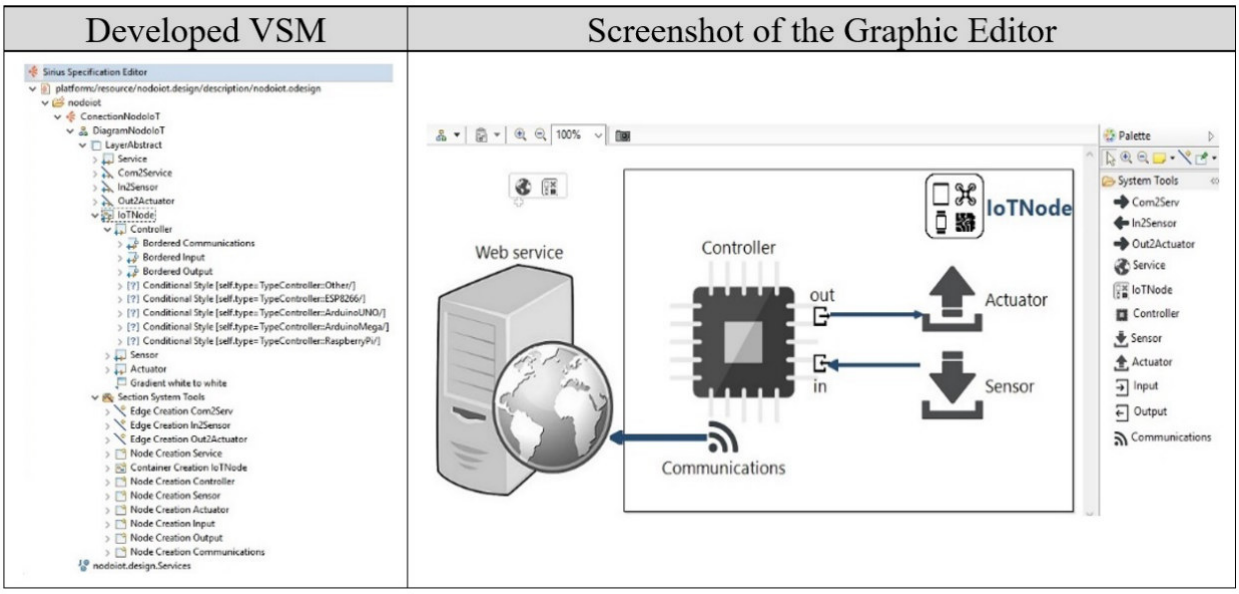

Figura 14 – *Screenshot of the tool for developing the Graphical Editor. Fonte: (ALULEMA D. ; CRIADO, 2019).*

3.3.2 *AutoIoT: A framework based on User-driven MDE for generating IoT applications*

Os autores (NEPOMUCENO, 2020) destacam uma alta aderência em trabalhos que utilizam proposta MDE para IoT e por meio dela melhorar sua complexidade encontrada pelos autores. Os usuário precisam ter conhecimento da abordagem MDE para utilizá-la.

Portanto, os autores (NEPOMUCENO, 2020) destacam uma abordagem utilizando MDE para sistemas de *Internet of Things* orientada a usuário por meio do *framework* na Figura 17. O *framework* possui três objetivos específicos, primeiro modelos do cenário IoT, segundo gerar o código da aplicação ao lado do servidor e por último poder estender o código.

O processo de modelagem é definido conforme o metamodelo criado, definindo cenário, dispositivos, protocolos de comunicação e conexão com o banco de definição de dados.

Para começar, o processo de utilização do *framework* o usuário só precisa informar um arquivo em formato JSON para criar um cenário IoT. Depois dessa etapa o *framework* entrega os objetos para a geração do código da aplicação. Para a geração ao lado do servidor, é necessário builder para transformar M2T que gerar o código ao lado do servidor. E por último, a possibilidade de extensão de código, caso queiram adicionar outro requisito. E a aplicação estará pronta para ser utilizada.

E para avaliação do *framework* houve um experimento do *AutoIoT*, os desenvolvedores

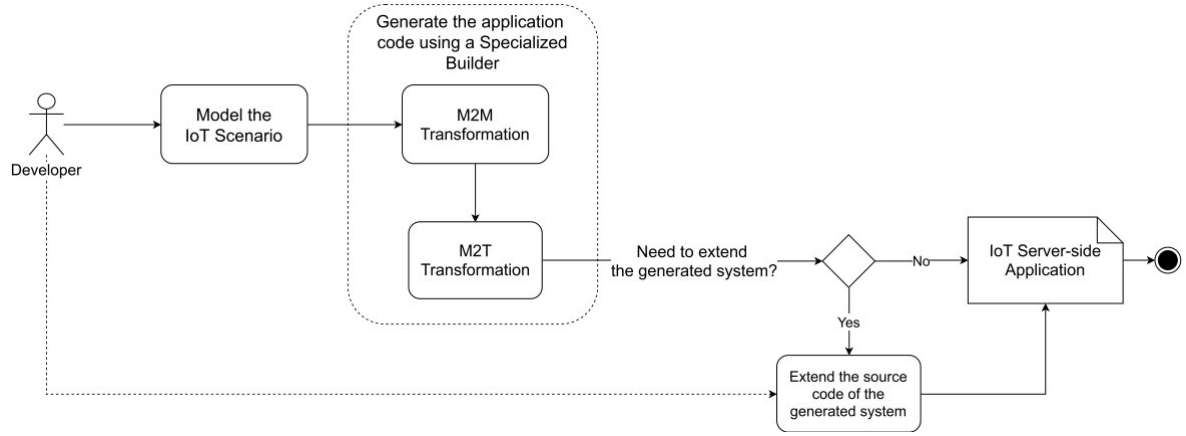

Figura 15 – *The developer starts the process by creating the model that represents the IoT scenario. Fonte:(NEPOMUCENO, 2020).*

de IoT e da Web aderiram bem à aplicação, mas somente 74% dos participantes entendiam os sistemas gerados. E por fim não é capaz de suprir todas as camadas da IoT se concentrado apenas na geração da aplicação ao lado do servidor.

3.4 Síntese

Neste capítulo, apresentou-se o estado da arte referente a *Web of Things* conforme W3C, abordagens MDE e propostas IoT. Nove trabalhos foram descritos com suas características, objetivos principais e ferramentas utilizadas.

4 *Framework*

Neste capítulo, pretende-se apresentar um *framework* para geração de *Thing Description* conforme a W3C e dispositivo IoT que se comunicam por meio do protocolo *WebSockets*. Esse *framework* é baseado em MDE e tem por objetivo proporcionar comunicação entre as propostas WoT e IoT.

Uma metodologia é proposta para detalhar as etapas de geração da *Thing Description* e IoT *device*. Fechando o capítulo, os metamodelos desenvolvidos para serem usados no *framework* são apresentados.

4.1 Um *Framework* para suportar WoT e IoT

Este *framework* é baseado em MDE, biblioteca *Node-WoT*, linguagem *JavaScript*, linguagem C, W3C *Web of Thing*, *Internet of Things* e *WebSockets*. Permitindo a criação de propostas *WoT Thing Description* conforme a W3C se comuniquem com IoT *devices* por meio *WebSockets*. A intenção é proporcionar a interoperabilidade entre às duas propostas e incentivar a utilização da WoT.

A Figura 18 demonstra o *framework* proposto, com seus modelos, metamodelos e o fluxo de transformações tanto de WoT e IoT. A primeira parte do *framework* constituida pelo desenvolvimento do WoT conforme descrito a seguir:

- Na primeira etapa, há a criação do PIM, que dá a possibilidade de gerar TD para outras plataformas de acordo com cada PSM. O modelo facilita a criação de TD conforme atuadores e sensores, como LED, temperatura, etc;
- Na segunda etapa, PSM *abstract* é gerado de acordo com o metamodelo W3C é gerado a partir de um PIM;
- Na terceira etapa, a transformação tem como entrada o PSM *abstract* onde ocorre a transformação para PSM *concrete*, com a inserção de detalhes de *JavaScript*, *Node-WoT* e *WebSockets*;
- Na quarta etapa, o mecanismo de transformação tem um PSM *concrete* como entrada, gerando o Código-fonte como saída;
- Na quinta etapa, o *framework* é responsável por criar o código-fonte na linguagem *JavaScript* (JS);

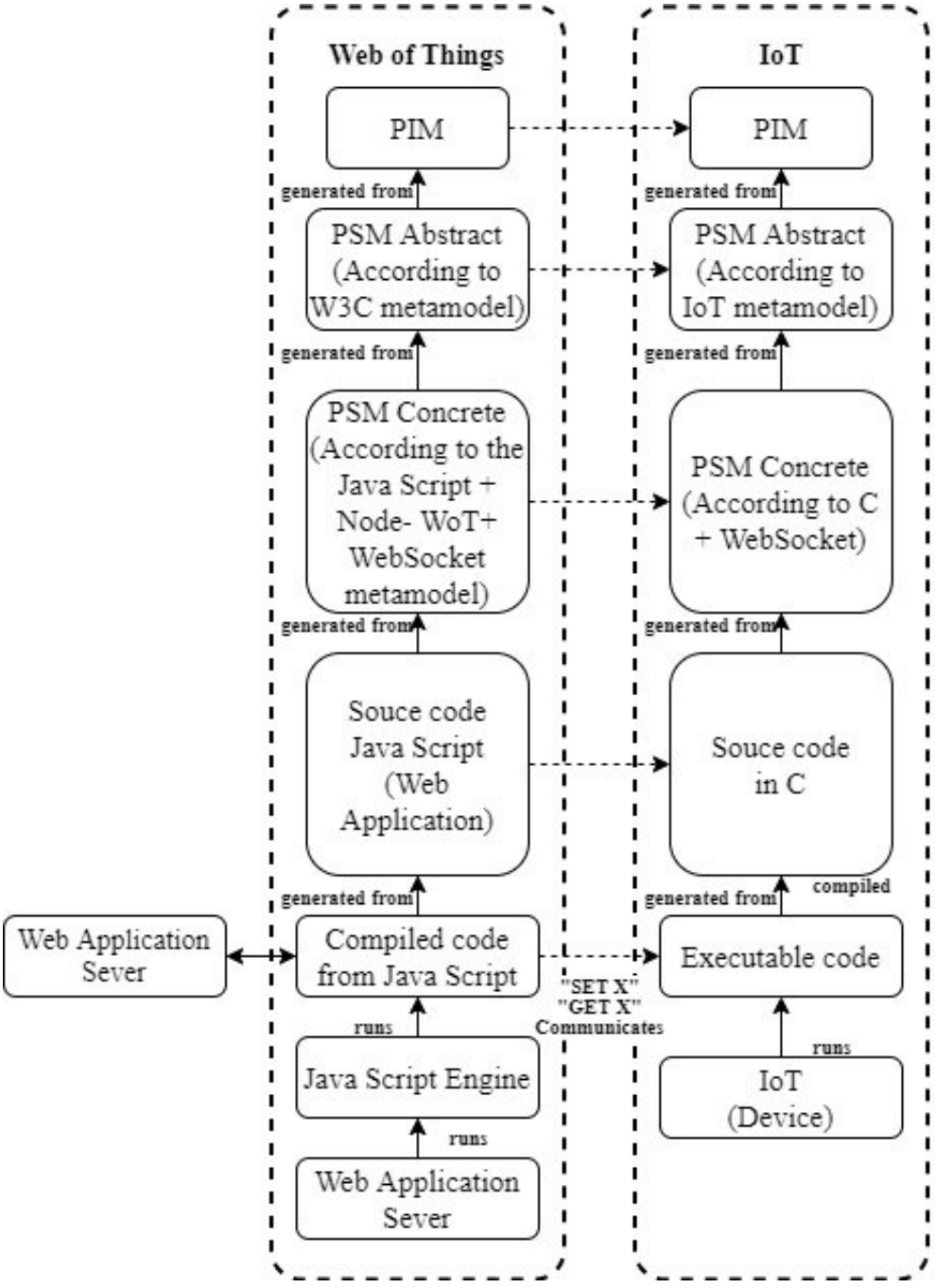

Figura 16 – Framework para a geração WoT *Thing Description* conforme a W3C se comuniquem com IoT devices por meio *WebSockets*. Fonte: Autor.

- Na sexta etapa, após a compilação do código, é necessário utilizar a *Web Application Server* em *Node.js* para que haja comunicação com o dispositivo IoT e para dar acesso a TD por meio de URIs.

A segunda parte do framework constituida pelo desenvolvimento do IoT conforme descrito a seguir:

- Na primeira etapa, há a criação do PIM, que dá a possibilidade de modelar IoT device para outras plataformas de acordo com cada PSM. O modelo dá a possibilidade de criar código de dispositivo conforme a placa *Esp8266* para sensores e atuadores físicos;
- Na segunda etapa, PSM *abstract* de acordo com o metamodelo IoT é gerado a partir de um PIM;
- Na terceira etapa, a transformação tem como entrada o PSM *abstract* onde ocorre a transformação para PSM *concrete*, com a inserção de detalhes da linguagem C e *WebSockets*;
- Na quarta etapa, o mecanismo de transformação tem um PSM *concrete* como entrada, gerando o Código-fonte como saída;
- Na quinta etapa, o *framework* é responsável por criar o código-fonte na linguagem C;
- Na sexta etapa, após a compilação do código, é necessário executar o código em uma placa *Esp8266*.

4.2 Metodologia para criação de Thing Description conforme a W3C

Uma metodologia criada para utilização de uma parte do *framework*. A criação da *Web of Thing* TD W3C proposto é apresentada na Figura 19, sendo composta pelas etapas seguintes:

- *Create PIM W3C*: Possui a lógica do negócio;
- *Definition of* TD: Define as propriedades necessárias para criar a TD, fazendo referência ao PIM;
- *Apply Transformation definition (M2M)*: Define aplicação de transformação de modelo para modelo;

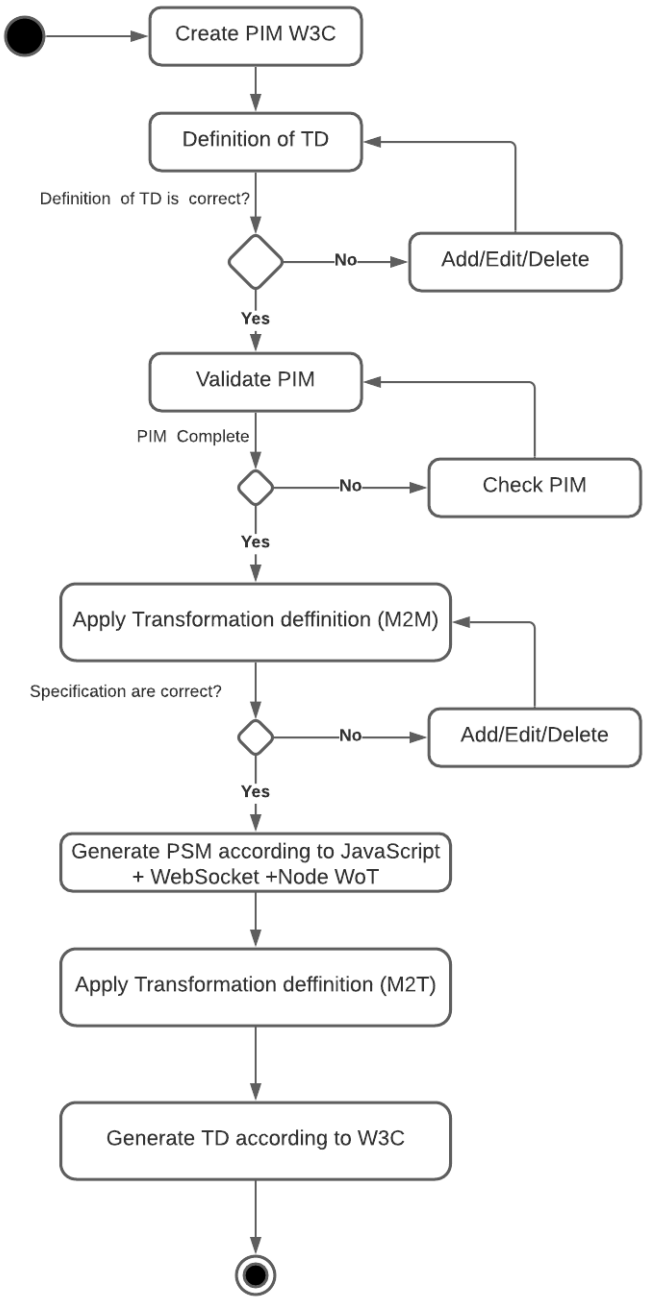

Figura 17 – Metodologia para geração de WoT Thing Description According W3C (baseado em (SILVA, 2020)).

- *Generate PSM according to JavaScript + WebSocket + Node-WoT*: Gera o PSM adicionando plataformas específicas para criar a TD;
- *Apply Transformation definition (M2T)*: Aplicação de transformação modelo para texto;
- *Generate TD according to W3C*: Gera o Código de acordo conforme a plataforma específica de TD.

4.3 Metodologia para criação do *software* do IoT *device*

Uma metodologia criada para utilização de uma parte do *framework*. IoT *device* proposto é apresentada na Figura 20, sendo composta pelas etapas seguintes:

- *Create PIM*: Possui a lógica do negócio;
- *Definition of IoT Device*: Define as propriedades necessárias para criação do IoT *device*;
- *Apply Transformation definition (M2M)*: Define aplicação de transformação de modelo para modelo;
- *Generate PSM according to C + WebSocket*: Gera o PSM acrescentando plataforma específica para criar a IoT *device*;
- *Apply Transformation definition (M2T)*: Aplicação de transformação modelo para texto;
- *Generate Code according to IoT device*: Gera o Código de acordo conforme a plataforma específica do IoT *device*.

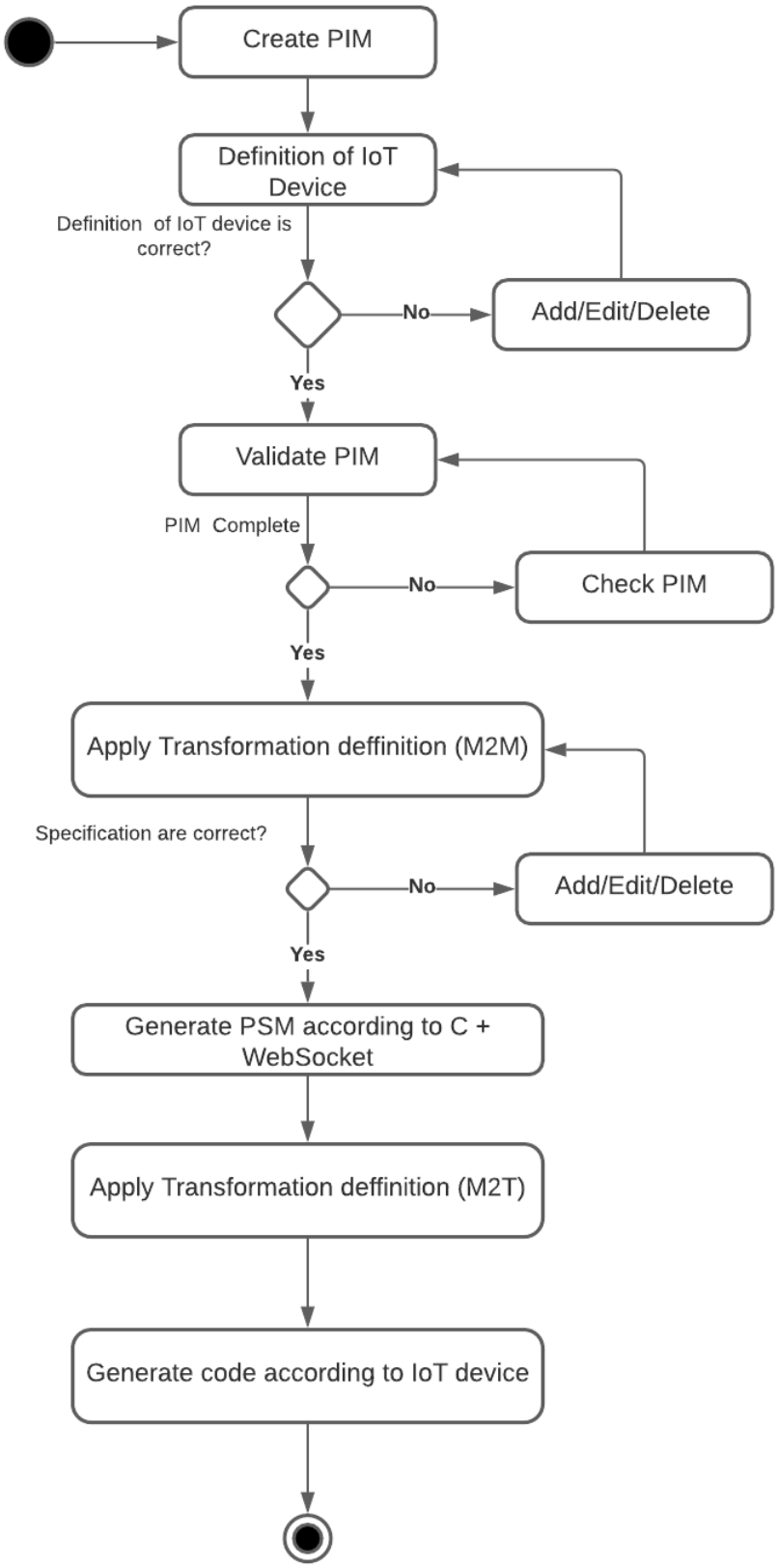

Figura 18 – Metodologia para geração de IoT device (baseado em (SILVA, 2020)

4.4 Metamodelos

Os metamodelos propostos são para garantir a utilização da Web of Things W3C e a utilização de IoT device para que os dois possam se comunicar.

