Capítulo 6

Do device à cloud com a Plataforma SOFT-IoT: sua infraestrutura IoT em poucas horas

Leandro Andrade[1], Cleber Lira[3,1], Brenno Mello[1], Andressa Andrade[1], Antonio Coutinho[2,1], Fabíola Greve[1], Cássio Prazeres[1]

[1]Universidade Federal da Bahia. Departamento de Ciência da Computação (UFBA/DCC)

[2]Universidade Estadual de Feira de Santana. Departamento de Tecnologia (UEFS/DTEC)

[3] Instituto Federal de Educação, Ciência e Tecnologia da Bahia. (IFBA)

(leandrojsa, brenno.mello, fabiola, prazeres)@ufba.br,

dsandrade@dcc.ufba.br, cleberlira@ifba.edu.br, acoutinho@uefs.br

Abstract

The Internet of Things (IoT) is playing a great role in the technology scenario due to its high potential and impact on different society segments. Currently, several IoT initiatives have simultaneously developed new architectures, platforms and applications which has resulted in the creation of several parallel IoT ecosystems. This diversity makes access to IoT devices, and their integration in potential applications, difficult. In this chapter, we present the SOFT-IoT platform, which introduces the Fog of Things (FoT) concept in order to exploit the processing, storage and network capacity of local resources, allowing for the integration of different devices in a seamless IoT architecture.

Resumo

A Internet das Coisas (Internet of Things - IoT) vem desempenhando um importante papel no cenário tecnológico devido ao seu alto potencial e impacto em diferentes segmentos da sociedade. Recentemente, diferentes iniciativas em IoT desenvolveram simultaneamente novas arquiteturas, plataformas e aplicações, o que resultou na criação de diversos ecossistemas IoT paralelos. Essa diversidade dificulta o acesso a dispositivos IoT e sua integração em potenciais aplicações. Neste capítulo, apresentamos a plataforma SOFT-IoT, que introduz o conceito de Névoa das Coisas (Fog of Things - FoT) visando explorar o processamento, o armazenamento e a capacidade de rede dos recursos locais, permitindo a integração de diferentes dispositivos em uma arquitetura IoT estruturada.

6.1. Introdução

A comunidade acadêmica e a indústria têm feito significantes progressos em Internet das Coisas (*Internet of Things* – IoT) e áreas correlatas, em diferentes direções, incluindo o desenvolvimento de novas arquiteturas, plataformas e aplicações. Entretanto, esses progressos têm resultado na criação de diversos ecossistemas IoT paralelos, dificultando a criação de um ecossistema global, que permitiria habilitar o acesso a um vasto número de dispositivos existentes (e futuros), aumentando, dessa forma, a quantidade e o escopo de potenciais aplicações IoT [Sundmaeker et al. 2010]. Assim, o IERC (*European Research Cluster on the Internet of Things*) publicou um relatório contendo um conjunto de desafios de pesquisa, melhores práticas, recomendações e próximos passos na direção de prover interoperabilidade na Internet das Coisas [Serrano et al. 2015a]. Nesse relatório, os autores apontam que interoperabilidade e a entrega de serviços e de dados são importantes desafios que devem ser tratados por novas soluções para a Internet das Coisas.

Por um lado, interoperabilidade em IoT deve garantir meios de habilitar anotação semântica, identificação, registro e descoberta de dados provenientes dos dispositivos na IoT [Serrano et al. 2015b]. Por outro lado, serviços IoT devem garantir a entrega e utilização desses dados a usuários, aplicações ou mesmo outros serviços [Bassi et al. 2013]. Diversas soluções baseadas em Computação em Nuvem (*Cloud Computing*) têm sido propostas pela indústria e na academia com foco em entrega de serviços e interoperabilidade. Entretanto, soluções baseadas em nuvem apresentam algumas limitações [Abdelshkour 2015]: conectividade com a Internet é um pré-requisito essencial; dados e aplicações são processados na nuvem, tarefas que consomem muito tempo em se tratando de grandes volumes de dados; muito uso da largura de banda, pois é preciso enviar dados coletados de dispositivos IoT por segundos para a nuvem; tempo de resposta alto e problemas de escalabilidade, dado a dependência de servidores localizados em locais remotos.