4.4.1 Metamodelo *Thing Description* W3C

O metamodelo *Thing Description* foi criado baseado no modelo UML criado pela W3C (W3C, 2020i), portanto, conservamos as classes e as conexões. Para que todas às vezes que for modelar sensor ou um atuador utilize o metamodelo da Figura 21. Destacamos suas classes:

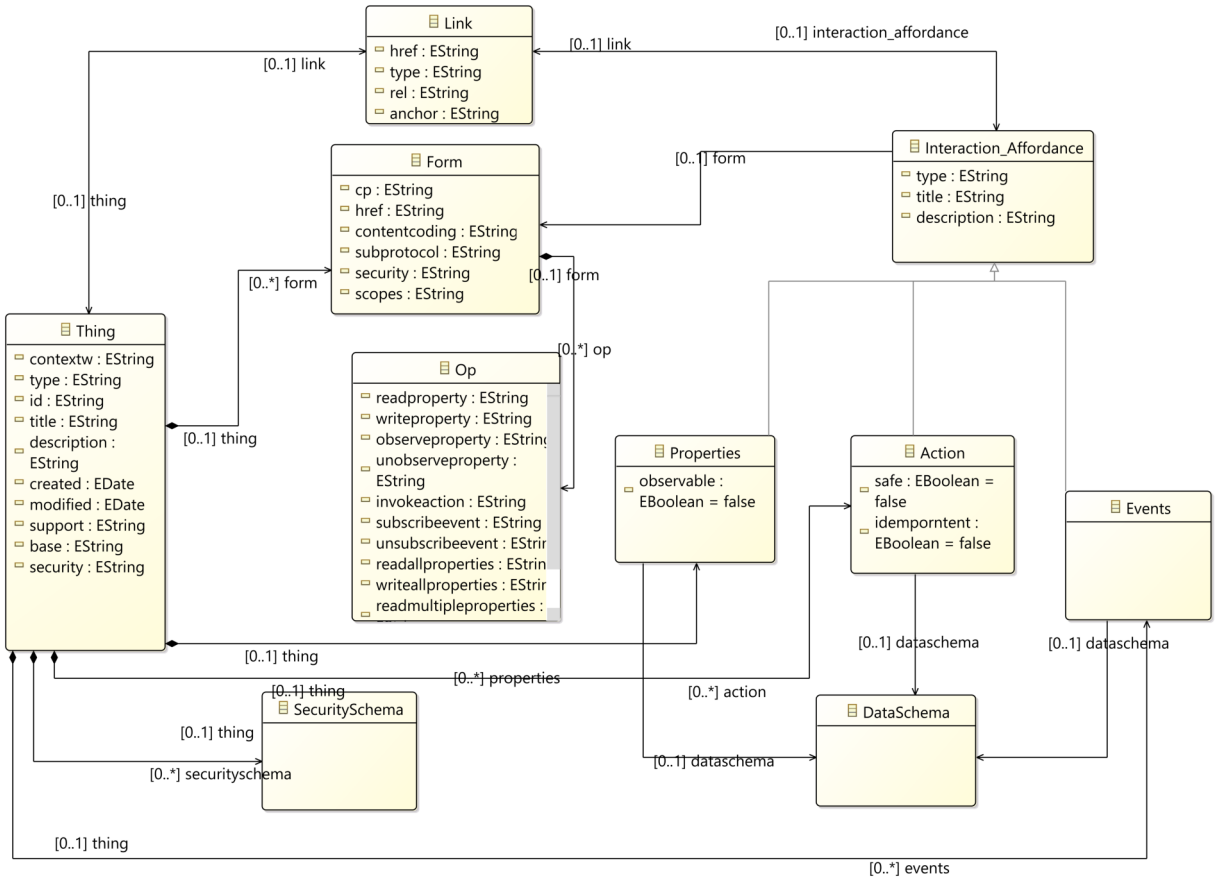

Figura 19 – Metamodelo Thing Description (baseado em (W3C, 2020i)).

- *Thing*: A parte principal do metamodelo, descreve um dispositivo físico ou virtual. Podendo ser sensores ou atuadores;
- *Link*: Promove a ligação entre a *Thing* e a *interaction affordances*;
- *Interaction Affordances*: é uma superclasse que acopla três classes herdeiras utilizadas para interagir com a *Thing*:

Properties: Expõe as características e estado da *Thing*.

Action: Expõe como manipular o estado da *Thing*.

Events: Qualquer evento que pode disparar conforme qualquer mudança de estado.

- *Forms*: Pode ser utilizado para controlar hipermídia na web, recursos e o estado da *Thing* como método ou destino de envio, na TD especificamente *properties, action e events*;
- *Op*: Descrição semântica de executar o formulário conforme a ação do cliente;
- *DataSchema*: Definição de metadata, especificamente *data types*;
- *Security*: Descreve a metadata de mecanismo de segurança.

4.4.2 Metamodelo *Node-WoT*

O metamodelo *Node-WoT* foi criado baseado na biblioteca *Node-WoT* (THINGWEB, 2020), que permite a criação da implementação da *WoT Scripting Api* utilizando a linguagem *JavaScript*. O metamodelo na Figura 22, é destacado pelas classes:

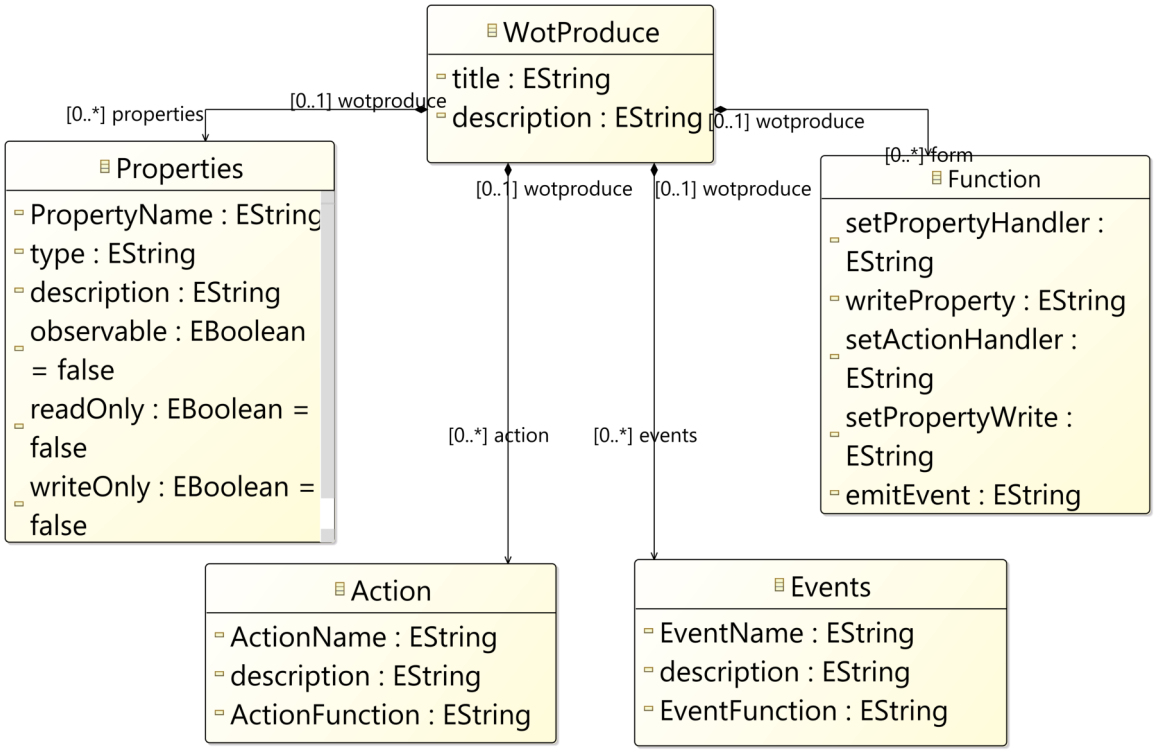

Figura 20 – Metamodelo Node-WoT (baseado em (THINGWEB, 2020)).

- *WotProduce*: É uma promise necessária para expor a *Thing Description*, possuindo o título da coisa e sua descrição conforme *Node-WoT*;

- *Properties*: Expõe as características e estado da *Thing*, muito parecido com as propriedades do WoT, se diferenciando pelo *readOnly* ou *writeOnly* deixado bem claro se a propriedade pode ser lida, ou alterada;
- *Actions*: Expõe como manipular o estado da *Thing*;
- *Events*: Qualquer evento que pode disparar conforme qualquer mudança de estado;
- *Forms*: Funções utilizadas para consumir e alterar propriedade especificamente *properties, action e events.*

4.4.3 Metamodelo IoT *Device*

O metamodelo IoT *Device* foi criado baseado em uma parte do cenário de dispositivos físicos da IoT, que permite a criação e implementação de dispositivo físico em placas *Esp8266, Arduino Uno, Raspberry Pi* e outros. O metamodelo na Figura 23, é destacado pelas classes:

- *Thing*: Dispositivo que possui conexão com a *internet*;

  SmartObject: Em caso de precisar adicionar objetos inteligentes que se comuniquem com dispositivo IoT;

  ConectecPhysicalEntinit: Entidade Física;
- *IotAplication*: Aplicação completa da IoT, para que futuramente possa comportar mais itens;
- *Component*: Uma classe, responsável por conter a propriedade em comum com sensores e atuadores;
- *Controler*: Tipo de placa utilizado;
- Pin: O número do pin na placa;
- *Comunication*: Tipo de comunicação da placa;

  WebSocket: Protocolo de comunicação;
- *TypeController*: Tipos de placas que podem ser selecionados.

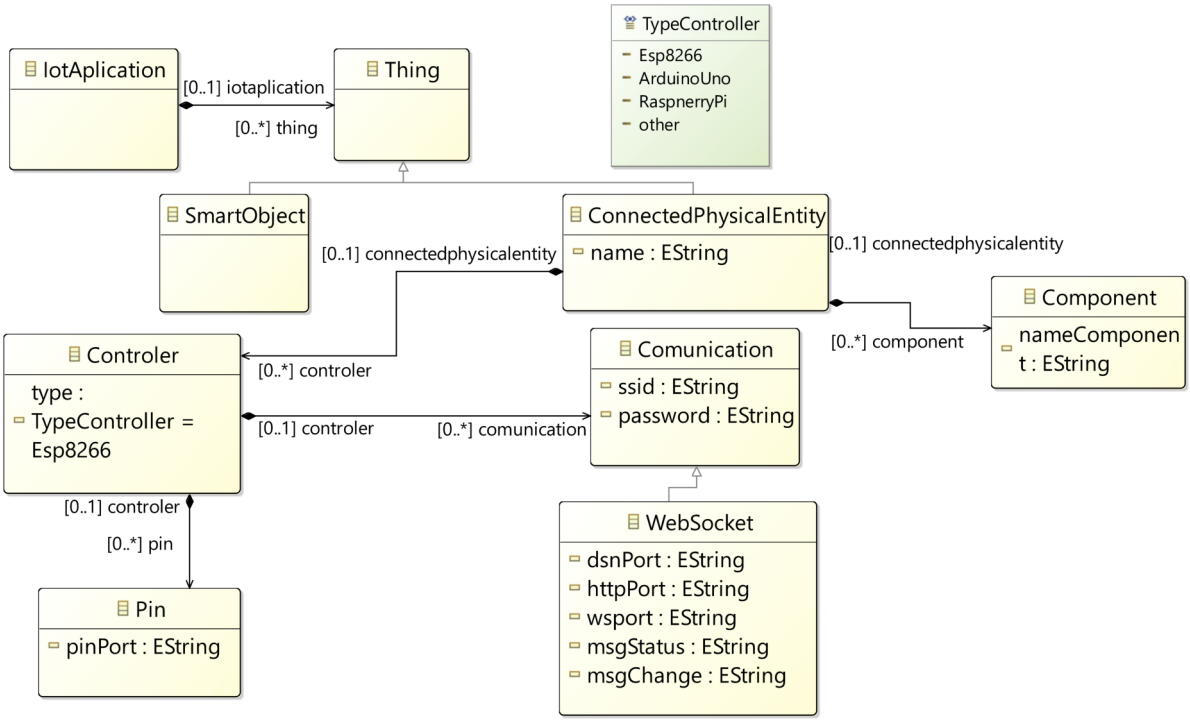

Figura 21 – Metamodelo IoT device (baseado em (ALULEMA D. ; CRIADO, 2019)).

4.4.4 Metamodelo C

O metamodelo C foi criado baseado na linguagem C e no protocolo de comunicação WebSockets, que permite a criação e a implementação do código em C. O metamodelo na Figura 24, é destacado pelas classes:

- Programa: Este elemento que compõe toda a linguagem C;
- *Main*: Ponto de partida para execução do programa;
- *TypeC*: A superclasse que possui os tipos de variáveis na linguagem C;

  PrimitiveType: Tipos primitivos como *char, int, float, double e void*;

  Struct: É uma coleção de variáveis relacionadas;
- *Function*: representam as funções que programa poderá ter;
- *WSsocket*: Protocolo websocket de comunicação utilizado.

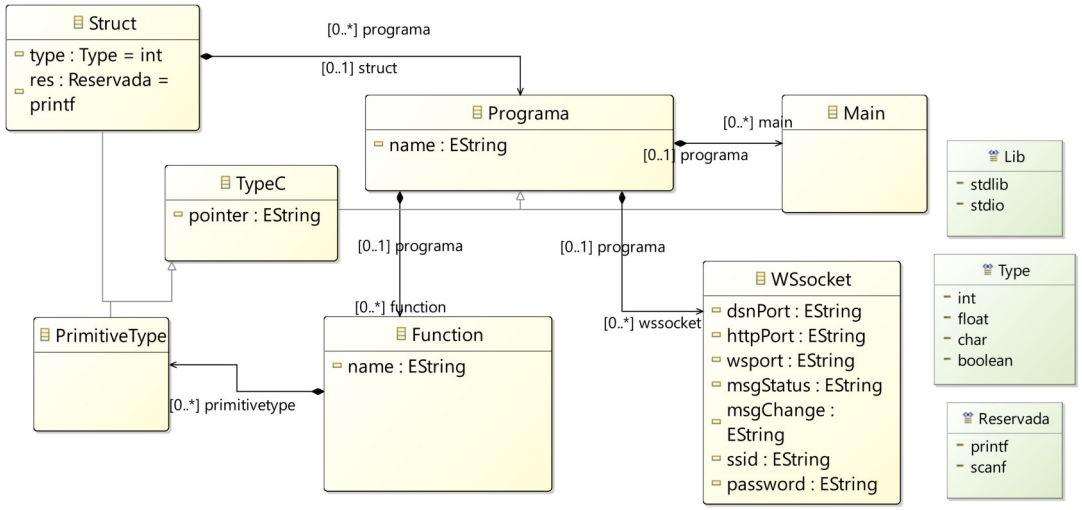

Figura 22 – Metamodelo Linguagem C. Fonte: Autor.

4.5 Definição de transformação

As definições de transformações foram aplicadas tanto para *Web of Things* e *Internet of Things*.

4.5.1 Definição de transformação para *Web of Things Thing Description*

As regras de transformação aplicadas na primeira parte do *framework* WoT. A primeira transformação é de *Thing Description* para *Node-WoT* e *WebSockets*. E após as definições de transformação a geração do código *Thing Description*.

A Figura 25 apresenta uma parte da regra de transformação conforme o metamodelo *Thing Description* e metamodelo *Node-WoT*. A Figura 26 apresenta um fragmento da transformação com *WebSockets*. Após as transformações, e há geração do código da *Thing Description* na linguagem *JavaScript* possuindo a estrutura da biblioteca *Node-WoT*, com o protocolo *WebSockets* para se comunicar com IoT *device*. As transformações foram feitas utilizando ATL. As demais transformações estarão em anexo.

```
-- @nsURI W3C=http://www.example.org/thingDescription_2
-- @nsURI NODE=http://www.example.org/node_Wot

module REGRAW3C;
create OUT : NODE from IN : W3C;


rule Thing2WotProduce{
from e:W3C!Thing
to out:NODE!WotProduce(
    title<-e.title,
    description<-e.description

)
    }

rule Properties2Properties{
from e:W3C!Properties
to out:NODE!Properties(
    PropertyName<-e.title,
    type<-e.type,
    description<-e.description,
    wotproduce<-e.thing
)
    }

rule Events2Events{
from e:W3C!Events
to out:NODE!Events(
    EventName<-e.title,
    description<-e.description,
    wotproduce<-e.thing
)
    }
```

Figura 23 – Definição de transformação em ATL de *Thing Description* para *Node-WoT*(transformação modelo-à-modelo). Fonte: Autor.

```
library Biblioteca;

helper context NODE!WotProduce def: toString(): String =
     'var result
     Wot.Produce({' + '\n' +
        '\t' + 'title : "' + self.title +'",' +'\n' +
        '\t' + 'description:"' + self.description + '",' + '\n' +
    '\t' +'\t' +    self.properties -> iterate(i; acc : String = '' |
            acc + i.toString()) +
            '\n'  +

    '\t' +'\t' + self.action-> iterate(i; acc : String = '' |
            acc + i.toString()) +
            '\n'  +

    '\t' +'\t' + self.events-> iterate(i; acc : String = '' |
            acc + i.toString()) +
            '\n'+
                '})\n' +
    -- getThingDescription
    '\t'  + '.then (function(thing) {' + '\n' +
            'console.log("Produced "+ thing.getThingDescription().title);' + '\n' +
    -- WriteProperty
            'thing.'+

            self.form->collect(e | e.writeProperty)->
            iterate(i; acc : String = '' |
            if i='true' then
            'writeProperty' else 'erro' endif
            )
            + '("' +
        self.properties->collect(e | e.PropertyName)
         ->iterate(i; acc : String = '' |
         acc + i.toString()) +
                    '",get'+
```

Figura 24 – Definição de transformação em ATL de *Thing Description* para código (transformação modelo-à-codigo). Fonte: Autor.

4.5.2 Definição de transformação para *Internet of Things IoT*

As regras de transformação aplicadas na segunda parte do *framework* IoT. A primeira transformação, IoT *device* para C e *WebSockets*, e após as definições de transformação a geração do código na linguagem C.

A Figura 27 apresenta uma parte da regra de transformação conforme o metamodelo IoT *device* e C. A Figura 28 apresenta um fragmento da transformação com *WebSockets*. Após as transformações há a geração do código do IoT device possuindo o protocolo de comunicação *WebSocket* para se comunicar com WoT. As transformações também foram feitas utilizando ATL. As demais transformações estarão em anexo.

```
module REGRAIOTDEVICE2C;
create OUT : C from IN : IOT;

rule Connect2Programa{
from e:IOT!ConnectedPhysicalEntity
to out:C!Programa(
    name<-e.name
)
    }
rule Websocket2WSsocket{
from e:IOT!WebSocket
to out:C!WSsocket(
    dsnPort<-e.dsnPort,
    httpPort<-e.httpPort,
    wsport<-e.wsport,
    ssid <- e.ssid,
    password <- e.password,
    msgChange <- e.msgChange,
    msgStatus <- e.msgStatus,
    programa<-e.controler.connectedphysicalentity
)
    }

rule Pin2Main{
from e:IOT!Pin
to out:C!Main(
    name<-e.pinPort,
    programa<-e.controler.connectedphysicalentity
)
    }
rule Component2Function{
from e:IOT!Component
to out:C!Function(
    name<-e.nameComponent,
    programa <- e.connectedphysicalentity

)
```

Figura 25 – Definição de transformação em ATL IoT device para linguagem C (transformação modelo-à-modelo). Fonte: Autor.

```
library BibliotecaC;


helper context C!Programa def: toString() : String =
    '#include<Wifi.h>'+ '\n'+
    '#include<ESP8266WiFi.h>'+ '\n' +
    '#include<FS.h>'+ '\n'+
    '#include<ESPAsyncWebServer.h>'+ '\n'+
    '#include<WebSocketsServer.h>'+ '\n'
     +
     self.main ->iterate(i; acc : String = '' |
            acc + i.toString()) + '\n'+

     --Conexão com websocket
    self.wssocket -> iterate(i; acc : String = '' |
            acc + i.toString()) +
            '\n'  +
    'AsyncWebServer server('+
        self.wssocket->collect(e | e.httpPort)
         ->iterate(i; acc : String = '' |
         acc + i.toString()) + ');\n'+
         'char msg_buf[10]; '+


    -- Iniciar a porta do websocket
    'WebSocketsServer webSocket = WebSocketsServer(' +

        self.wssocket->collect(e | e.wsport)
         ->iterate(i; acc : String = '' |
         acc + i.toString()) + ');\n'+

'//Call and Recive any WebSocket Message\n' +
'void onWebSocketEvent(uint8_t client_num,
                       WStype_t type,
                       uint8_t * payload,
                       size_t length) {'+

--Funções de WebSocket
```

Figura 26 – Definição de transformação em ATL de IoT device para código (transformação modelo-à-codigo). Fonte: Autor.

4.6 Síntese

Nesta seção, um *framework* para geração de casos de teste de desempenho e eficiência energética para aplicações voltadas para dispositivos móveis foi apresentado.

Uma metodologia para aplicação do *framework* utilizando a MDE e os metamodelos, a prototipagem e as definições de transformações utilizadas também foram apresentados nesta seção.

5 Prototipagem

Neste capítulo, tem-se como objetivo apresentar uma prototipagem do processo de utilização do *framework* para geração de *Thing Description* conforme a W3C e dispositivo IoT utilizando a transformação entre modelos por meio das ferramentas EMF, *EcoreTolols* e a linguagem ATL. Com principal objetivo de simplificar a utilização dos *framework*.

5.1 *Softwares* utilizados

Os *softwares* empregados para implementação do *framework* proposto são os seguintes:

- ATL/ EMFTVM para Eclipse versão 4.2.1 (ATL, 2020);
- Eclipse versão 2019-09;
- *ECORETOOLS: Ecore Diagram Editor*, versão 4.0 (ECLIPSE, 2020b);
- *EMF Compare: model and merge versão 3.3* (ECLIPSE, 2020a).

Eclipse foi utilizado como ambiente de desenvolvimento integrado (IDE) para suportar a criação e transformação do *framework*. EMF em conjunto com *EcoreTools* fazem a transformação entre modelos. A lingugaem ATL, foi utilizada para a transformação entre modelos e para a inserção de *helpers* para transformação.

5.1.1 Implementação da parte *Web of Things Framework*

O protótipo foi desenvolvido com um conjunto de *plug-ins* para *IDE-Eclipse*. Para a utilização do *framework*, é necessário os metamodelos *Thing Description* e *Node-WoT* para as transformações. O framework não implementa a parte do PIM, pois a prototipagem só é realizada a partir do PSM Abstract.

O primeiro passo para utilizar o *framework* conforme a Figura 29, Figura 30, Figura 31 e Figura 32 é a validação e geração de *plug-ins* para a criação de uma nova instância para ocorrer as transformações em ATL. O segundo passo é abrir uma nova instância do Eclipse. Para a utilização como podemos perceber no Modelo do metamodelo *Thing Description* Figura 31 e demonstrado em formato de árvore. Este PSM *abstract* foi gerado a partir do modelo do metamodelo *Thing Description* preenchido conforme a Figura 34.

O terceiro passo, a transformação de PSM *abstract* para PSM *concrete* e PSM *Concrete* em Código, com as regras de uso de *helpers* inseridas na biblioteca ws para

comunicação com qualquer dispositivo IoT conforme a Figura 35, para que por fim ocorra a geração de código. Ficando de responsabilidade do programador especificar os atributos ws como porta, IP endereço, etc. Assim finalizando o uso do *framework* para *Web of things*.

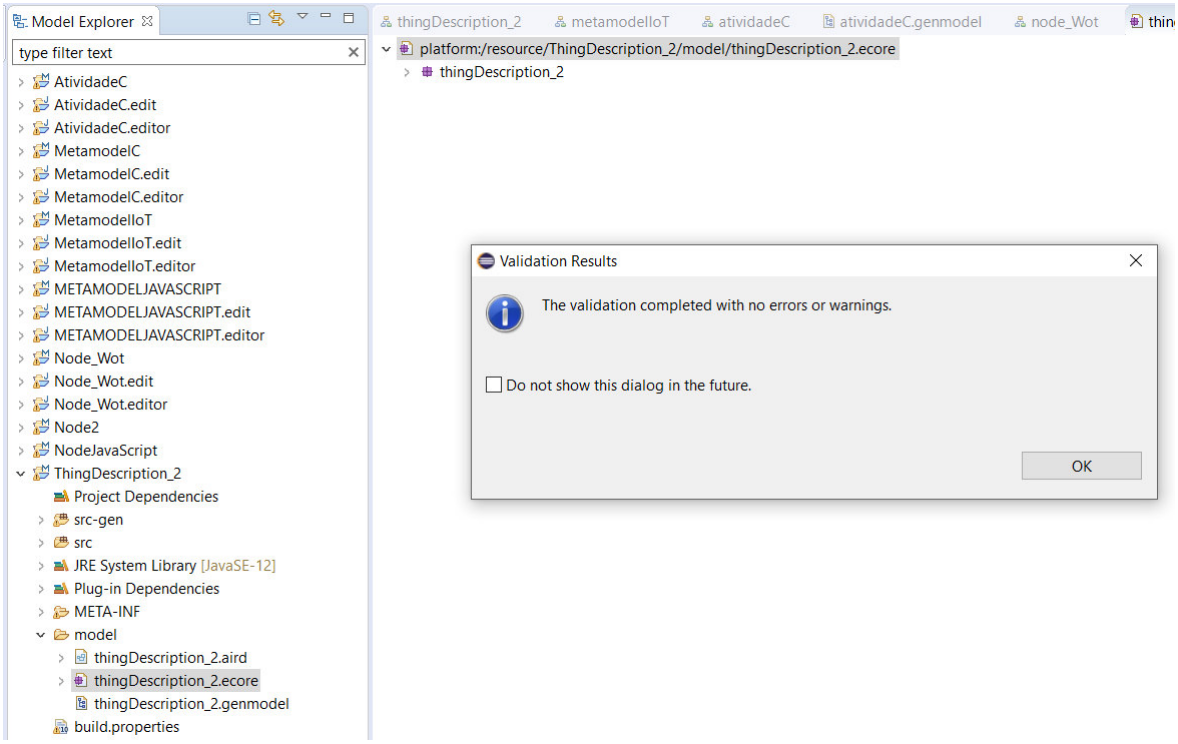

Figura 27 – Validação para a geração de Thing Description. Fonte: Autor.

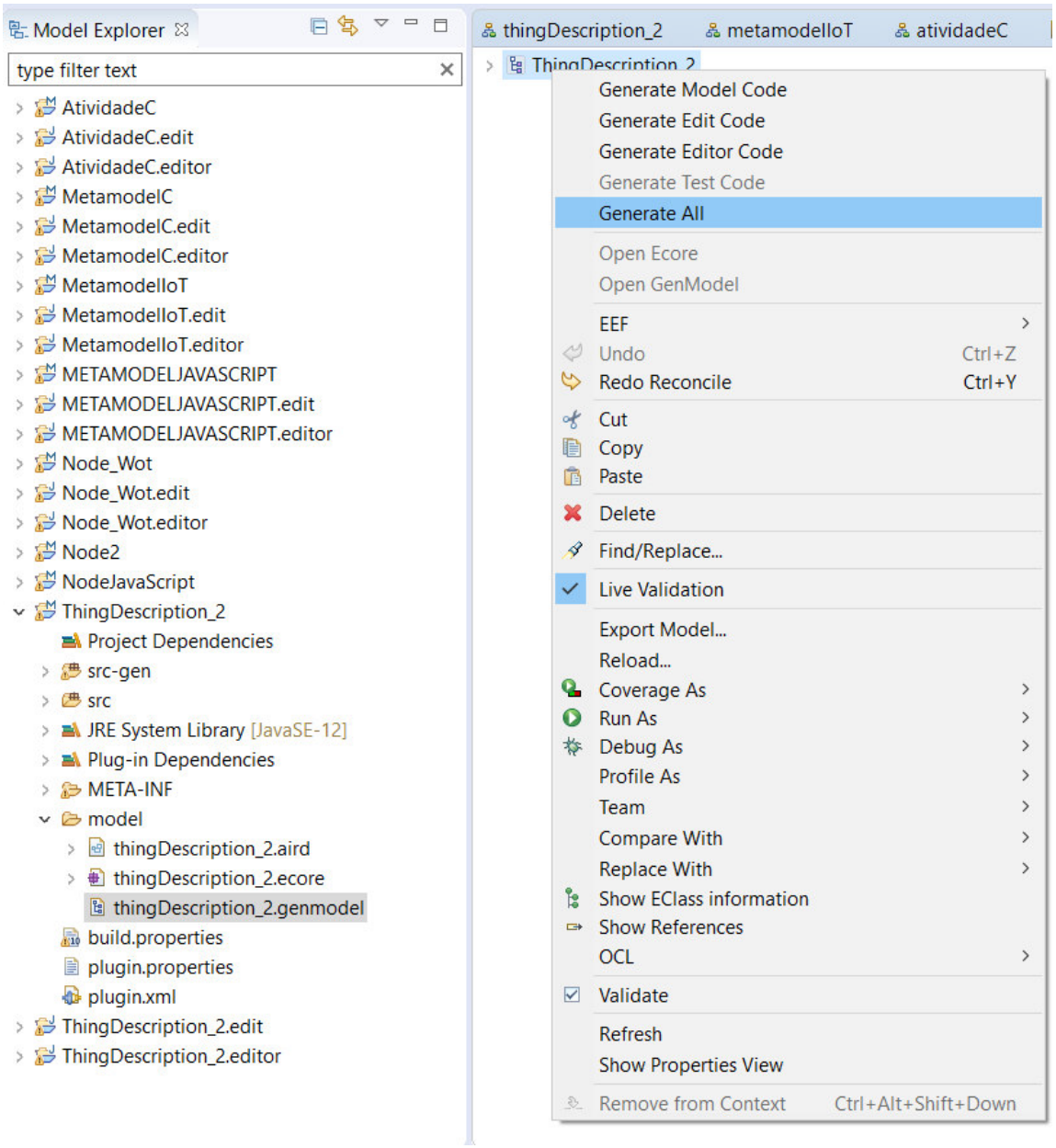

Figura 28 – Geração de pluging para de Thing Description. Fonte: Autor.

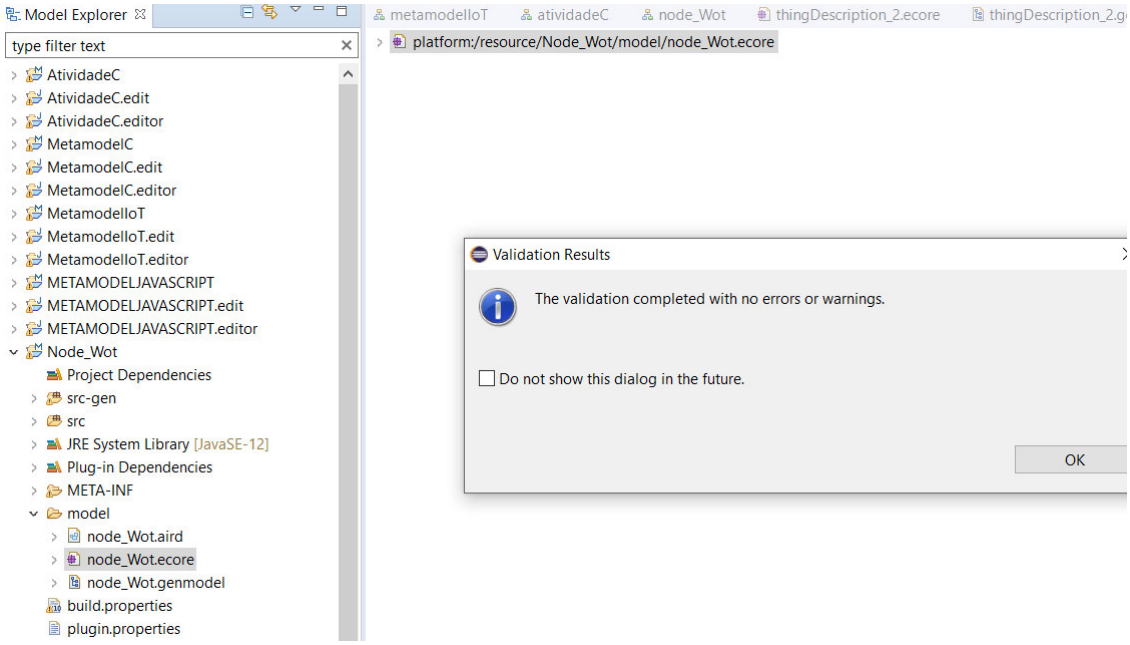

Figura 29 – Validação para geração da biblioteca Node-WoT. Fonte: Autor.

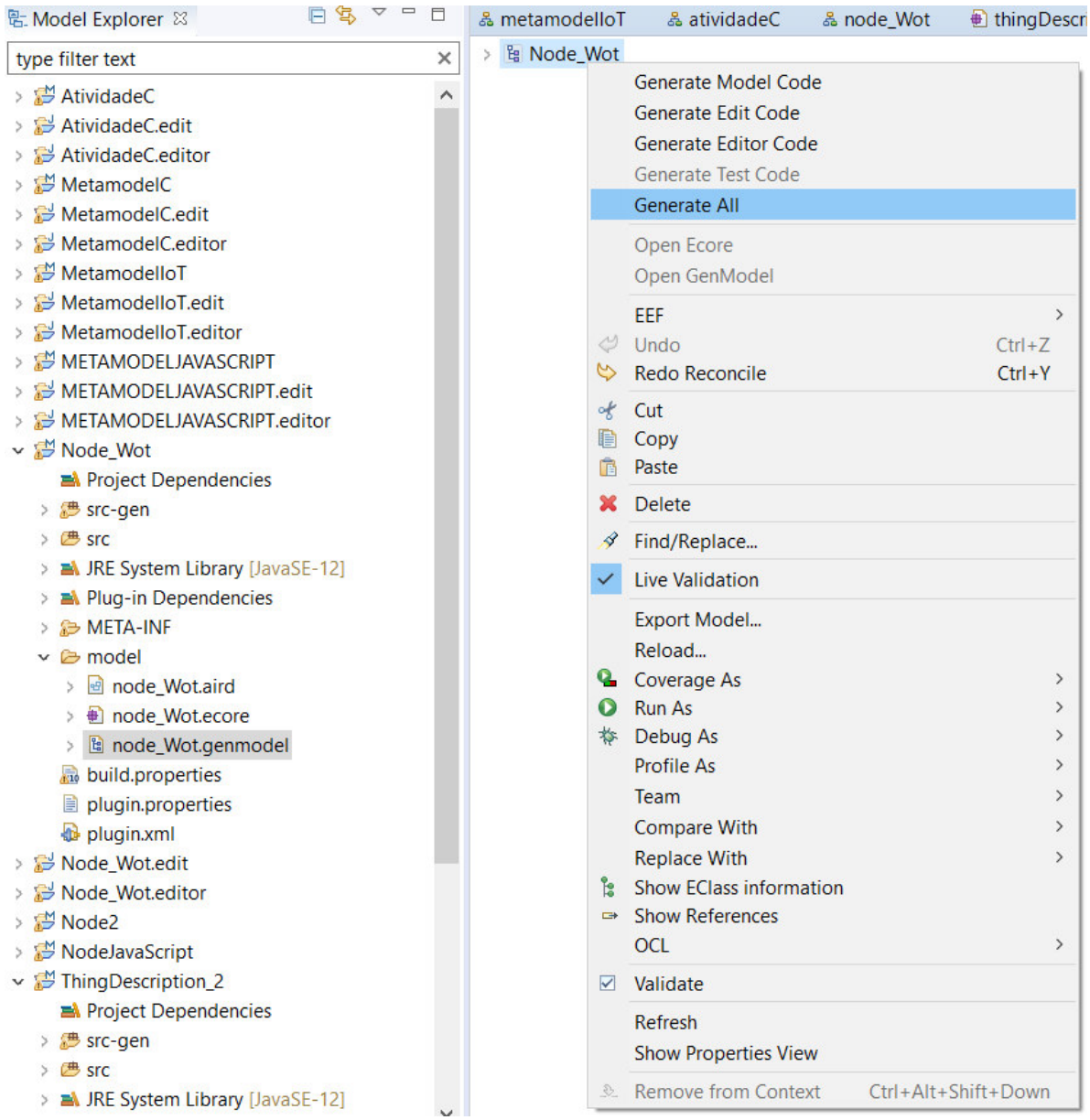

Figura 30 – Geração de pluging para biblioteca Node-WoT. Fonte: Autor.

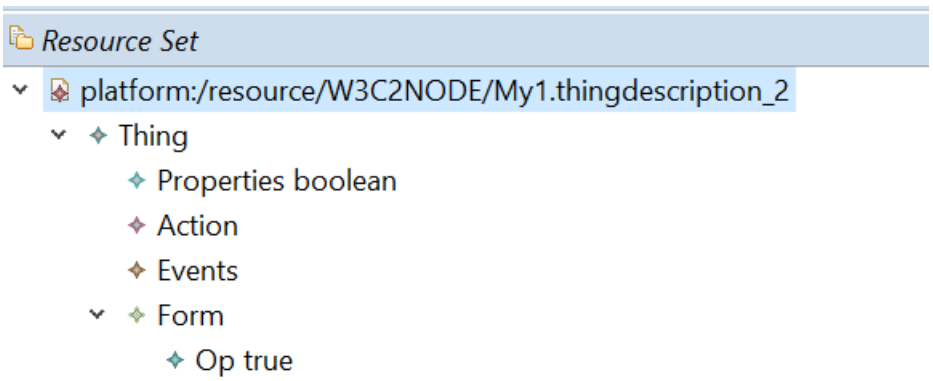

Figura 31 – Modelo do metamodelo Thing Description. Fonte: Autor.

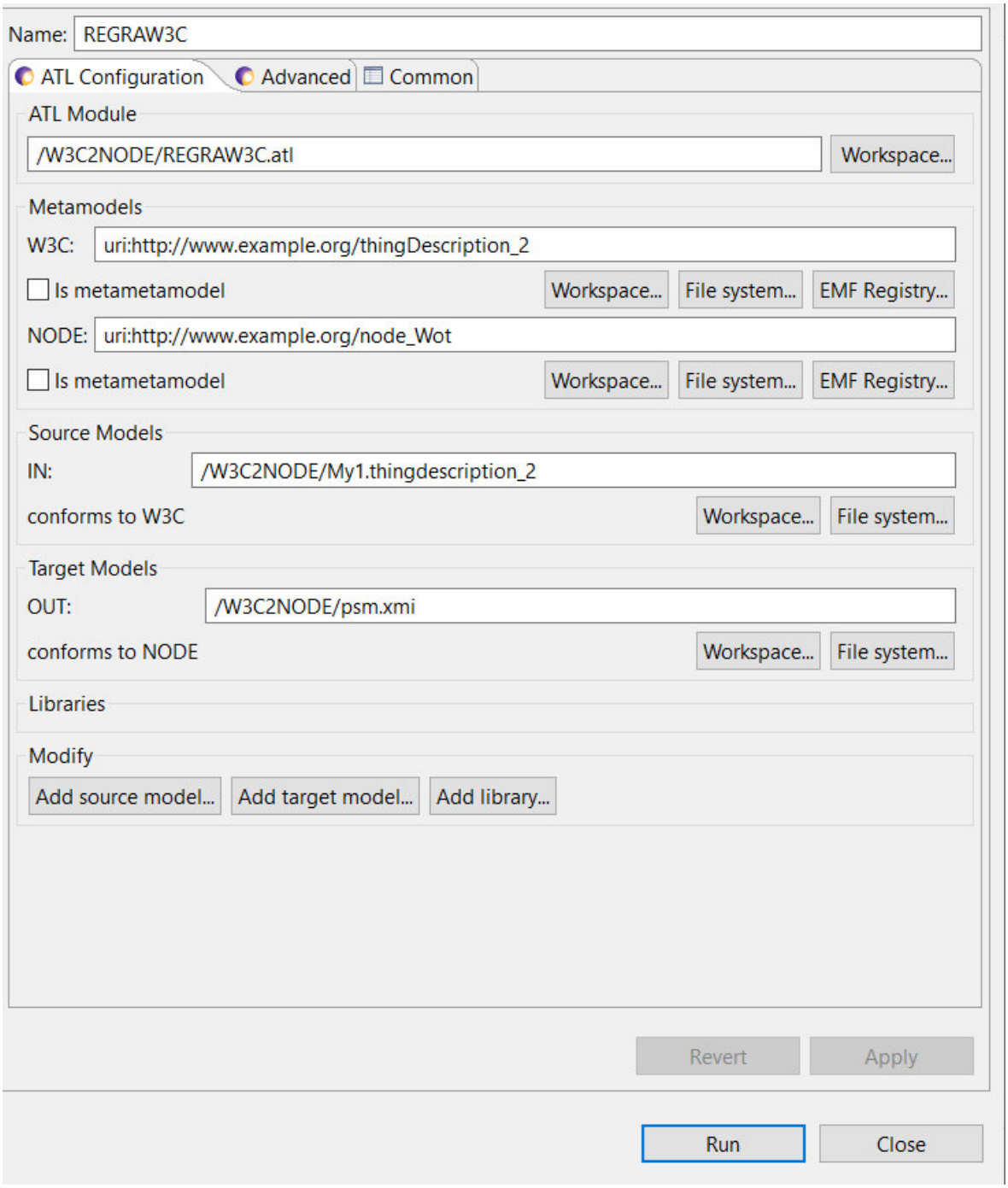

Figura 32 – Configuração de definição de transformação PSM Abstract. Fonte: Autor.

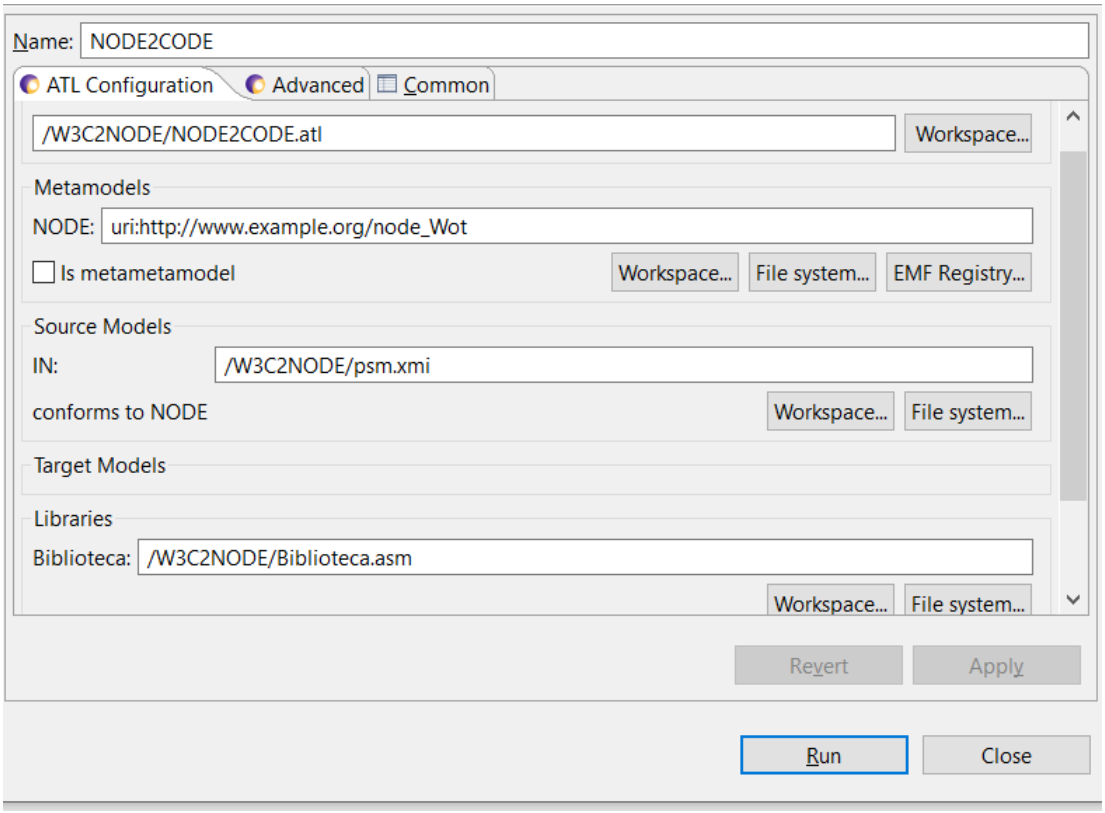

Figura 33 – Configuração de definição de transformação PSM Abstract para PSM Concrete para código. Fonte: Autor.

5.1.2 Implementação da parte *Internet of Things Framework*

O segundo passo para utilizar o *framework* para *Internet of Things* conforme os metamodelos IoT *device* e linguagem C são as transformações ilustradas na Figura 36, Figura 37, Figura 38 e Figura 39 a validação possuindo a estrutura da biblioteca WS e geração de *plug-ins* para a criação de uma nova instância para ocorrer as transformações em ATL. O segundo passo é abrir uma nova instância do Eclipse. Para a utilização do modelo do metamodelo *Thing* IoT *Device*, Figura 40 podendo ser visualizado em formato de árvore. Este PSM *abstract* foi gerado a partir do preenchimento do modelo do metamodelo IoT *device* conforme a Figura 41.

O terceiro passo, a transformação de PSM *abstract* para PSM *concrete* e PSM Concrete em Código, com as regras de uso de *helpers* inseridas na biblioteca ws para comunicação com qualquer *Thing Description* conforme a Figura 42, para que por fim ocorre a geração de código. Ficando de responsabilidade do programador especificar os atributos ws, e especificação da placa como porta, pin, etc. Assim finalizando o uso do *framework* para WoT e IoT.

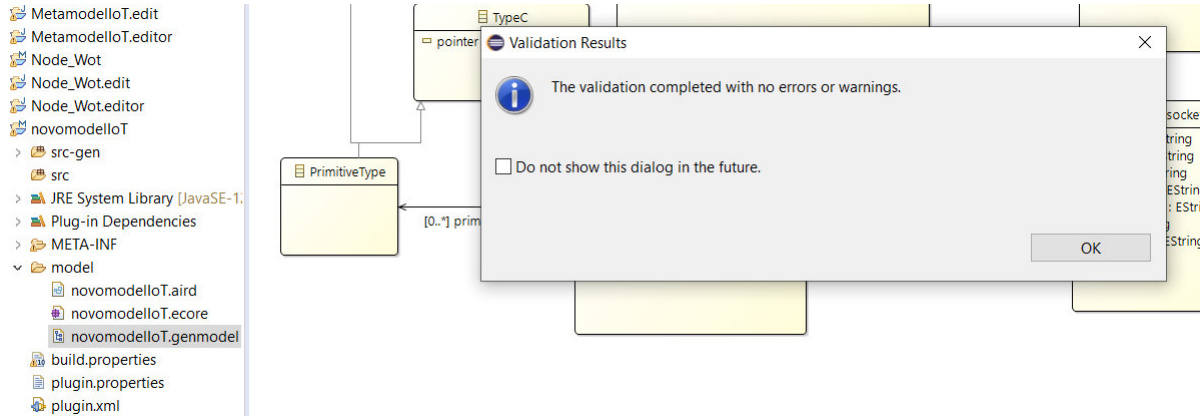

Figura 34 – Validação para a geração de IoT Device. Fonte: Autor.

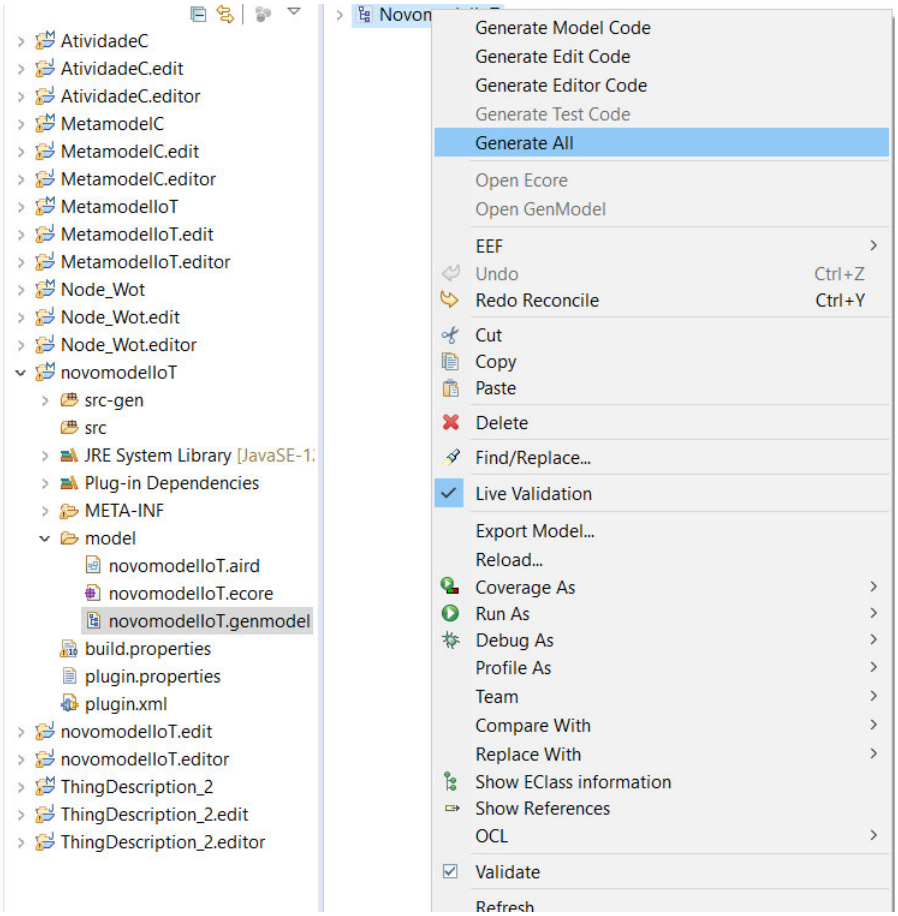

Figura 35 – Geração de plugins para de IoT Device. Fonte: Autor.

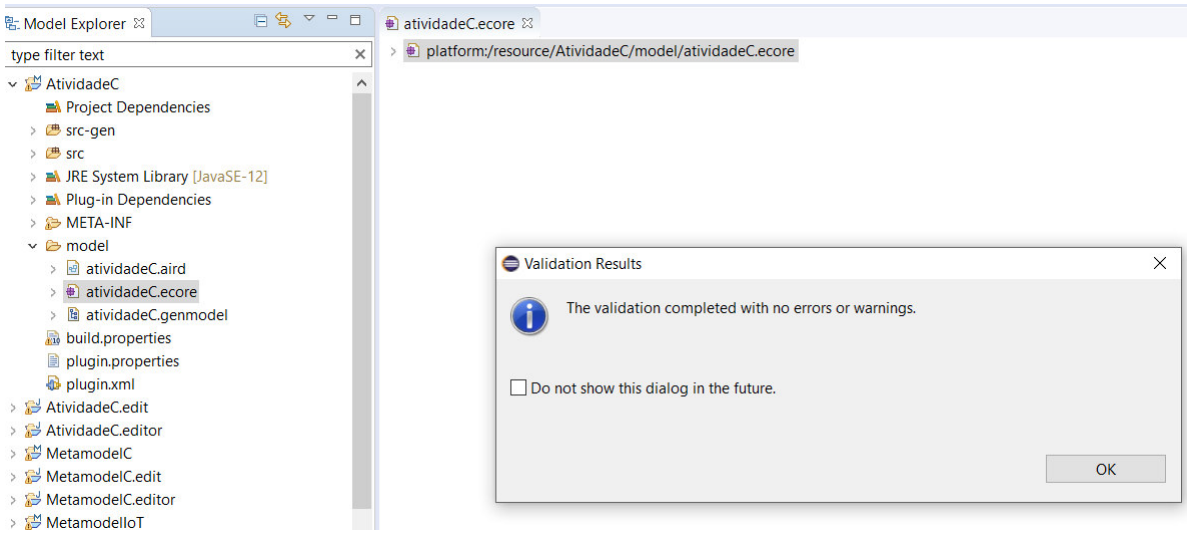

Figura 36 – Validação para a geração de linguagem C. Fonte: Autor.

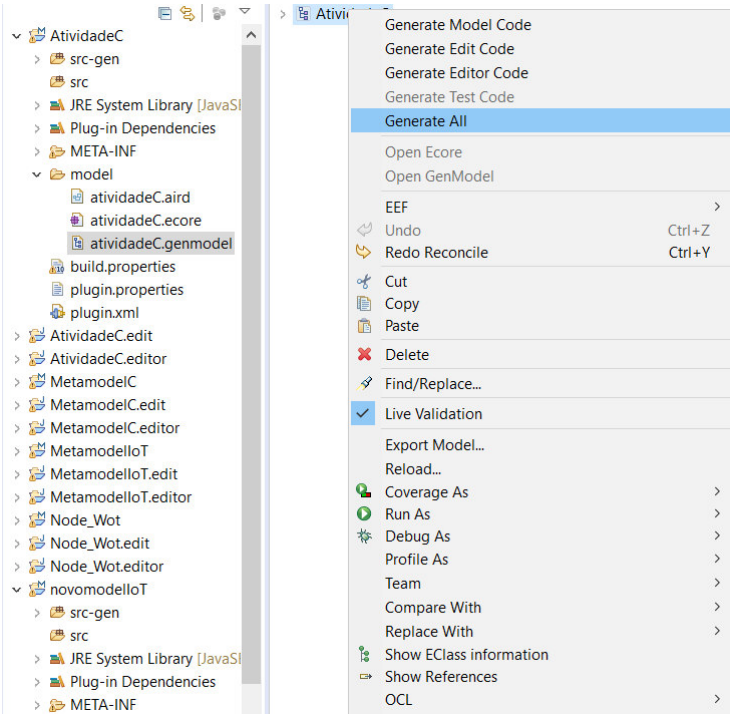

Figura 37 – Geração de plugins para linguagem C. Fonte: Autor.

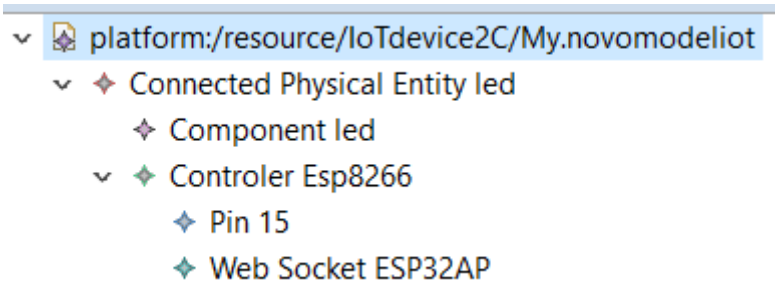

Figura 38 – Modelo do metamodelo IoT *Device*. Fonte: Autor.

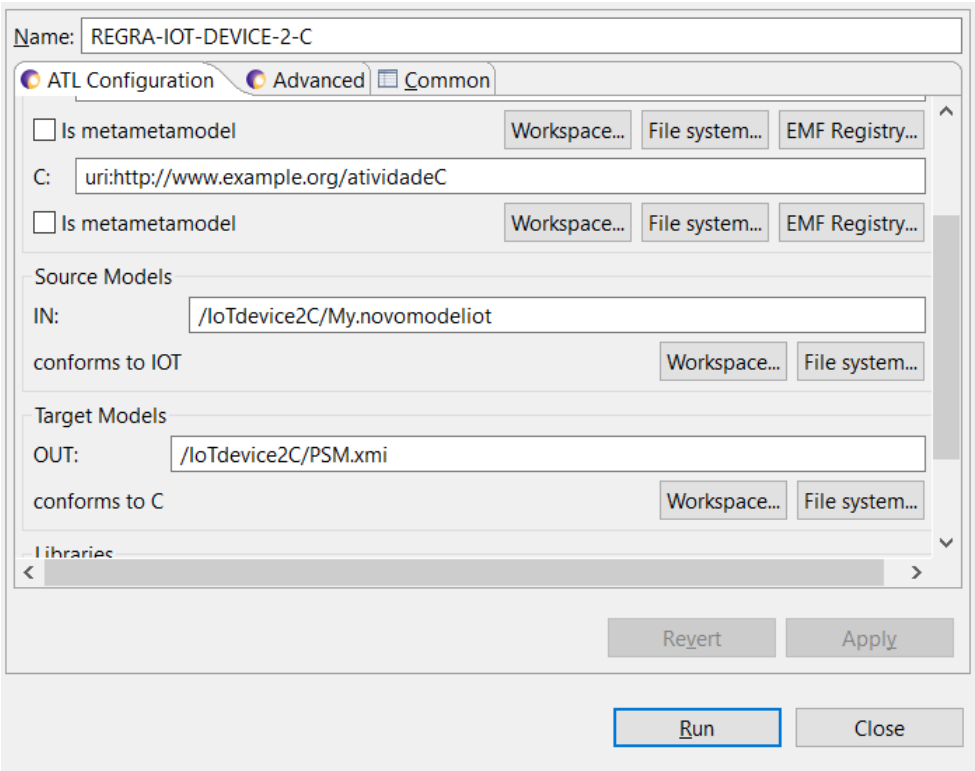

Figura 39 – Configuração de definição de transformação PSM Abstract. Fonte: Autor.

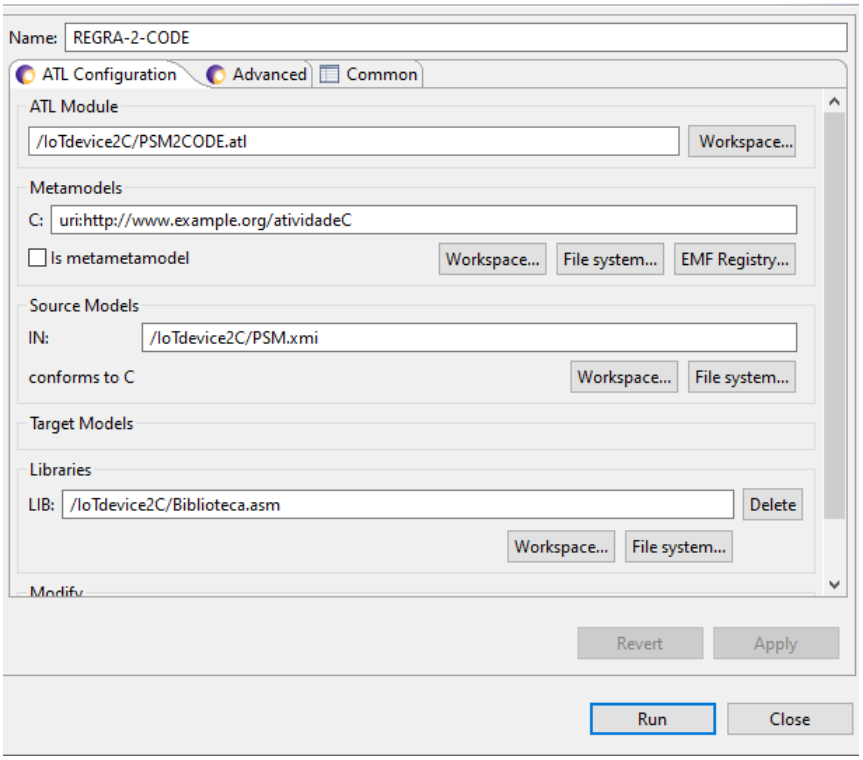

Figura 40 – Configuração de definição de transformação PSM Abstract para PSM Concrete para código. Fonte: Autor.

5.2 Síntese

Neste capítulo, apresentou-se a prototipagem do framework para a utilização e geração de W3C *Thing Description* e IoT *device*. Necessários para aplicação de exemplos no próximo capítulo.

6 Exemplos Ilustrativos

Neste capítulo, três exemplos ilustrativos são apresentados para ilustrar a utilização do *framework* proposto e demonstrar as funcionalidades por ele oferecidas. As aplicações demonstradas foram executadas em Node Js onde a biblioteca *Node-WoT* disponibiliza uma URI de acesso para todas as *Things*.

O primeiro exemplo consiste em ser mais descritivo contendo a demonstração de uma *Thing Description* de um LED, se comunicando via *WebSocket* para o acionamento do LED em uma placa *Esp8266* e a TD em execução pelo servidor *Node JS*. O segundo exemplo é uma Thing Description de sensor de temperatura se comunicando via *WebSocket* com um sensor de temperatura em uma placa *Esp8266*. O terceiro e último, uma *Thing Description* de um sensor de presença se comunicando via *WebSocket* com Pir em uma placa *Esp8266*.

Os três exemplos destacados foram escolhidos para ilustrar cenários que utilizam sensores e atuadores como Smart Homes e Smart Cities.

6.1 Descrição do exemplo ilustrativo acionamento de LED

Por meio do *framework* na parte de WoT conforme a Figura 43 demonstra a criação do modelo de negócio para construção de uma *Thing Description* de um LED. Este modelo foi criado pelo metamodelo Thing Description, podendo preencher todas as classes principais como *Thing, properties, action, events* e os *forms*.

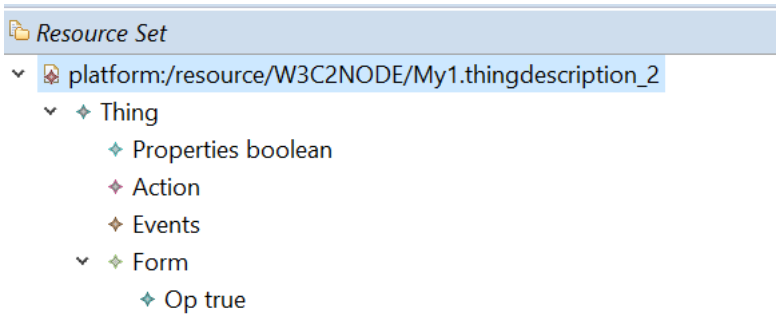

Figura 41 – Modelo Thing Description para um LED. Fonte: Autor.

A partir desse momento há geração das transformações feitas pelo *framework*, nas transformações para PSM em formato de XMI para definição de transformação com o metamodelo *Node-WoT* conforme a linguagem JS resultando em PSM *concrete*.

Após a geração do PSM conforme *Node-WoT* e *WebSockets*, uma definição de transformação de modelo-a-texto foi aplicada, ou seja, uma *Thing Description* na linguagem *JavaScript* com a inserção da biblioteca *Node-WoT* possuindo protocolo de comunicação *WebSocket* para se comunicar com IoT *device*.

A Figura 44 demonstra o código da TD de um LED. O código criado foi para expor uma *Thing* utilizando *WoT.produce*, sendo uma *promise* utilizada para processamento assíncrono, retornando o objeto *thing*.

Continuando com a Figura 44 onde possui os métodos citados na *WoT Script Api*, como *getThingDescription()*, que retorna o objeto que representa a *Thing Description* em conjunto com o título. Após, a utilização do método *writeProperty()*, que cria a propriedade "on" em conjunto com a função *geton()*, que busca a informação do dispositivo. O método *setPropertyReadHandler()*, que cria uma *promise*, utilizando a função *geton()*, para retornar o valor do dispositivo, sempre que o valor da propriedade for solicitado. E *setActionHandler()* contém o método de solicitações das ações.

C: > ExemploEclipseMyLamp > JS MyLamp.js > ...

```
1  var result
2      Wot.Produce({
3      title : "MyLamp",
4      description:"A thing to control the Led",
5          properties: {
6          on:{
7              type :"boolean",
8              description:"My lamp led onoff"
9              }
10 },
11         action: {
12         toggle:{
13             description:"current status of the lamp on off"
14         }
15 },
16         event: {
17         overheat:{
18             description:"Alert sent whem the led temperature is too high"
19         }
20 },
21 })
22     .then (function(thing) {
23 console.log("Produced "+ thing.getThingDescription().title);
24 thing.writeProperty("on",geton());
25 thing.setPropertyHandler("on", function(){
26         return new promise((resolve,reject)=>{
27         resolve(geton());
28     });
29  });
30 thing.setActionHandler("toggle",function (value,option){
31     changedStatus(value)
32  });
33 thing.expose().then(function(){console.info(thing.getThingDescription().title + "ready"); });
34 const WebSocket = require("ws");
35     const websocket = new WebSocket("ip");
36     websocket.on("open", getOn);      function geton(){
```

Figura 42 – Parte do código gerado Thing Description de um LED. Fonte: Autor.

Na Figura 45 a utilização do método *thing.expose()* é para expor a *Thing* no terminal, contendo os URIs necessários para alterá-la a Thing. Após são as inserções das propriedades da conexão com *WebSocket*. A função *geton()*, são para o contato com a thing sendo virtual ou não, no caso específico a comunicação com LED está sendo realizada através de *WebSockets*.

Outra função *changeLedStatus()* encontrada na Figura 45, chamada pelo método da *interface exposedThing*, que muda o valor da propriedade do dispositivo sendo um LED para true ou false, ligado ou desligado. E por último, a função *catch* que retorna qualquer erro da *promise*.

```
    // expose the thing
    thing.expose().then(function () { console.info(thing.getThingDescription().title + " ready"); });
    const WebSocket = require('ws');
    const websocket = new WebSocket('ws://192.168.4.1:1337/');
    websocket.on('open', getOn);
    function getOn() {

        var testando = websocket.send('getLEDState');
          websocket.on('message', function message(msg) {
          if(msg == 0){
              result=false;
              obterResult(result)
          } else{
              result=true;
              obterResult(result)
          }
        });
        function obterResult(){
          return result;
      }
     //  return websocket.on('message', message());
       return obterResult();
    }
    function changeLedStatus(newValue){
        // normally, you would do physical action to change the temperature
        //do nothing
       // thing.writeProperty("on",newValue);
        websocket.send('toggleLED');
        return;
    }
})
    .catch(function (e) {
    console.log(e);
});
```

Figura 43 – Parte final do código da Thing Description de um LED. Fonte: Autor.

Na parte de IoT de aplicação do *framework* conforme a Figura 46 a criação do modelo de negócio para construção do código de um atuador ou led. Este modelo foi criado pelo metamodelo IoT *device*, podendo preencher todas as suas classes.

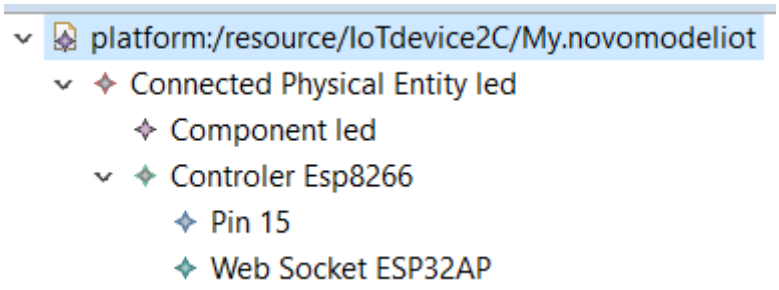

Figura 44 – Modelo IoT para um atuador LED. Fonte: Autor.

Após a criação do modelo, as transformações são feitas pelo *framework*, de PIM para as transformações para PSM em formato de *XMI*, para definição de transformação com o metamodelo da linguagem C resultando em *PSM Concrete*.

Após a geração do PSM *Concrete* conforme *WebSockets*, uma definição de transformação de modelo-a-texto foi aplicada gerando código para a placa *Esp8266* que controla um LED em ligá-lo ou desligá-lo.

A Figura 47 contém o código de controle do LED. O código foi criado com a inserção das bibliotecas *Wifi, Esp8266Wifi, FS, ESPsyn, WebsocktServer*, necessárias para a compilação e execução do código. As variáveis utilizadas para *Wifi*, e as específicas do *Websocket* são para conexão. Após o início da conexão com o cliente *void onWebSocketEvent()*, onde o *switch*, possui os casos possíveis em relação à conexão, como a identificação se o cliente se conectou, não conectou, ou não reconhecimento da mensagem.

Na Figura 48, em caso de recebimento da mensagem *WebSocket*, denominado *toggleLED*, será utilizada a função para alterar o estado do LED, de ligado para desligado e desligado para ligado . Quando ao recebimento da mensagem sempre será retornado o estado real do LED, de forma binária.

Já na Figura 49, as funções *case*, são de default da biblioteca *WebSocket*. *O void setup()*, possuindo a definição da porta serial, as conexões de *Wifi* da placa *Esp8266* e por último o *Void loop*, fazendo com que a conexão com o *WebSocket* não pare de executar a conexão.

led

```
#include<Wifi.h>
#include<ESP8266WiFi.h>
#include<FS.h>
#include<ESPAsyncWebServer.h>
#include<WebSocketsServer.h>
const char *ssid = ESP32AP;
const char *password = LetMeInPlz;
const char *msg_toggle_led = "toggleLED";
const char *msg_get_led = "getLEDState";
const int dns_port =53;
const int http_port= 80;
const int ws_port =1337;

AsyncWebServer server(80);
char msg_buf[10]; int led_state=0;
WebSocketsServer webSocket = WebSocketsServer(1337);
//Call and Recive any WebSocket Message
void onWebSocketEvent(uint8_t client_num,
                      WStype_t type,
                      uint8_t * payload,
                      size_t length) {// Figure out the type of WebSocket event
  switch(type) {

    // Client has disconnected
    case WStype_DISCONNECTED:
      Serial.printf("[%u] Disconnected!
", client_num);
      break;

    // New client has connected
    case WStype_CONNECTED:
      {
        IPAddress ip = webSocket.remoteIP(client_num);
```

Figura 45 – Parte de código gerado do controle de um LED para placa *Esp8266*. Fonte: Autor.

```
      // New client has connected
      case WStype_CONNECTED:
        {
          IPAddress ip = webSocket.remoteIP(client_num);
          Serial.printf("[%u] Connection from ", client_num);
          Serial.println(ip.toString());
        }
        break;

      // Handle text messages from client
      case WStype_TEXT:
      // Print out raw message
      Serial.printf("[%u] Received text: %s
", client_num, payload);
      // Recive message
       if ( strcmp((char *)payload, "toggleLED") == 0 ) {
           led_state = led_state ? 0 : 1;
          Serial.printf("Toggling LED to %u\n", led_state);
          digitalWrite(led_pin, led_state);

            }
       else if ( strcmp((char *)payload, "getLEDState") == 0 ) {

          sprintf(msg_buf, "%d", led_state);
          Serial.printf("Sending to [%u]: %s\n", client_num, msg_buf);
          webSocket.sendTXT(client_num, msg_buf);

       } else {
          Serial.println("[%u] Message not recognized");
       }
       break;
```

Figura 46 – Parte de código gerado para conexão do WebSockete para acionamento e retorno do status de um LED. Fonte: Autor.

```
      // For everything else: do nothing
      case WStype_BIN:
      case WStype_ERROR:
      case WStype_FRAGMENT_TEXT_START:
      case WStype_FRAGMENT_BIN_START:
      case WStype_FRAGMENT:
      case WStype_FRAGMENT_FIN:
      default:
        break;
    }
}void setup() {

  // Start Serial port
  Serial.begin(115200);

  // Start access point
  WiFi.softAP(ssid, password);

  // Print our IP address
  Serial.println();
  Serial.println("AP running");
  Serial.print("My IP address: ");
  Serial.println(WiFi.softAPIP());

  // Start WebSocket server and assign callback
  webSocket.begin();
  webSocket.onEvent(onWebSocketEvent);
}
void loop() {
  // Look for and handle WebSocket data
  webSocket.loop();
}
```

Figura 47 – Última parte do código gerado para conexão do Wifi da placa retorno do status de um LED. Fonte: Autor.

Após a criação da *Thing Description do LED*, é necessário executar o código em *Node Js*, utilizando o comando de linha (A biblioteca irá disponibilizar uma URI de acesso). A TD pode ser acessada por um navegador ou um cliente *REST*. Após é utilizado um dos protocolos disponíveis pela biblioteca para ler e alterar os modelos de interações.

Para exemplificar melhor na Figura 50, a *Thing Description* do LED acessada por meio da URI, destacando o título, descrição, propriedades, o tipo sendo *boolean*, descrição da *Thing, readonly, observable, writeOnly,* e o *forms*.

Os *forms* contêm a referência de acesso da propriedade (http://[2804:14d:8c88:8a32::3]:8080/MyLampLed/properties/on), o *content type*, op de leitura *readproperty* e o método *GET* para protocolo HTTP.

O mesmo acontece com a referência de acesso do *observable* (http://[2804:14d:8c88:8a32::3]:8080/MyLampLed/properties/on/observable) uma técnica que tem como objetivo lidar com atividades assíncronas e mudanças, como um push no servidor ou uma propriedade nova for criada. Isso permite que a função perceba as mudanças do estado emitidas para lidar com atividades em tempo real.

```
{
    "title": "MyLampLed",
    "description": "A Thing to control the LED",
    "properties": {
        "on": {
            "type": "boolean",
            "description": "My Lamp led onOff",
            "observable": true,
            "readOnly": true,
            "writeOnly": false,
            "forms": [
                {
                    "href": "http://[2804:14d:8c88:8a32::3]:8080/MyLampLed/properties/on",
                    "contentType": "application/json",
                    "op": [
                        "readproperty"
                    ],
                    "htv:methodName": "GET"
                },
                {
                    "href": "http://[2804:14d:8c88:8a32::3]:8080/MyLampLed/properties/on/observable",
                    "contentType": "application/json",
                    "op": [
                        "observeproperty"
                    ],
                    "subprotocol": "longpoll"
                },
```


Figura 48 – Parte da API da Thing Description de um LED- Properties. Fonte: Autor.

Na Figura 51, a continuação dos formulários na interação com as propriedades por meio do protocolo de comunicação CoAP (coap://[2804:14d:8c88:8a32:4946:b3d0:6534:2ac8]:5683/MyLampLed/properties/on), *contente Type*, e *op readproperty*.


```
{
    "href": "coap://[2804:14d:8c88:8a32:4946:b3d0:6534:2ac8]:5683/MyLampLed/properties/on",
    "contentType": "application/json",
    "op": [
        "readproperty"
    ]
},
```


Figura 49 – Parte da API da Thing Description de um LED- Properties para protocolo CoAP. Fonte: Autor.

Para a interação com as ações na Figura 52, a TD disponibiliza *toggles*, descrição, e os formulários de interação, op de *invokeaction* e protocolo HTTP com método *POST*. Já na Figura 53, são as interações feitas pelo protocolo CoAP.

```
"actions": {
    "toggle": {
        "description": "current status of the lamp(on|off)",
        "forms": [
            {
                "href": "http://[2804:14d:8c88:8a32::3]:8080/MyLampLed/actions/toggle",
                "contentType": "application/json",
                "op": [
                    "invokeaction"
                ],
                "htv:methodName": "POST"
            },
            {
                "href": "http://[2804:14d:8c88:8a32:4946:b3d0:6534:2ac8]:8080/MyLampLed/actions/toggle",
                "contentType": "application/json",
                "op": [
                    "invokeaction"
                ],
                "htv:methodName": "POST"
            },
```


Figura 50 – Parte da API da Thing Description de um LED- Action. Fonte: Autor.


```
{
    "href": "coap://[2804:14d:8c88:8a32::3]:5683/MyLampLed/actions/toggle",
    "contentType": "application/json",
    "op": "invokeaction"
},
```


Figura 51 – Parte da API da Thing Description de um LED- Action para protocolo CoAP. Fonte: Autor.

E por último na Figura 54, eventos, sendo do tipo *overheat* caso o estado no LED fique com a temperatura muito alta. As formas de interação podem ser pelo protocolo HTTP como subprotocolo um *longpoll*, ou como na Figura 55 com utilização do protocolo *Websocket*. O protocolo WebSocket somente é utilizado na interação do evento. E por último o protocolo de comunicação CoAP.


```
"events": {
    "overheat": {
        "description": "Alert sent when the Led temperature is too high",
        "forms": [
            {
                "href": "http://[2804:14d:8c88:8a32::3]:8080/MyLampLed/events/overheat",
                "contentType": "application/json",
                "subprotocol": "longpoll",
                "op": [
                    "subscribeevent"
                ]
            },
```


Figura 52 – Parte da API da Thing Description de um LED- Events. Fonte: Autor.

```
{
    "href": "ws://192.168.0.27:8080/MyLampLed/events/overheat",
    "contentType": "application/json",
    "op": "subscribeevent"
},
{
    "href": "coap://[2804:14d:8c88:8a32::3]:5683/MyLampLed/events/overheat",
    "contentType": "application/json",
    "op": "subscribeevent"
},
```


Figura 53 – Parte da API da Thing Description de um LED- Action para protocolo Websocket e CoAP. Fonte: Autor.

Na Figura 56, após os eventos se destaca o contexto da *Thing* que demonstra a padronização conforme a W3C, na versão um da documentação. O tipo, requisito de segurança, o identificador único e o formulário para a interação de mais de uma propriedade caso exista, mas infelizmente só para o protocolo HTTP.


```
"@context": "https://www.w3.org/2019/wot/td/v1",
"@type": "Thing",
"security": [
    "nosec_sc"
],
"id": "urn:uuid:b3f2487b-eec4-43c5-a339-84e6267dff4d",
"forms": [
    {
        "href": "http://[2804:14d:8c88:8a32::3]:8080/MyLampLed/all/properties",
        "contentType": "application/json",
        "op": [
            "readallproperties",
            "readmultipleproperties"
        ]
    },
```


Figura 54 – Parte da API da Thing Description de um LED All properties. Fonte: Autor.

6.2 Exemplo ilustrativo sensor de Temperatura

Outro exemplo aplicado pelo *framework* na parte de WoT conforme a Figura 57, é a criação do modelo de metamodelo *Thing Description* para a construção de uma *Thing Description* de um sensor de temperatura, definindo o nome da *Thing*, *properties, action, events,* e *form.*

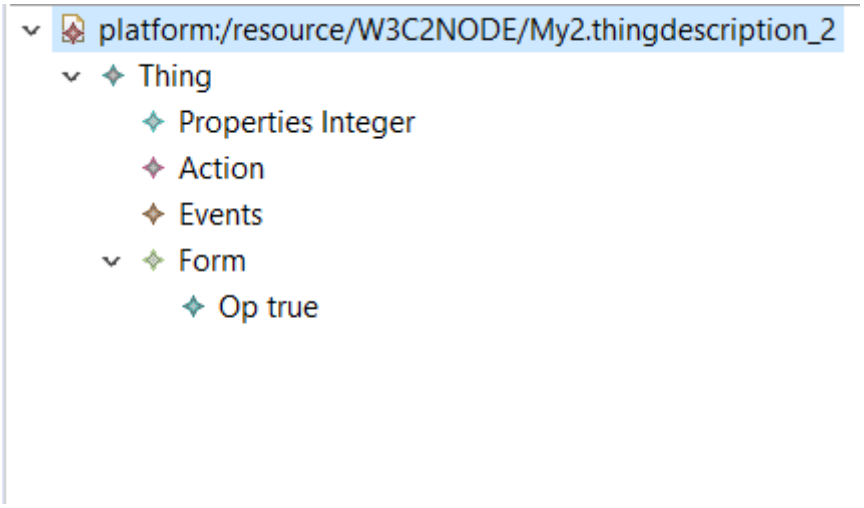

Figura 55 – Modelo Thing Description para um sensor de Temperatura. Fonte: Autor.

Após as transformações feitas pelo *framework*, o código gerado se encontra na Figura 58, definindo WoT. Produce para expor a *Thing*, as *properties* e suas descrições, responsável por informar o valor da temperatura. *Action* caso seja necessário o aumento ou diminuição da temperatura, mas para isso é necessário possuir o dispositivo capaz de tal feito, sendo assim apenas simulação de um dispositivo virtual. *Event*, para verificação da temperatura caso esteja muito alta pelo evento chamado *overheat*.

```
var result;
    Wot.Produce({
    title : "MyTemperature",
    description:"MyTemperature Thing",
        properties: {
        temperature:{
            type :"Integer",
            description:"My temperature"
            }
},
        action: {
        increment:{
            description:"Incrementing the temperature of the room with 0 to 5 increments"
         }
},
        event: {
        overheat:{
            description:"Alert sent when the room temperature is too high"
        }
},
})
```

Figura 56 – Parte do código gerado Thing Description de um sensor de temperatura. Fonte: Autor.

Na parte do *framework* IoT, conforme a Figura 59, a criação do modelo de metamodelo IoT *Device* da construção do código de um sensor de temperatura, que só verifica a temperatura do ambiente.

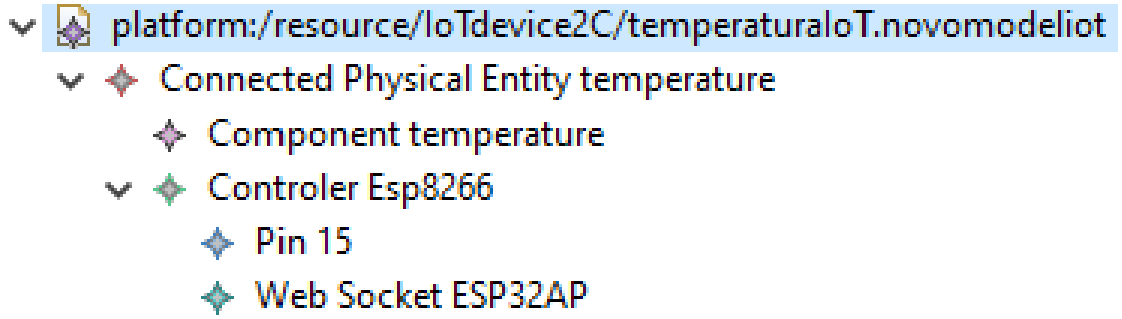

Figura 57 – Modelo IoT para um sensor de temperatura. Fonte: Autor.

Após as transformações feitas pelo *framework*, uma parte do código gerado se encontra na Figura 60, cada vez em que for enviado a mensagem pelo cliente *getTempState*, será retornado o valor da temperatura. A TD que manipula o dispositivo IoT está disponível em anexo.

```
 if ( strcmp((char *)payload, "getTempState") == 0 ) {

int valorObtido = digitalRead(PIN_SENSOR);
float milivolts = (valorObtido/1024.0) * 3300;
float celsius = milivolts/10;
Serial.print("A temperatura é de = ");
Serial.println(celsius);

sprintf(msg_buf, "%f", celsius);

Serial.printf("Sending to [%u]: %s\n", client_num, msg_buf);
webSocket.sendTXT(client_num, msg_buf);
```

Figura 58 – Parte de código gerado para conexão do WebSocket para retorno do status da temperatura. Fonte: Autor.

6.3 Exemplo ilustrativo sensor de Presença Pir

Para o desenvolvimento da TD de um sensor de presença Pir, é necessário o preenchimento do modelo de metamodelo *Thing Description* na Figura 61, definindo o nome da *Thing*, *properties, action, events, Form* e os tipos de op. Após as transformações feitas pelo *framework*, há a geração de código e uma parte dele se encontra na Figura 62.

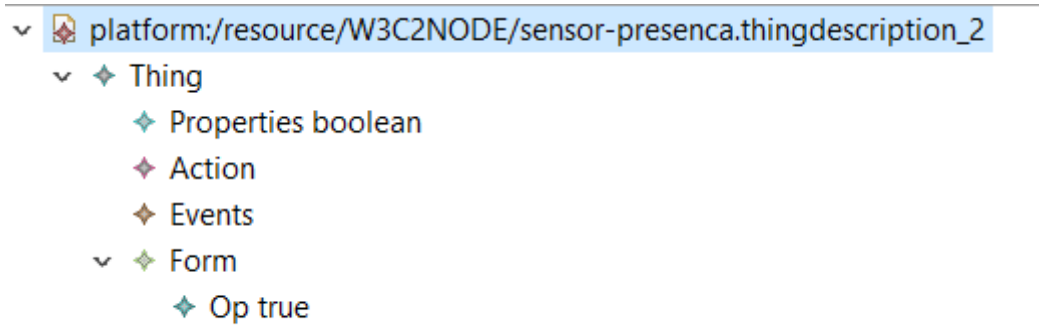

Figura 59 – Modelo Thing Description para um sensor Pir. Fonte: Autor.

A figura 62 mostra a construção da TD, pelo *WoT.Produce*, com título *myPirSensor*, possuindo como descrição verificar se existe presença ou não em uma distância de até sete metros do tipo booleano, a *properties* responsável por retornar o valor do sensor de (*true* ou *false*). *Actions*, caso necessário mudar o status do sensor, porém, não implementamos essa ação, pois um sensor de presença não necessita de mudar sua propriedade. *Events*, também não possuem nenhum específico aplicado ao pir. Mas é necessário manter a estrutura (*properties, action e events*) para o usuário entender que não necessita mudança do estado ou evento.

```
var result;
WoT.produce({
    title: "MyPirSensor",
    description: "A Thing to get sensor presence",
    properties: {
        pirsensor: {
            type: "boolean",
            description: "My sensor true or false",
            observable: true,
            readOnly: true
        }
    },
    actions: {
        toggle: {
            description:"No actione need"
            }
        },
    events: {
        no: {
            description: "No event specific"
        }
    },
})
```

Figura 60 – Parte do codigo gerado Thing Description de um sensor de presença. Fonte: Autor.

Para o desenvolvimento do código da plataforma IoT, é necessário a utilização do modelo do metamodelo IoT *Device* na figura 63, preenchendo todos os campos das classes conforme as propriedades de um sensor de presença. As transformações são feitas pelo *framework* como resultado do código na Figura 64.

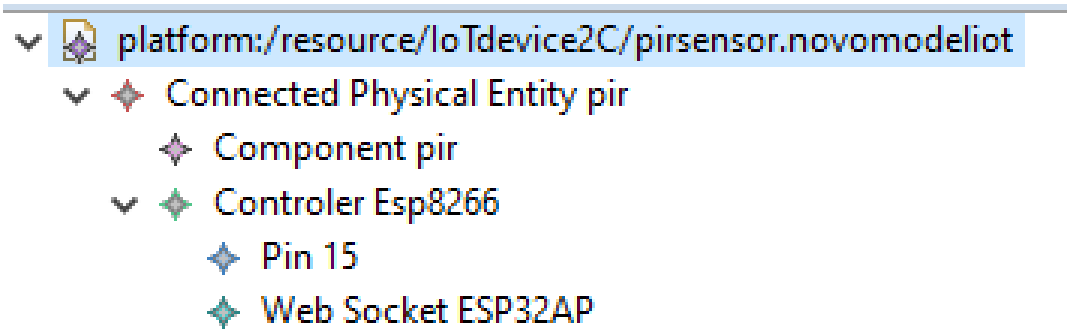

Figura 61 – Modelo IoT Device para um sensor de Presença Pir. Fonte: Autor.

A parte do código gerado pelo *framework* conforme a Figura 64, mostra apenas a parte da comunicação do *websocket* no envio da mensagem pelo cliente, *getPirState* para a verificação do sinal do sensor e o retorno do estado para o cliente, especificamente TD *MyPirSensor*.

```
// Recive message
 if ( strcmp((char *)payload, "getPirState") == 0 ) {

     int sinal = digitalRead(PIN_SENSOR);
         Serial.print("A presença é de = ");
         Serial.println(sinal);

    sprintf(msg_buf, "%d", sinal);
    }
```

Figura 62 – Parte de código gerado para conexão do WebSocket para o retorno do status do sensor Pir. Fonte: Autor.

6.4 Análise dos Resultados e Comparações

Os seguintes critérios foram escolhidos como relevantes para a análise dos trabalhos apresentados no Estado da Arte:

- Criação de *Thing Description* baseado na W3C: Para compreender se os trabalhos possuem a padronização da W3C;
- Automatização na geração de código: Se possuem automatização na geração de códigos fontes;
- Protocolos de comunicação: Quais os protocolos são utilizados;
- Linguagem de programação para a geração da *Thing Description*: Quais as linguagens são utilizadas para gerar a TD;
- Aplicação de WoT para *Smart Spaces*: Se possuem aplicações de WoT em *Smart Spaces*, para aplicações de cenários prontos;
- Implementação de *WoT Scripting Api ProducedThing*: Se possuem aplicação da interface;
- Implementação de *WoT Scripting Api ConsumedThing*: Se possuem aplicação da interface;
- Implementação de *WoT Scripting Api DiscoveryThing*: Se possuem aplicação da interface;
- Comunicação com IoT *device*: Se existe a comunicação com dispositivo IoT.

Destacamos as pesquisas apresentadas em comparação com os critérios definidos:

1. A pesquisa (HASSINE HB. ; KORKAN, 2019), propõe a utilização da W3C em conjunto com a biblioteca Node-WoT, para a simulação de Thing Description virtuais na linguagem javaScript, facilitando o teste para o desenvolvimento de TD e possibilitando mais de um protocolo de comunicação (HTTP, CoAP e WebSockets) para sua manipulação. Mas o trabalho não foca na implementação em dispositivos físicos, aplicação para Smart Spaces, automatização de código e aplicação de Script API.
2. A pesquisa (SAKAMOTO T. ; ITO, 2017), propõe uma arquitetura dinâmica para Web API conforme a W3C utilizando protocolos MQTT e linguagem JavaScript para implementação de dispositivos físicos. Contudo a pesquisa da W3C vem sendo atualizada desde da publicação do artigo em 2017 tornando a arquitetura

desatualizada. Também não possui, automatização de código, geração de Thing Description e aplicação Script Api;

3. A pesquisa (BEZERRA, 2019) utiliza as abordagens MDE em conjunto padrões da WoT W3C que interagem com qualquer WoT de acordo com a TD para existir consumo de TD e adaptação de interface para manipular WoT. Mas a pesquisa não se aplica para Smart Spaces, Script Api e não gera código para dispositivos IoT;

4. A pesquisa (CORREDOR, 2012), propõe uma metodologia que permite a construção de espaços inteligentes para Web of Things. Automatização e geração de códigos para dispositivos IoT. Mas o trabalho não estar de acordo com os padrões da W3C ;

5. A pesquisa (URKIA, 2019), utiliza a abordagem MDE, apesar de possuir o padrão da W3C e automatização dos códigos, a proposta utiliza apenas de um protocolo de comunicação da TD, e a geração de código apenas para dispositivos para C + +. Mas a pesquisa não foca em aplicações para Smart Spaces e aplica apenas discovery do Script Api;

6. A pesquisa (ALULEMA D. ; CRIADO, 2019), utiliza abordagem MDE para a criação de nós de hardware para que desenvolvedores não precisem ter um conhecimento prévio para o desenvolvimento. Contudo a pesquisa não aplica os padrões da W3C e nem para Smart Spaces ;

7. A pesquisa (NEPOMUCENO, 2020), utiliza das abordagens para a criação de código, cenários para dispositivos IoT e a possibilidade de estender códigos sem precisar de um conhecimento prévio além de um arquivo JSON. Porém não aplica para os padrões da W3C e Smart Spaces.

As Tabelas 2 e 3 retratam os trabalhos pesquisados quais deles obedecem os nove critérios destacados em comparação com o *framework* proposto.

O *framework* proposto contempla quase todos os critérios destacados, como a criação de *Thing Description* baseado na W3C, automatização de código tanto para Thing Description quanto para dispositivo IoT. A utilização de protocolos *WebSockets* entre as tecnologias WoT e IoT. A linguagem de programação ATL utilizada para geração de *Thing Description* necessária para automatização de códigos por meio de modelos. A Implementação *WoT Script Api* para *ProducedThing interface*, *ConsumedThing interface* e *DiscoveryThing interface*, e a comunicação com IoT *device* por meio de *WebSockets*. E o único critério não contemplado pelo *framework*, são as aplicações de WoT para *Smart Spaces*. Portanto, em comparação com os trabalhos destacados, o único que contempla o maior número de critérios.

Tabela 2 – Análise dos Resultados e Comparação

Trabalhos	Criação da Thing Description baseado na W3C	Automatização na geração de código	Protocolos de comunicação	Linguagem de programação para geração de Thing Description
Virtual-Thing: Thing Description based Virtualization (HASSINE HB. ; KORKAN, 2019)	Sim	Não	HTTP, CoAP e Webscoket	JavaScript
Dynamically Exposing and Controlling Physical Devices by Expanding Web of Things Scheme (SAKAMOTO T. ; ITO, 2017)	Sim	Não	MQTT	JavaScript
A Model-Based Approachto Generate Reactive and Customizable User Interface for Web of Things (BEZERRA, 2019)	Sim	Sim	TCP	JavaScript
A development Methodology to Facilitate the Integration of Smart Spaces into the Web of Things (CORREDOR, 2012)	Não	Sim	Não	Não
Enabling Easy Web of Things Compatible Device Generation using a Model-Diven Engineering Approach (URKIA, 2019)	Sim	Sim	CoAP	C++
A Model-Driven Approach for the Integration of Hardware Nodes in the IoT (ALULEMA D. ; CRIADO, 2019)	Não	Sim	Bluetooth e Wifi	Não
AutoIoT: A Framework Based on User-Driven MDE for Generation IoT Applications (NEPOMUCENO, 2020)	Não	Sim	Não	Não
Trabalho Proposto	Sim	Sim	WebSockets	ATL

Tabela 3 – Análise dos Resultados e Comparação

Trabalhos	Aplicação de WoT para Smart Spaces	Implementação de WoT Scripting Api ProducedThing Interface	Implementação de WoT Scripting Api ConsumedThing Interface	Implementação de WoT Scripting Api DiscoveryThing Interface	Comunicação com IoT Device
Virtual-Thing: Thing Description based Virtualization (HASSINE HB. ; KORKAN, 2019)	Não	Sim	Sim	Não	Não
Dynamically Exposing and Controlling Physical Devices by Expanding Web of Things Scheme (SAKAMOTO T. ; ITO, 2017)	Não	Sim	Sim	Sim	Sim
A Model-Based Approachto Generate Reactive and Customizable User Interface for Web of Things (BEZERRA, 2019)	Não	Não	Não	Não	Não
A development Methodology to Facilitate the Integration of Smart Spaces into the Web of Things (CORREDOR, 2012)	Sim	Não	Não	Não	Não
Enabling Easy Web of Things Compatible Device Generation using a Model-Diven Engineering Approach (URKIA, 2019)	Não	Não	Não	Sim	Sim
A Model-Driven Approach for the Integration of Hardware Nodes in the IoT (ALULEMA D. ; CRIADO, 2019)	Não	Não	Não	Não	Sim
AutoIoT: A Framework Based on User-Driven MDE for Generation IoT Applications (NEPOMUCENO, 2020)	Não	Não	Não	Não	Sim
Trabalho Proposto	Não	Sim	Sim	Sim	Sim

6.5 Síntese

Este capítulo apresentou três exemplos para ilustrar a utilização da abordagem proposta para geração de *Thing Description* em conjunto, dispositivos IoT para placa *Esp8266*.

Nestes estudos, os metamodelos propostos são aplicados para as definições de transformações são essenciais para geração e composição do framework que possibilitou a construção de projetos WoT em conjunto com projetos IoT.

E,por último, uma análise das pesquisas foi realizada com base em critérios comparativos com intuito de fundamentar a proposta dessa dissertação.

7 Conclusões e Trabalhos Futuros

Neste capítulo, discute-se os objetivos atingidos, as limitações desta abordagem e as sugestões de trabalhos futuros serão apresentados.

Este trabalho apresentou um *framework* baseado em MDE para suportar WoT e dispositivo IoT de modo a apoiar a modelagem de *Thing Description* e desenvolver o WoT em conjunto com a IoT.

Metamodelos foram desenvolvidos. Dentre eles, os para o desenvolvimento WoT, metamodelos de *Thing Description* foram desenvolvidos conforme a W3C e *Node-WoT* foram produzidos para geração TD para a linguagem Js e suporte aos protocolos HTTP, CoAP e *WebSocket*. E para desenvolvimento IoT, os metamodelos IoT *device* para suportar sensores e atuadores em conjunto com protocolo *WebSocket*. E metamodelo para linguagem C em conjunto com protocolo de comunicação de *WebSocket*.

Um protótipo foi construído, abrangendo a comunicação entre às duas propostas WoT e IoT. E Finalmente, exemplos ilustrativos foram apresentados para demonstrar a utilização do *framework* desenvolvido. Pode-se considerar, portanto que a abordagem desenvolvida pode fornecer:

- Capacidade de se adaptar e evoluir em conjunto com os padrões da W3C por meio de *Model Driven Engineering*: Por meio da abordagem MDE é possível adaptar os modelos se aplicados de forma correta;
- Incentivo da padronização da W3C por modelos: O *framework* proposto prevê a padronização da *Thing Description* onde o usuário só necessita de um conhecimento prévio para preencher os dados do modelo criado;
- Beneficiar projetos IoT trazidas pela *Web of Things*: Por meio do *framework* os projetos IoT podem utilizar a TD para manipular seus dispositivos facilitando a inserção em projetos WoT;
- Fácil adaptação a novas propostas através de modelos: Os modelos fornecem uma alta abstração proporcionada pelos modelos por meio da arquitetura em camadas, e assim uma fácil manutenção;
- Automatização na criação de códigos: Por meio do *framework* proposto na transformação modelo a texto a geração do código de forma automática;

7.1 Objetivos Alcançados

Considerando que o principal objetivo desta dissertação consiste no desenvolvimento de uma abordagem em MDE que suporte a geração de *Thing Description* em conjunto com IoT *device*, os seguintes objetivos foram alcançados:

- Proposta de uma abordagem em MDE para geração *Web of Things*, *Thing Description* W3C em conjunto com dispositivo IoT por *WebSockets*;
- Aplicação da padronização proposta pela W3C em conjunto com a biblioteca *Node-WoT* para linguagem Js e *WebSocket*;
- Aplicação de sensores e atuadores para placa *Esp8266* em linguagem C e *WebSockets*;
- Automatização para geração de TD durante o processo de desenvolvimento;
- Automatização de sensores e atuadores durante o processo de desenvolvimento;
- Protótipo do *Framework* e o fornecimento de três exemplos ilustrativos para validação da abordagem.

7.2 Limitações e Trabalhos Futuros

A pesquisa atingiu o objetivo principal, porém, apresenta as seguintes limitações:

- A abordagem não cria mais de uma propriedade, ações ou eventos;
- Abordagem não atinge todos os sensores e atuadores existentes;
- Abordagem não gera código para portas analógicas;
- A biblioteca utilizada não executa mais de uma TD em um servidor Node ao mesmo tempo;
- Abordagem não utiliza o modelo PIM do *framework* WoT e IoT na prototipagem;

Os trabalhos futuros são apresentados como segue:

- Ampliação do *framework* para a criação de PSM para outras linguagens de programação;
- Ampliação da abordagem para gerar mais de uma propriedade, ações ou eventos;
- Ampliação do cenário de *Web of Things* para aplicação em *Smart Spaces virtual* ou não;

- Ampliação do *framework* para geração de código para outras placas como *arduino*, *raspberry* e etc;
- Ampliação do *framework* para a utilização do PIM na prototipagem;
- Propor um experimento controlado com duas equipes para desenvolver com e sem o *framework*, para tentar comprovar rapidez e facilidade de uso.

7.3 Publicações

O trabalho apresentado neste manuscrito de dissertação de mestrado foi aceito na forma de artigo *ESSE 2020: Proceedings of the 2020 European Symposium on Software Engineering, Rome Italy November, 2020*. Esta conferência teve seus anais publicados pelo *ACM (International Conference Proceeding Serie (ICPS e está disponível na ACM Digital Library ISBN: 978-1-4503-7762-1))*. O artigo é referenciado como segue:

DE OLIVEIRA, RAYANNE SILVA ; Lopes, Denivaldo. *An Approach based on Model Driven Engineering to Support the Development of Web of Things. In: ESSE 2020: 2020 European Symposium on Software Engineering, 2020, Rome Italy. Proceedings of the 2020 European Symposium on Software Engineering. New York: ACM, 2020. p. 168*. DOI: https://doi.org/10.1145/3393822.3432337.

Referências

ALULEMA D. ; CRIADO, J. . I. L. A model-driven approach for the integration of hardware. *New Knowledge in Information Systems and Technologies*, p. 801–811, 2019. Citado 8 vezes nas páginas 8, 9, 43, 44, 56, 90, 91 e 92.

ATL. *ATL DOCUMENTATION*. 2020. Disponível em:<https://www.eclipse.org/atl/documentation/>. Acesso em 22/09/2020. Citado 4 vezes nas páginas 8, 19, 23 e 63.

BEZERRA, J. e. a. A model-based approach to generate reactive and customizable user interfaces for the web of things. *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web, WebMedia 2019*, 2019. Citado 6 vezes nas páginas 8, 39, 40, 90, 91 e 92.

BORGES, H. P. e. a. *Uma Arquitetura Baseada Em Modelos*. 2017. Monografia, Federal Institute of Education, Science and Technology of Maranhão, São Luís, Brasil. Citado na página 21.

CORREDOR, I. e. a. A development methodology to facilitate the integration of smart spaces into the web of things. *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, 2012. Citado 6 vezes nas páginas 8, 41, 42, 90, 91 e 92.

ECLIPSE. *ECLIPSE MODELING FRAMEWORK*. 2020. Disponível em: <https://www.eclipse.org/modeling/emf/>. Acesso em 22/09/2020. Citado 2 vezes nas páginas 19 e 63.

ECLIPSE. *EcoreTools*. 2020. Disponível em: <https://www.eclipse.org/ecoretools/doc/>. Acesso em 22/09/2020. Citado 3 vezes nas páginas 19, 22 e 63.

GUINARD DOMINIQUE ; TRIFA, V. *Building the Web of Things*. Shelter Island. New York, USA.: Manning Publications, 2016. Citado 7 vezes nas páginas 8, 16, 17, 24, 25, 26 e 27.

HANES D. ; SALGUEIRO, G. . B. R. *Fundamentos de IoT: tecnologias de rede, protocolos e casos de uso para a Internet das coisas*. [S.l.]: Cisco Press, junho 2017, 2017. Citado 3 vezes nas páginas 8, 23 e 24.

HASSINE HB. ; KORKAN, E. S. S. Virtual-thing: Thing description based virtualization. *arXiv:1909.03297*, 2019. Citado 6 vezes nas páginas 8, 36, 37, 89, 91 e 92.

KINTSCHNER, M. *Parada de Ônibus Inteligente: Uma Aplicação para Cidades Inteligentes Baseado no Conceito de Web das Coisas*. 2017. Monografia,Universidade Federal do Rio Grande do Sul Instituto de Informática, Brasil. Citado 5 vezes nas páginas 16, 24, 27, 29 e 32.

KLEPPE ANNEKE. ; WARMER, J. B. W. *MDA Explained: The modelo Driven Architecture: Practice and Promise*. Canada: Addison-Wesley Professional, 2003. Citado 5 vezes nas páginas 8, 17, 18, 21 e 22.

KON F; SANTANA, Z. F. E. Cidades inteligentes: Conceitos, plataformas e desafios. In: *XXXVI Congresso da Sociedade Brasileira de Computação, Porto Alegre RS*. [S.l.: s.n.], 2016. Citado na página 24.

LAB, L. T. S. *Web of Things (WoT)*. 2021. Disponível em:<https://tingge.github.io/html/wot.html>. Acesso em 10/08/2021. Citado 2 vezes nas páginas 8 e 27.

LOMBARDI, A. *Websocket: Lightweight Client-Server Communications*. [S.l.]: O'Reilly Media, 2015. Citado na página 33.

MARTINS A. J.; MAZAYEV, A. . C. N. Hypermedia apis for the web of things. *Center for Electronics, Optoelectronics and Telecommunications, University of Algarve*, 2017. Citado na página 34.

MOHAGHEGHI, e. a. Where does model-driven engineering help? experiences from three industrial cases. *Softw Syst Model*, 2011. Citado 3 vezes nas páginas 18, 21 e 22.

MOZILLA. *Mozilla W3C Differences*. 2020. Disponível em: <https://github.com/webthingsio/wiki/wiki/Mozilla-W3C-Differences>. Acesso em 22/09/2020. Citado na página 34.

MOZILLA. *Mozilla WebThings Documentation*. 2020. Disponível em: <https://iot.mozilla.org/docs/>. Acesso em 22/09/2020. Citado na página 34.

NEPOMUCENO, T. e. a. Autoiot: A framework based on user-driven mde for generating iot applications. *Proceedings of the ACM Symposium on Applied Computing 2020*, 2020. Citado 6 vezes nas páginas 8, 44, 45, 90, 91 e 92.

SAKAMOTO T. ; ITO, H. . N. k. Dynamically exposing and controlling physical devices by expanding web of things scheme. *IEEE 41st Annual Computer Software and Applications Conference*, 2017. Citado 6 vezes nas páginas 8, 37, 39, 89, 91 e 92.

SILVA, A. R. Model-driven engineering: A survey supported by unifes conceptual model. *Computer Languages, Systems Structures*, p. 139–155, 05 2015. Citado 4 vezes nas páginas 17, 18, 21 e 22.

SILVA, L. D. F. D. *UMA ABORDAGEM BASEADA EM ENGENHARIA DIRIGIDA POR MODELOS PARA SUPORTAR A GERAÇÃO DE CASOS DE TESTES EM APLICAÇÕES VOLTADAS PARA DISPOSITIVOS MÓVEIS*. 2020. Dissertação,Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Federal do Maranhão, Brasil. Citado 3 vezes nas páginas 8, 50 e 52.

THINGWEB. *A Web of Thing Implementation*. 2020. Disponível em: <https://www.thingweb.io/>. Acesso em 22/09/2020. Citado 3 vezes nas páginas 8, 19 e 54.

URKIA, M. e. a. Enabling easy web of things compatible device generation using a model-driven engineering approach iot 2019. *Association for Computing Machinery*, 2019. Citado 6 vezes nas páginas 8, 41, 43, 90, 91 e 92.

W3C. *PROPOSED Web of Things Working Group Charter*. 2020. Disponível em:<https://www.w3.org/2019/11/proposed-wot-wg-charter-2019.html>. Acesso em 22/12/2020. Citado 2 vezes nas páginas 16 e 27.

W3C. *Recent changes*. 2020. Disponível em:<https://www.w3.org/WoT/IG/wiki/Special:RecentChanges>. Acesso em 22/12/2020. Citado na página 17.

W3C. *This is the landing page for W3C's activities for the Web of Things*. 2020. Disponível em: <https://www.w3.org/WoT/>. Acesso em 22/09/2020. Citado na página 16.

W3C. *Web of Things Interest Group Charter*. 2020. Disponível em: <https://www.w3.org/2016/07/wot-ig-charter.html>. Acesso em 22/12/2020. Citado 3 vezes nas páginas 16, 26 e 27.

W3C. *Web of Things (WoT) Architecture*. 2020. Disponível em:<https://w3c.github.io/wot-architecture/>. Acesso em 22/12/2020. Citado 4 vezes nas páginas 8, 16, 17 e 28.

W3C. *Web of Things (WoT) Binding Templates*. 2020. Disponível em:<https://w3c.github.io/wot-binding-templates/>. Acesso em 22/12/2020. Citado 2 vezes nas páginas 16 e 32.

W3C. *Web of Things (WoT) Scripting API*. 2020. Disponível em:<https://w3c.github.io/wot-scripting-api/>. Acesso em 22/12/2020. Citado 2 vezes nas páginas 16 e 33.

W3C. *Web of Things (WoT) Security and Privacy Guidelines*. 2020. Disponível em:<https://w3c.github.io/wot-security/>. Acesso em 22/12/2020. Citado na página 16.

W3C. *Web of Things (WoT) Thing Description*. 2020. Disponível em:<https://w3c.github.io/wot-thing-description/>. Acesso em 22/12/2020. Citado 6 vezes nas páginas 8, 16, 29, 30, 31 e 53.

W3C. *Web of Things (WoT): Use Cases and Requirements*. 2020. Disponível em:<https://w3c.github.io/wot-usecases/#sec-requirements>. Acesso em 22/12/2020. Citado na página 17.

WANG V.; SALIM, F. M. P. *The Definitive Guide to HTML5 WebSocket*. [S.l.]: Apress, 2013. Citado na página 33.

8 Anexos

8.1 Regras de transformações Thing Description e Node-WoT

```
1  -- @nsURI W3C=http://www.example.org/thingDescription_2
2  -- @nsURI NODE=http://www.example.org/node_Wot
3
4  module REGRAW3C;
5  create OUT : NODE from IN : W3C;
6
7
8  rule Thing2WotProduce{
9  from e:W3C!Thing
10 to out:NODE!WotProduce(
11    title<-e.title,
12    description<-e.description
13
14 )
15    }
16
17 rule Properties2Properties{
18 from e:W3C!Properties
19 to out:NODE!Properties(
20    PropertyName<-e.title,
21    type<-e.type,
22    description<-e.description,
23    wotproduce<-e.thing
24 )
25    }
26
27 rule Events2Events{
28 from e:W3C!Events
29 to out:NODE!Events(
30    EventName<-e.title,
31    description<-e.description,
32    wotproduce<-e.thing
33 )
34    }
35 rule Action2Action{
36 from e:W3C!Action
```

```
37 to out:NODE!Action(
38   ActionName<-e.title,
39   description<-e.description,
40   wotproduce<-e.thing
41 )
42   }
43
44
45 rule formOp2Function{
46 from e:W3C!Op
47 to out:NODE!Function(
48   setPropertyHandler<-e.readproperty,
49   writeProperty<-e.writeproperty,
50   emitEvent<-e.subscribeevent,
51   setActionHandler<-e.invokeaction,
52   wotproduce<-e.form.thing
53
54 )
55   }
```

8.2 Regras de transformações - Codigo para a geração de Thing Description

```
1  library Biblioteca;
2
3  helper context NODE!WotProduce def: toString(): String =
4     'var result
5     Wot.Produce({' + '\n' +
6       '\t' + 'title : "' + self.title +'",' +'\n' +
7       '\t' + 'description:"' + self.description + '",' + '\n' +
8    '\t' +'\t' +  self.properties -> iterate(i; acc : String = '' |
9        acc + i.toString()) +
10       '\n' +
11
12   '\t' +'\t' + self.action-> iterate(i; acc : String = '' |
13       acc + i.toString()) +
14       '\n' +
15
16   '\t' +'\t' + self.events-> iterate(i; acc : String = '' |
17       acc + i.toString()) +
```

```
18        '\n'+
19          '})\n' +
20    -- getThingDescription
21    '\t'  + '.then (function(thing) {' + '\n' +
22        'console.log("Produced "+ thing.getThingDescription().title);' + '\n' +
23    -- WriteProperty
24        'thing.'+
25
26        self.form->collect(e | e.writeProperty)->
27        iterate(i; acc : String = '' |
28        if i='true' then
29        'writeProperty' else 'erro' endif
30        )
31       + '("' +
32      self.properties->collect(e | e.PropertyName)
33         ->iterate(i; acc : String = '' |
34       acc + i.toString()) +
35            '",get'+
36      self.properties->collect(e | e.PropertyName)
37         ->iterate(i; acc : String = '' |
38       acc + i.toString())
39            + '());\n'
40    -- SetPropertyHandler
41      + 'thing.'+
42      self.form->collect(e | e.setPropertyHandler)->
43        iterate(i; acc : String = '' |
44        if i='true' then
45        'setPropertyHandler' else 'erro' endif
46        ) +
47        '("'+
48       self.properties->collect(e | e.PropertyName)
49         ->iterate(i; acc : String = '' |
50       acc + i.toString()) + '", function(){\n' +
51        '\t \t return new promise((resolve,reject)=>{ \n'+
52          '\t \t resolve(get' +
53      self.properties->collect(e | e.PropertyName)
54         ->iterate(i; acc : String = '' |
55       acc + i.toString())+ '());\n \t });\n }); \n'+
56
57  -- Set ActionHandler
58      'thing.' +
59      self.form->collect(e | e.setActionHandler)->
```

```
60        iterate(i; acc : String = '' |
61        if i='true' then
62        'setActionHandler' else 'erro' endif
63        )+ '("' +
64      self.action->collect(e | e.ActionName)
65         ->iterate(i; acc : String = '' |
66       acc + i.toString()) + '",function (value,option)
67       {\n' +
68          '\t changedStatus(value)\n });\n' +
69 -- Expondo a thing
70   'thing.expose().then(function(){console.info
71   (thing.getThingDescription().title + "ready"); });\n'+
72 -- Conex o com Websocket
73   'const WebSocket = require("ws");
74     const websocket = new WebSocket("ip");
75     websocket.on("open", getOn);'+
76 -- Fun  o get (thing)
77   ' \t function get'+
78   self.properties->collect(e | e.PropertyName)
79        ->iterate(i; acc : String = '' |
80      acc + i.toString()) + '(){
81       //var testando = websocket.send(message to server to return status);
82       var testando = websocket.send();
83             websocket.on("message", function message(msg) {
84             if(msg == 0){
85                 result=false;
86                 obterResult(result)
87             } else{
88                 result=true;
89                 obterResult(result)
90             }
91           });
92
93           function obterResult(){
94             return result;
95        }
96      //  return websocket.on("message", message());
97        return obterResult();
98
99      }\n' +
100 -- Fun  o Change(thing)
101      'function changedStatus(value){
```

```
102
103         //Mudar o status aqui
104        // websocket.send(message to change the state to server);
105           websocket.send();
106           return;
107        }})\n' +
108        '.catch(function(e){
109         console.log(e);
110        });';
111
112
113
114  helper context NODE!Properties def: toString(): String =
115    'properties: {' + '\n'+
116    '\t\t'+  self.PropertyName + ':{' + '\n'+
117    '\t\t\t'+  'type :"' + self.type + '",' + '\n'+
118    '\t\t\t'+ 'description:"'+ self.description + '"' + '\n' +
119    '\t\t\t}' +'\n' +
120      '},'
121    ;
122
123  helper context NODE!Action def: toString(): String =
124    'action: {' + '\n'+
125    '\t\t'+ self.ActionName + ':{' + '\n' +
126    '\t\t\t description:"'+   self.description + '"' + '\n' +
127    '\t\t } \n'+
128      '},'
129    ;
130  helper context NODE!Events def: toString(): String =
131    'event: {' + '\n'+
132      '\t' +'\t'+self.EventName + ':{' +'\n' +
133      '\t' +'\t'+'\t'+  'description:"'+ self.description + '"' + '\n' +
134      '\t' +'\t'+'}' +'\n'+
135      '},'
136    ;
137  helper context NODE!Function def: toString(): String =
138    -- Checking the function
139    if self.setPropertyHandler = 'true' then
140    'setPropertyHandler'
141    else ''
142    endif +
143    --Function writeProperty
```

```
144    if(self.writeProperty='true') then 'writeProperty'
145    else ''
146    endif+
147    -- Function Action
148    if(self.setActionHandler='true') then 'setActionHandler'
149    else ''
150    endif +
151    --Function Event
152    if(self.emitEvent='true') then 'emitEvent'
153    else ''
154    endif
155    ;
```

8.3 Regras de Transformações IoT para lingugaem C

```
1  -- @nsURI C=http://www.example.org/atividadeC
2  -- @nsURI IOT=http://www.example.org/novomodelIoT
3
4  module REGRAIOTDEVICE2C;
5  create OUT : C from IN : IOT;
6
7
8  rule Connect2Programa{
9  from e:IOT!ConnectedPhysicalEntity
10 to out:C!Programa(
11   name<-e.name
12 )
13   }
14 rule Websocket2WSsocket{
15 from e:IOT!WebSocket
16 to out:C!WSsocket(
17   dsnPort<-e.dsnPort,
18   httpPort<-e.httpPort,
19   wsport<-e.wsport,
20   ssid <- e.ssid,
21   password <- e.password,
22   msgChange <- e.msgChange,
23   msgStatus <- e.msgStatus,
24   programa<-e.controler.connectedphysicalentity
25 )
26   }
```

```
27
28 rule Pin2Main{
29 from e:IOT!Pin
30 to out:C!Main(
31   name<-e.pinPort,
32   programa<-e.controler.connectedphysicalentity
33 )
34   }
35 rule Component2Function{
36 from e:IOT!Component
37 to out:C!Function(
38   name<-e.nameComponent,
39   programa <- e.connectedphysicalentity
40
41 )
42
43 }
```

8.4 Regras de Transformações- IoT device para código da placa Esp8266

```
1  library BibliotecaC;
2
3
4  helper context C!Programa def: toString() : String =
5    '#include<Wifi.h>'+ '\n'+
6    '#include<ESP8266WiFi.h>'+ '\n' +
7    '#include<FS.h>'+ '\n'+
8    '#include<ESPAsyncWebServer.h>'+ '\n'+
9    '#include<WebSocketsServer.h>'+ '\n'
10    +
11     self.main->iterate(i; acc : String = '' |
12        acc + i.toString()) + '\n'+
13
14    --Conex o com websocket
15    self.wssocket -> iterate(i; acc : String = '' |
16        acc + i.toString()) +
17        '\n' +
18    'AsyncWebServer server('+
19      self.wssocket->collect(e | e.httpPort)
```

```
20          ->iterate(i; acc : String = '' |
21        acc + i.toString()) + ');\n'+
22        'char msg_buf[10]; '+
23
24
25    -- Iniciar a porta do websocket
26    'WebSocketsServer webSocket = WebSocketsServer(' +
27
28      self.wssocket->collect(e | e.wsport)
29          ->iterate(i; acc : String = '' |
30       acc + i.toString()) + ');\n'+
31
32  '//Call and Recive any WebSocket Message\n' +
33  'void onWebSocketEvent(uint8_t client_num,
34                         WStype_t type,
35                         uint8_t * payload,
36                         size_t length) {'+
37
38  --Fun  es de WebSocket
39
40  '// Figure out the type of WebSocket event
41    switch(type) {
42  '+
43  --Cliente desconectou
44  '
45      // Client has disconnected
46      case WStype_DISCONNECTED:
47        Serial.printf("[%u] Disconnected!\n", client_num);
48        break;
49   '+
50   --Cliente conectou
51   '
52      // New client has connected
53      case WStype_CONNECTED:
54        {
55          IPAddress ip = webSocket.remoteIP(client_num);
56          Serial.printf("[%u] Connection from ", client_num);
57          Serial.println(ip.toString());
58        }
59        break;
60     '+
61  --Recebendo menssagem do cliente
```

```
62      '
63        // Handle text messages from client
64        case WStype_TEXT:
65        // Print out raw message
66        Serial.printf("[%u] Received text: %s\n", client_num, payload);
67  '+
68   '      // Recive message
69         if ( strcmp((char *)payload, "' +
70
71        self.wssocket->collect(e | e.msgChange)
72      ->iterate(i; acc : String = '' |
73        acc + i.toString())
74           +'") == 0 ) { '+
75           self.function -> iterate(i; acc : String = '' |
76         acc + i.toString()) +
77         '\n'+
78        ' } '+
79  --Retornando o estado para o cliente(N o necessita
80  de um retorno em outra fun   o)
81
82  --Mensagem n o reconhecida
83       ' else {
84           Serial.println("[%u] Message not recognized");
85         }
86         break;'+
87  --Mensagens padr es do Websocket
88         '
89       // For everything else: do nothing
90       case WStype_BIN:
91       case WStype_ERROR:
92       case WStype_FRAGMENT_TEXT_START:
93       case WStype_FRAGMENT_BIN_START:
94       case WStype_FRAGMENT:
95       case WStype_FRAGMENT_FIN:
96       default:
97         break;
98     }
99  }'+
100 'void setup() {'+
101 --Init na thing
102 '
103
```

```
104   // Start Serial port
105   Serial.begin(115200);
106
107   // Start access point
108   WiFi.softAP(ssid, password);
109
110   // Print our IP address
111   Serial.println();
112   Serial.println("AP running");
113   Serial.print("My IP address: ");
114   Serial.println(WiFi.softAPIP());
115
116   // Start WebSocket server and assign callback
117   webSocket.begin();
118   webSocket.onEvent(onWebSocketEvent);
119 }
120
121 void loop() {
122
123   // Look for and handle WebSocket data
124   webSocket.loop();
125 }'  ;
126
127
128
129 helper context C!WSsocket def: toString() : String =
130   'const char *ssid = '+ self.ssid + ';\n'+
131   'const char *password = '+ self.password + ';\n' +
132   'const char *msg_toggle_led = "'+ self.msgChange + '";\n' +
133   'const char *msg_get_led = "'+ self.msgStatus + '";\n' +
134   'const int dns_port ='+ self.dsnPort +';\n' +
135   'const int http_port= '+ self.httpPort + ';\n' +
136   'const int ws_port ='+ self.wsport + ';\n'
137
138
139   ;
140 helper context C!Function def: toString(): String=
141
142   if self.name='led' then
143     'int led_state=0;\n
144       led_state = led_state ? 0 : 1;
145         Serial.printf("Toggling LED to %u\n", led_state);
```

```
146         digitalWrite(led_pin, led_state);
147         sprintf(msg_buf, "%d", led_state);
148         Serial.printf("Sending to [%u]: %s\n", client_num, msg_buf);
149         webSocket.sendTXT(client_num, msg_buf);
150         '
151
152   else
153     if self.name='temperatura' then
154     'int valorObtido = digitalRead(PIN_SENSOR);
155         float milivolts = (valorObtido/1024.0) * 3300;
156         float celsius = milivolts/10;
157         Serial.print("A temperatura    de = ");
158         Serial.println(celsius);
159
160         sprintf(msg_buf, "%f", celsius);
161
162         Serial.printf("Sending to [%u]: %s\n", client_num, msg_buf);
163         webSocket.sendTXT(client_num, msg_buf);
164  '
165     else
166       if self.name='pir' then
167       '
168               int sinal = digitalRead(PIN_SENSOR);
169               Serial.print("A presen a    de = ");
170               Serial.println(sinal);\n
171             sprintf(msg_buf, "%d", sinal);
172               Serial.printf("Sending to [%u]: %s\n", client_num, msg_buf);
173               webSocket.sendTXT(client_num, msg_buf);'
174       else
175         'N o existe codigo pra esse componente'
176   endif
177   endif
178   endif
179   ;
180
181 helper context C!Main def: toString(): String=
182   'const int PIN_SENSOR= ' + self.name + '; \n'   ;
```

8.5 Thing Description Led

```
1     "title": "MyLampLed",
```

```
2     "description": "A Thing to control the LED",
3     "properties": {
4         "on": {
5             "type": "boolean",
6             "description": "My Lamp led onOff",
7             "observable": true,
8             "readOnly": true,
9             "writeOnly": false,
10            "forms": [
11                {
12                    "href": "http://[2804:14d:8c88:8a32::3]:
13                    8080/MyLampLed/properties/on",
14                    "contentType": "application/json",
15                    "op": [
16                        "readproperty"
17                    ],
18                    "htv:methodName": "GET"
19                },
20                {
21                    "href": "http://[2804:14d:8c88:8a32::3]:
22                    8080/MyLampLed/properties/on/observable",
23                    "contentType": "application/json",
24                    "op": [
25                        "observeproperty"
26                    ],
27                    "subprotocol": "longpoll"
28                },
29                {
30                    "href": "http://[2804:14d:8c88:8a32:4946:b3d0:
31                    6534:2ac8]:8080/MyLampLed/properties/on",
32                    "contentType": "application/json",
33                    "op": [
34                        "readproperty"
35                    ],
36                    "htv:methodName": "GET"
37                },
38                {
39                    "href": "http://[2804:14d:8c88:8a32:4946:b3d0:6534:
40                   2ac8]:8080/MyLampLed/properties/on/observable",
41                    "contentType": "application/json",
42                    "op": [
43                        "observeproperty"
```

```
44                      ],
45                      "subprotocol": "longpoll"
46                  },
47                  {
48                      "href": "http://[2804:14d:8c88:8a32:e50d:b731:
49                      11ee:cc0b]:8080/MyLampLed/properties/on",
50                      "contentType": "application/json",
51                      "op": [
52                          "readproperty"
53                      ],
54                      "htv:methodName": "GET"
55                  },
56                  {
57                      "href": "http://[2804:14d:8c88:8a32:e50d:b731:11ee:
58                     cc0b]:8080/MyLampLed/properties/on/observable",
59                      "contentType": "application/json",
60                      "op": [
61                          "observeproperty"
62                      ],
63                      "subprotocol": "longpoll"
64                  },
65                  {
66                      "href": "http://192.168.0.27:8080/MyLampLed/
67                      properties/on",
68                      "contentType": "application/json",
69                      "op": [
70                          "readproperty"
71                      ],
72                      "htv:methodName": "GET"
73                  },
74                  {
75                      "href": "http://192.168.0.27:8080/MyLampLed/
76                      properties/on/observable",
77                      "contentType": "application/json",
78                      "op": [
79                          "observeproperty"
80                      ],
81                      "subprotocol": "longpoll"
82                  },
83                  {
84                      "href": "coap://[2804:14d:8c88:8a32::3]:5683/MyLampLed/
85                      properties/on",
```

```
86                      "contentType": "application/json",
87                      "op": [
88                          "readproperty"
89                      ]
90                  },
91                  {
92                      "href": "coap://[2804:14d:8c88:8a32:4946:b3d0:
93                      6534:2ac8]:5683/MyLampLed/properties/on",
94                      "contentType": "application/json",
95                      "op": [
96                          "readproperty"
97                      ]
98                  },
99                  {
100                     "href": "coap://[2804:14d:8c88:8a32:e50d:b731:11ee:
101                     cc0b]:5683/MyLampLed/properties/on",
102                     "contentType": "application/json",
103                     "op": [
104                         "readproperty"
105                     ]
106                 },
107                 {
108                     "href": "coap://192.168.0.27:5683/MyLampLed/
109                     properties/on",
110                     "contentType": "application/json",
111                     "op": [
112                         "readproperty"
113                     ]
114                 }
115             ]
116         }
117     },
118     "actions": {
119         "toggle": {
120             "description": "current status of the lamp(on|off)",
121             "forms": [
122                 {
123                     "href": "http://[2804:14d:8c88:8a32::3]:8080/
124                     MyLampLed/actions/toggle",
125                     "contentType": "application/json",
126                     "op": [
127                         "invokeaction"
```

```
128                     ],
129                     "htv:methodName": "POST"
130                 },
131                 {
132                     "href": "http://[2804:14d:8c88:8a32:4946:b3d0:
133                     6534:2ac8]:8080/MyLampLed/actions/toggle",
134                     "contentType": "application/json",
135                     "op": [
136                         "invokeaction"
137                     ],
138                     "htv:methodName": "POST"
139                 },
140                 {
141                     "href": "http://[2804:14d:8c88:8a32:e50d:b731:11ee:
142                     cc0b]:8080/MyLampLed/actions/toggle",
143                     "contentType": "application/json",
144                     "op": [
145                         "invokeaction"
146                     ],
147                     "htv:methodName": "POST"
148                 },
149                 {
150                     "href": "http://192.168.0.27:8080/MyLampLed/
151                     actions/toggle",
152                     "contentType": "application/json",
153                     "op": [
154                         "invokeaction"
155                     ],
156                     "htv:methodName": "POST"
157                 },
158                 {
159                     "href": "coap://[2804:14d:8c88:8a32::3]:5683/
160                     MyLampLed/actions/toggle",
161                     "contentType": "application/json",
162                     "op": "invokeaction"
163                 },
164                 {
165                     "href": "coap://[2804:14d:8c88:8a32:4946:b3d0:
166                     6534:2ac8]:5683/MyLampLed/actions/toggle",
167                     "contentType": "application/json",
168                     "op": "invokeaction"
169                 },
```

```
170                 {
171                     "href": "coap://[2804:14d:8c88:8a32:e50d:b731:11ee
172                     :cc0b]:5683/MyLampLed/actions/toggle",
173                     "contentType": "application/json",
174                     "op": "invokeaction"
175                 },
176                 {
177                     "href": "coap://192.168.0.27:5683/MyLampLed/
178                     actions/toggle",
179                     "contentType": "application/json",
180                     "op": "invokeaction"
181                 }
182             ],
183             "idempotent": false,
184             "safe": false
185         }
186     },
187     "events": {
188         "overheat": {
189             "description": "Alert sent when the
190             Led temperature is too high",
191             "forms": [
192                 {
193                     "href": "http://[2804:14d:8c88:8a32::3]:8080/
194                     MyLampLed/events/overheat",
195                     "contentType": "application/json",
196                     "subprotocol": "longpoll",
197                     "op": [
198                         "subscribeevent"
199                     ]
200                 },
201                 {
202                     "href": "http://[2804:14d:8c88:8a32:4946:b3d0:
203                     6534:2ac8]:8080/MyLampLed/events/overheat",
204                     "contentType": "application/json",
205                     "subprotocol": "longpoll",
206                     "op": [
207                         "subscribeevent"
208                     ]
209                 },
210                 {
211                     "href": "http://[2804:14d:8c88:8a32:e50d:b731:11ee:
```

```
212                     cc0b]:8080/MyLampLed/events/overheat",
213                     "contentType": "application/json",
214                     "subprotocol": "longpoll",
215                     "op": [
216                         "subscribeevent"
217                     ]
218                 },
219                 {
220                     "href": "http://192.168.0.27:8080/MyLampLed/events/
221                     overheat",
222                     "contentType": "application/json",
223                     "subprotocol": "longpoll",
224                     "op": [
225                         "subscribeevent"
226                     ]
227                 },
228                 {
229                     "href": "ws://[2804:14d:8c88:8a32::3]:8080/MyLampLed/
230                     events/overheat",
231                     "contentType": "application/json",
232                     "op": "subscribeevent"
233                 },
234                 {
235                     "href": "ws://[2804:14d:8c88:8a32:4946:b3d0:6534:2ac8]:
236                     8080/MyLampLed/events/overheat",
237                     "contentType": "application/json",
238                     "op": "subscribeevent"
239                 },
240                 {
241                     "href": "ws://[2804:14d:8c88:8a32:e50d:b731:
242                     11ee:cc0b]:8080/MyLampLed/events/overheat",
243                     "contentType": "application/json",
244                     "op": "subscribeevent"
245                 },
246                 {
247                     "href": "ws://192.168.0.27:8080/MyLampLed/events/
248                     overheat",
249                     "contentType": "application/json",
250                     "op": "subscribeevent"
251                 },
252                 {
253                     "href": "coap://[2804:14d:8c88:8a32::3]:5683/MyLampLed/
```

```
254                     events/overheat",
255                     "contentType": "application/json",
256                     "op": "subscribeevent"
257                 },
258                 {
259                     "href": "coap://[2804:14d:8c88:8a32:4946:
260                     b3d0:6534:2ac8]:5683/MyLampLed/events/overheat",
261                     "contentType": "application/json",
262                     "op": "subscribeevent"
263                 },
264                 {
265                     "href": "coap://[2804:14d:8c88:8a32:e50d:
266                     b731:11ee:cc0b]:5683/MyLampLed/events/overheat",
267                     "contentType": "application/json",
268                     "op": "subscribeevent"
269                 },
270                 {
271                     "href": "coap://192.168.0.27:5683/MyLampLed/
272                     events/overheat",
273                     "contentType": "application/json",
274                     "op": "subscribeevent"
275                 }
276             ]
277         }
278     },
279     "@context": "https://www.w3.org/2019/wot/td/v1",
280     "@type": "Thing",
281     "security": [
282         "nosec_sc"
283     ],
284     "id": "urn:uuid:b3f2487b-eec4-43c5-a339-84e6267dff4d",
285     "forms": [
286         {
287             "href": "http://[2804:14d:8c88:8a32::3]:8080/MyLampLed/
288             all/properties",
289             "contentType": "application/json",
290             "op": [
291                 "readallproperties",
292                 "readmultipleproperties"
293             ]
294         },
295         {
```

```
296             "href": "http://[2804:14d:8c88:8a32:4946:b3d0:6534:2ac8]:
297             8080/MyLampLed/all/properties",
298             "contentType": "application/json",
299             "op": [
300                 "readallproperties",
301                 "readmultipleproperties"
302             ]
303         },
304         {
305             "href": "http://[2804:14d:8c88:8a32:e50d:b731:11ee:cc0b]:
306             8080/MyLampLed/all/properties",
307             "contentType": "application/json",
308             "op": [
309                 "readallproperties",
310                 "readmultipleproperties"
311             ]
312         },
313         {
314             "href": "http://192.168.0.27:8080/MyLampLed/all/properties",
315             "contentType": "application/json",
316             "op": [
317                 "readallproperties",
318                 "readmultipleproperties"
319             ]
320         }
321     ],
322     "securityDefinitions": {
323         "nosec_sc": {
324             "scheme": "nosec"
325         }
326     }
327 }
```

8.6 Thing Description Temperatura

```
1  {
2      "title": "TemperatureController",
3      "description": "A Thing to control the temperature
4      of the room and also get alerts in too high temperatures",
5      "properties": {
6          "temperature": {
```

```
7               "type": "integer",
8               "description": "My Current temperature value",
9               "observable": true,
10              "readOnly": true,
11              "unit": "Celcius",
12              "writeOnly": false,
13              "forms": [
14                  {
15                      "href": "http://[2804:14d:8c88:8a32::9]:8080/
16                      TemperatureController/properties/temperature",
17                      "contentType": "application/json",
18                      "op": [
19                          "readproperty"
20                      ],
21                      "htv:methodName": "GET"
22                  },
23                  {
24                      "href": "http://[2804:14d:8c88:8a32::9]:
25                      8080/TemperatureController/properties/
26                      temperature/observable",
27                      "contentType": "application/json",
28                      "op": [
29                          "observeproperty"
30                      ],
31                      "subprotocol": "longpoll"
32                  },
33                  {
34                      "href": "http://[2804:14d:8c88:8a32:4946:b3d0:6534:
35                      2ac8]:8080/TemperatureController/properties/
36                      temperature",
37                      "contentType": "application/json",
38                      "op": [
39                          "readproperty"
40                      ],
41                      "htv:methodName": "GET"
42                  },
43                  {
44                      "href": "http://[2804:14d:8c88:8a32:4946:b3d0:
45                      6534:2ac8]:8080/TemperatureController/
46                      properties/temperature/observable",
47                      "contentType": "application/json",
48                      "op": [
```

```
49                          "observeproperty"
50                      ],
51                      "subprotocol": "longpoll"
52                  },
53                  {
54                      "href": "http://[2804:14d:8c88:8a32:34f2:f927:
55                     d84:6f5d]:8080/TemperatureController/
56                     properties/temperature",
57                      "contentType": "application/json",
58                      "op": [
59                          "readproperty"
60                      ],
61                      "htv:methodName": "GET"
62                  },
63                  {
64                      "href": "http://[2804:14d:8c88:8a32:34f2:f927:
65                   d84:6f5d]:8080/TemperatureController/properties/
66                   temperature/observable",
67                      "contentType": "application/json",
68                      "op": [
69                          "observeproperty"
70                      ],
71                      "subprotocol": "longpoll"
72                  },
73                  {
74                      "href": "http://192.168.0.92:8080/
75                      TemperatureController/properties/temperature",
76                      "contentType": "application/json",
77                      "op": [
78                          "readproperty"
79                      ],
80                      "htv:methodName": "GET"
81                  },
82                  {
83                      "href": "http://192.168.0.92:8080/
84                      TemperatureController/properties/
85                      temperature/observable",
86                  "contentType": "application/json",
87                      "op": [
88                          "observeproperty"
89                      ],
90                      "subprotocol": "longpoll"
```

```
91                  },
92                  {
93                      "href": "coap://[2804:14d:8c88:8a32::9]:
94              5683/TemperatureController/properties/temperature",
95                      "contentType": "application/json",
96                      "op": [
97                          "readproperty"
98                      ]
99                  },
100                 {
101                     "href": "coap://[2804:14d:8c88:8a32:4946:b3d0:6534:
102         2ac8]:5683/TemperatureController/properties/temperature",
103                     "contentType": "application/json",
104                     "op": [
105                         "readproperty"
106                     ]
107                 },
108                 {
109                     "href": "coap://[2804:14d:8c88:8a32:34f2:f927:
110     d84:6f5d]:5683/TemperatureController/properties/temperature",
111                     "contentType": "application/json",
112                     "op": [
113                         "readproperty"
114                     ]
115                 },
116                 {
117                     "href": "coap://192.168.0.92:5683/Temperature
118                     Controller/properties/temperature",
119                     "contentType": "application/json",
120                     "op": [
121                         "readproperty"
122                     ]
123                 }
124             ]
125         }
126     },
127     "actions": {
128         "toggle": {
129             "description": "current status of the temperature",
130             "forms": [
131                 {
132                     "href": "http://[2804:14d:8c88:8a32::9]:8080/
```

```
133                     TemperatureController/actions/toggle",
134                     "contentType": "application/json",
135                     "op": [
136                         "invokeaction"
137                     ],
138                     "htv:methodName": "POST"
139                 },
140                 {
141                     "href": "http://[2804:14d:8c88:8a32:4946:b3d0:6534:
142                   2ac8]:8080/TemperatureController/actions/toggle",
143                     "contentType": "application/json",
144                     "op": [
145                         "invokeaction"
146                     ],
147                     "htv:methodName": "POST"
148                 },
149                 {
150                     "href": "http://[2804:14d:8c88:8a32:34f2:f927:d84:6f5d]:
151                     8080/TemperatureController/actions/toggle",
152                     "contentType": "application/json",
153                     "op": [
154                         "invokeaction"
155                     ],
156                     "htv:methodName": "POST"
157                 },
158                 {
159                     "href": "http://192.168.0.92:8080/
160                     TemperatureController/actions/toggle",
161                     "contentType": "application/json",
162                     "op": [
163                         "invokeaction"
164                     ],
165                     "htv:methodName": "POST"
166                 },
167                 {
168                     "href": "coap://[2804:14d:8c88:8a32::9]:
169                     5683/TemperatureController/actions/toggle",
170                     "contentType": "application/json",
171                     "op": "invokeaction"
172                 },
173                 {
174                     "href": "coap://[2804:14d:8c88:8a32:4946:b3d0:
```

```
175                 6534:2ac8]:5683/TemperatureController/
176                 actions/toggle",
177                     "contentType": "application/json",
178                     "op": "invokeaction"
179                 },
180                 {
181                     "href": "coap://[2804:14d:8c88:8a32:34f2:f927:d84:6f5d]:
182                     5683/TemperatureController/actions/toggle",
183                     "contentType": "application/json",
184                     "op": "invokeaction"
185                 },
186                 {
187                     "href": "coap://192.168.0.92:5683/
188                     TemperatureController/actions/toggle",
189                     "contentType": "application/json",
190                     "op": "invokeaction"
191                 }
192             ],
193             "idempotent": false,
194             "safe": false
195         }
196     },
197     "events": {
198         "overheat": {
199             "description": "Alert sent when
200             the room temperature is too high",
201             "forms": [
202                 {
203                     "href": "http://[2804:14d:8c88:8a32::9]:
204                     8080/TemperatureController/events/overheat",
205                     "contentType": "application/json",
206                     "subprotocol": "longpoll",
207                     "op": [
208                         "subscribeevent"
209                     ]
210                 },
211                 {
212                     "href": "http://[2804:14d:8c88:8a32:
213 4946:b3d0:6534:2ac8]:8080/TemperatureController/events/overheat",
214                     "contentType": "application/json",
215                     "subprotocol": "longpoll",
216                     "op": [
```

```
217                         "subscribeevent"
218                     ]
219                 },
220                 {
221                     "href": "http://[2804:14d:8c88:8a32:34f2:f927:
222         d84:6f5d:8080/TemperatureController/events/overheat",
223                     "contentType": "application/json",
224                     "subprotocol": "longpoll",
225                     "op": [
226                         "subscribeevent"
227                     ]
228                 },
229                 {
230                     "href": "http://192.168.0.92:8080/
231             TemperatureControlle/events/overheat",
232                     "contentType": "application/json",
233                     "subprotocol": "longpoll",
234                     "op": [
235                         "subscribeevent"
236                     ]
237                 },
238                 {
239                     "href": "ws://[2804:14d:8c88:8a32::9]:
240         8080/TemperatureController/events/overheat",
241                     "contentType": "application/json",
242                     "op": "subscribeevent"
243                 },
244                 {
245                     "href":
246             "ws://[2804:14d:8c88:8a32:4946:b3d0:6534:2ac8]:
247             8080/TemperatureController/events/overheat",
248                     "contentType": "application/json",
249                     "op": "subscribeevent"
250                 },
251                 {
252                     "href"
253         "ws://[2804:14d:8c88:8a32:34f2:f927:d84:6f5d]:
254         8080/TemperatureController/events/overheat",
255                     "contentType": "application/json",
256                     "op": "subscribeevent"
257                 },
258                 {
```

```
259                     "href": "ws://192.168.0.92:8080/TemperatureController/
260                     events/overheat",
261                     "contentType": "application/json",
262                     "op": "subscribeevent"
263                 },
264                 {
265                     "href": "coap://[2804:14d:8c88:8a32::9]:5683/
266                     TemperatureController/events/overheat",
267                     "contentType": "application/json",
268                     "op": "subscribeevent"
269                 },
270                 {
271                     "href": "coap://[2804:14d:8c88:8a32:4946:
272     b3d0:6534:2ac8]:5683/TemperatureController/events/overheat",
273                     "contentType": "application/json",
274                     "op": "subscribeevent"
275                 },
276                 {
277                     "href": "coap://[2804:14d:8c88:8a32:34f2:f927:d84:6f5d]:
278                     5683/TemperatureController/events/overheat",
279                     "contentType": "application/json",
280                     "op": "subscribeevent"
281                 },
282                 {
283                     "href": "coap://192.168.0.92:5683/
284                     TemperatureController/events/overheat",
285                     "contentType": "application/json",
286                     "op": "subscribeevent"
287                 }
288             ]
289         }
290     },
291     "@context": "https://www.w3.org/2019/wot/td/v1",
292     "@type": "Thing",
293     "security": [
294         "nosec_sc"
295     ],
296     "id": "urn:uuid:af4abadc-84e6-4c8e-b401-68ff27dd9c38",
297     "forms": [
298         {
299             "href": "http://[2804:14d:8c88:8a32::9]:8080/
300             TemperatureController/all/properties",
```

```
301             "contentType": "application/json",
302             "op": [
303                 "readallproperties",
304                 "readmultipleproperties"
305             ]
306         },
307         {
308             "href": "http://[2804:14d:8c88:8a32:4946:b3d0:6534:2ac8]:
309             8080/TemperatureController/all/properties",
310             "contentType": "application/json",
311             "op": [
312                 "readallproperties",
313                 "readmultipleproperties"
314             ]
315         },
316         {
317             "href": "http://[2804:14d:8c88:8a32:34f2:f927:d84:6f5d]:
318             8080/TemperatureController/all/properties",
319             "contentType": "application/json",
320             "op": [
321                 "readallproperties",
322                 "readmultipleproperties"
323             ]
324         },
325         {
326             "href": "http://192.168.0.92:8080/TemperatureController/
327             all/properties",
328             "contentType": "application/json",
329             "op": [
330                 "readallproperties",
331                 "readmultipleproperties"
332             ]
333         }
334     ],
335     "securityDefinitions": {
336         "nosec_sc": {
337             "scheme": "nosec"
338         }
339     }
340 }
```

8.7 Thing Description Pir

```
{
    "title": "MyPirSensor",
    "description": "A Thing to get sensor presence",
    "properties": {
        "pirsensor": {
            "type": "boolean",
            "description": "My sensor true or false",
            "observable": true,
            "readOnly": true,
            "writeOnly": false,
            "forms": [
                {
                    "href": "http://[2804:14d:8c88:8a32::9]:
                    8080/MyPirSensor/properties/pirsensor",
                    "contentType": "application/json",
                    "op": [
                        "readproperty"
                    ],
                    "htv:methodName": "GET"
                },
                {
                    "href": "http://[2804:14d:8c88:8a32::9]:
            8080/MyPirSensor/properties/pirsensor/observable",
                    "contentType": "application/json",
                    "op": [
                        "observeproperty"
                    ],
                    "subprotocol": "longpoll"
                },
                {
                    "href": "http://[2804:14d:8c88:8a32:4946:b3d0:6534
            :2ac8]:8080/MyPirSensor/properties/pirsensor",
                    "contentType": "application/json",
                    "op": [
                        "readproperty"
                    ],
                    "htv:methodName": "GET"
                },
                {
                    "href": "http://[2804:14d:8c88:8a32:4946:
```

```
41      b3d0:6534:2ac8]:8080/MyPirSensor/properties/
42      pirsensor/observable",
43                  "contentType": "application/json",
44                  "op": [
45                      "observeproperty"
46                  ],
47                  "subprotocol": "longpoll"
48              },
49              {
50                  "href": "http://[2804:14d:8c88:8a32:34f2:f927:d84:6f5d]:
51                  8080/MyPirSensor/properties/pirsensor",
52                  "contentType": "application/json",
53                  "op": [
54                      "readproperty"
55                  ],
56                  "htv:methodName": "GET"
57              },
58              {
59                  "href": "http://[2804:14d:8c88:8a32:34f2:f927:d84:6f5d
60          :8080/MyPirSensor/properties/pirsensor/
61                  observable",
62                  "contentType": "application/json",
63                  "op": [
64                      "observeproperty"
65                  ],
66                  "subprotocol": "longpoll"
67              },
68              {
69                  "href": "http://192.168.0.92:8080/
70              MyPirSensor/properties/pirsensor",
71                  "contentType": "application/json",
72                  "op": [
73                      "readproperty"
74                  ],
75                  "htv:methodName": "GET"
76              },
77              {
78                  "href": "http://192.168.0.92:8080/MyPirSensor/
79                  properties/pirsensor/observable",
80                  "contentType": "application/json",
81                  "op": [
82                      "observeproperty"
```

```
83                      ],
84                      "subprotocol": "longpoll"
85                  },
86                  {
87                      "href": "coap://[2804:14d:8c88:8a32::9]:5683
88                      /MyPirSensor/properties/pirsensor",
89                      "contentType": "application/json",
90                      "op": [
91                          "readproperty"
92                      ]
93                  },
94                  {
95                      "href": "coap://[2804:14d:8c88:8a32:4946:b3d0:
96              6534:2ac8]:5683/MyPirSensor/properties/pirsensor",
97                      "contentType": "application/json",
98                      "op": [
99                          "readproperty"
100                     ]
101                 },
102                 {
103                     "href": "coap://[2804:14d:8c88:8a32:34f2:f927:d84:6f5d]:
104                     5683/MyPirSensor/properties/pirsensor",
105                     "contentType": "application/json",
106                     "op": [
107                         "readproperty"
108                     ]
109                 },
110                 {
111                     "href": "coap://192.168.0.92:5683/
112                 MyPirSensor/properties/pirsensor",
113                     "contentType": "application/json",
114                     "op": [
115                         "readproperty"
116                     ]
117                 }
118             ]
119         }
120     },
121     "actions": {
122         "toggle": {
123             "description": "No action needed",
124             "forms": [
```

```
125                 {
126                     "href": "http://[2804:14d:8c88:8a32::9]:
127                     8080/MyPirSensor/actions/toggle",
128                     "contentType": "application/json",
129                     "op": [
130                         "invokeaction"
131                     ],
132                     "htv:methodName": "POST"
133                 },
134                 {
135                     "href": "http://[2804:14d:8c88:8a32:4946:b3d0:
136                     6534:2ac8]:8080/MyPirSensor/actions/toggle",
137                     "contentType": "application/json",
138                     "op": [
139                         "invokeaction"
140                     ],
141                     "htv:methodName": "POST"
142                 },
143                 {
144                     "href": "http://[2804:14d:8c88:8a32:34f2:f927:
145                     d84:6f5d]:8080/MyPirSensor/actions/toggle",
146                     "contentType": "application/json",
147                     "op": [
148                         "invokeaction"
149                     ],
150                     "htv:methodName": "POST"
151                 },
152                 {
153                     "href": "http://192.168.0.92:8080/MyPirSensor/
154                     actions/toggle",
155                     "contentType": "application/json",
156                     "op": [
157                         "invokeaction"
158                     ],
159                     "htv:methodName": "POST"
160                 },
161                 {
162                     "href": "coap://[2804:14d:8c88:8a32::9]:
163                     5683/MyPirSensor/actions/toggle",
164                     "contentType": "application/json",
165                     "op": "invokeaction"
166                 },
```

```
167                 {
168                     "href": "coap://[2804:14d:8c88:8a32:4946:b3d0:
169                     6534:2ac8]:5683/MyPirSensor/actions/toggle",
170                     "contentType": "application/json",
171                     "op": "invokeaction"
172                 },
173                 {
174                     "href": "coap://[2804:14d:8c88:8a32:34f2:f927:d84:
175             6f5d]:5683/MyPirSensor/actions/toggle",
176                     "contentType": "application/json",
177                     "op": "invokeaction"
178                 },
179                 {
180                     "href": "coap://192.168.0.92:5683/
181                 MyPirSensor/actions/toggle",
182                     "contentType": "application/json",
183                     "op": "invokeaction"
184                 }
185             ],
186             "idempotent": false,
187             "safe": false
188         }
189     },
190     "events": {
191         "no": {
192             "description": "No event specific",
193             "forms": [
194                 {
195                     "href": "http://[2804:14d:8c88:8a32::9]:
196                 8080/MyPirSensor/events/no",
197                     "contentType": "application/json",
198                     "subprotocol": "longpoll",
199                     "op": [
200                         "subscribeevent"
201                     ]
202                 },
203                 {
204                     "href": "http://[2804:14d:8c88:8a32:
205             4946:b3d0:6534:2ac8]:8080/MyPirSensor/events/no",
206                     "contentType": "application/json",
207                     "subprotocol": "longpoll",
208                     "op": [
```

```
209                         "subscribeevent"
210                     ]
211                 },
212                 {
213                     "href": "http://[2804:14d:8c88:
214         8a32:34f2:f927:d84:6f5d:8080/MyPirSensor/events/no",
215                     "contentType": "application/json",
216                     "subprotocol": "longpoll",
217                     "op": [
218                         "subscribeevent"
219                     ]
220                 },
221                 {
222                     "href": "http://192.168.0.92:8080/
223                     MyPirSensor/events/no",
224                     "contentType": "application/json",
225                     "subprotocol": "longpoll",
226                     "op": [
227                         "subscribeevent"
228                     ]
229                 },
230                 {
231                     "href": "ws://[2804:14d:8c88:8a32::9]:8080/
232                     MyPirSensor/events/no",
233                     "contentType": "application/json",
234                     "op": "subscribeevent"
235                 },
236                 {
237                     "href": "ws://[2804:14d:8c88:8a32:4946:b3d0:6534:2ac8]:
238                     8080/MyPirSensor/events/ev",
239                     "contentType": "application/json",
240                     "op": "subscribeevent"
241                 },
242                 {
243                     "href": "ws://[2804:14d:8c88:8a32:34f2:f927:d84:6f5d]:
244                     8080/MyPirSensor/events/no",
245                     "contentType": "application/json",
246                     "op": "subscribeevent"
247                 },
248                 {
249                     "href": "ws://192.168.0.92:8080/MyPirSensor/events/no",
250                     "contentType": "application/json",
```

```
251                     "op": "subscribeevent"
252                 },
253                 {
254                     "href": "coap://[2804:14d:8c88:8a32::9]:5683/
255                     /events/no",
256                     "contentType": "application/json",
257                     "op": "subscribeevent"
258                 },
259                 {
260                     "href": "coap://[2804:14d:8c88:8a32:4946
261         :b3d0:6534:2ac8]:5683/MyPirSensor/events/no",
262                     "contentType": "application/json",
263                     "op": "subscribeevent"
264                 },
265                 {
266                     "href": "coap://[2804:14d:8c88:8a32:34f2:
267             f927:d84:6f5d]:5683/MyPirSensor/events/no",
268                     "contentType": "application/json",
269                     "op": "subscribeevent"
270                 },
271                 {
272                     "href": "coap://192.168.0.92:5683/
273             MyPirSensor/events/no",
274                     "contentType": "application/json",
275                     "op": "subscribeevent"
276                 }
277             ]
278         }
279     },
280     "@context": "https://www.w3.org/2019/wot/td/v1",
281     "@type": "Thing",
282     "security": [
283         "nosec_sc"
284     ],
285     "id": "urn:uuid:3fa26d65-f555-4c42-8bda-545ffecfb11f",
286     "forms": [
287         {
288             "href": "http://[2804:14d:8c88:8a32::9]:8080/MyPirSensor/
289             all/properties",
290             "contentType": "application/json",
291             "op": [
292                 "readallproperties",
```

```
293                 "readmultipleproperties"
294             ]
295         },
296         {
297             "href": "http://[2804:14d:8c88:8a32:4946:b3d0:6534:2ac8]:8080/
298             MyPirSensor/all/properties",
299             "contentType": "application/json",
300             "op": [
301                 "readallproperties",
302                 "readmultipleproperties"
303             ]
304         },
305         {
306             "href": "http://[2804:14d:8c88:8a32:34f2:f927:d84:6f5d]:8080/
307             MyPirSensor/all/properties",
308             "contentType": "application/json",
309             "op": [
310                 "readallproperties",
311                 "readmultipleproperties"
312             ]
313         },
314         {
315             "href": "http://192.168.0.92:8080/MyPirSensor/all/properties",
316             "contentType": "application/json",
317             "op": [
318                 "readallproperties",
319                 "readmultipleproperties"
320             ]
321         }
322     ],
323     "securityDefinitions": {
324         "nosec_sc": {
325             "scheme": "nosec"
326         }
327     }
328 }
```