Um novo conceito de infraestrutura para dados e serviços foi proposto inicialmente pela CISCO [Bonomi et al. 2012][Bonomi et al. 2014]: Computação em Névoa (*Fog Computing*, *Fog Networking* ou *Fogging*). Na IoT, a Computação em Névoa visa tirar parte da complexidade da nuvem e trazer para perto dos dispositivos, aplicações e/ou usuários, funcionando como uma espécie de "nuvem local privada" (névoa). Portanto, a Computação em Névoa propõe que [Abdelshkour 2015]: a conectividade ininterrupta com a Internet não seja essencial; o processamento de dados e a execução de aplicações também possam ocorrer nos limites da rede local; o uso da largura de banda seja otimizado; a melhora do tempo de resposta e da escalabilidade aconteça através do emprego de "pequenos servidores" locais próximos aos dispositivos e/ou usuários; e outros.

A Computação em Nuvem e a Computação em Névoa não são mutuamente exclusivas. De fato, em IoT, elas são complementares em diversos aspectos como, por exemplo: dispositivos (atuadores e sensores) continuam funcionando na névoa mesmo sem conectividade e quando a conectividade for possível, dados são enviados para a nuvem; antes de serem enviados à nuvem os dados são processados em servidores locais na névoa; análises de dados podem ser realizados na névoa e visualizados na nuvem ou realizados na nuvem e executados (atuadores) na névoa; usuários/aplicações locais podem acessar dispositivos e dados diretamente via névoa e usuários/aplicações remotos via nuvem, provendo, dessa forma, melhor qualidade de experiência (*Quality of Experience* - QoE); dentre outros.

Nesse contexto, este capítulo apresenta a plataforma SOFT-IoT (*Self-Organizing Fog of Things for the Internet of Things*), que provê soluções relacionadas à interoperabilidade e entrega de serviços na Internet das Coisas, desde a borda da rede até a nuvem. Para prover essas soluções, a plataforma SOFT-IoT apresenta o paradigma da "Névoa de Coisas" (*Fog of Things* - FoT) auto-organizável para entrega de serviços e produção de dados interoperáveis na borda da rede.

6.2. Plataforma SOFT-IoT

Com o objetivo de aproveitar as vantagens que a Computação em Névoa pode propiciar para IoT, em [Prazeres and Serrano 2016] é apresentado um novo paradigma chamado de *Fog of Things* (FoT). Dessa forma, como na Computação em Névoa, o FoT permite que parte da capacidade de processamento de dados e operações de entrega de serviços sejam processados em *gateways* (pequenos servidores) e/ou em servidores, ambos na borda da rede. O paradigma FoT vai além da Computação em Névoa nos seguintes aspectos:

- Explora a capacidade de processamento da borda da rede por meio do processamento de dados e da entrega de serviços em dispositivos, *gateways* IoT e/ou servidores locais;
- Define serviços IoT na borda da rede;
- Distribui esses serviços na borda da rede através de um *middleware* orientado para mensagens e serviços.

A Figura 6.1 ilustra uma plataforma baseada no paradigma *Fog of Things*, que é composta por uma combinação ou pela totalidade dos seguintes componentes: aplicações; dispositivos; *gateways*; servidores; *middleware* orientado a serviços de mensagens; e provedores de segurança. Esses componentes formam o núcleo de organização de uma FoT e são descritos em detalhes a seguir:

- *FoT-Device* - reusa e estende o conceito proposto por [Bassi et al. 2013], em que os dispositivos IoT são artefatos físicos que realizam a integração do mundo real com o mundo digital da Internet. Diante disso, no trabalho de [Bassi et al. 2013] são definidos três tipos de dispositivos IoT: sensor (por exemplo, um sensor de temperatura), atuador (por exemplo, um interruptor liga/desliga e etiquetas (por exemplo, etiquetas RFID). Os *FoT-Devices* têm algumas funcionalidades além das propostas por [Bassi et al. 2013], pois, podem: realizar transformação de dados brutos em conteúdo estruturado seguindo as diretrizes do *Linked Data*; agrupar/agregar dados antes de serem enviados ao gateway; formatar a mensagem a ser enviada ao gateway; dentre outras coisas, a depender das suas capacidades de processamento e memória. Além disso, os dispositivos FoT não requerem conectividade com a Internet porque eles realizam comunicação diretamente com os *gateways* que os gerenciam. Desse modo, os dispositivos podem continuar monitorando (sensores) e/ou atuando (atuadores) no ambiente, mesmo sem uma conexão com a Internet ativa, e todos os dados são enviados para serem processados nos *gateways*;

- *FoT-Gateway* - é um nó da rede FoT que tem como objetivo principal mapear os protocolos da camada de comunicação (Ethernet, Wi-Fi, ZigBee, Bluetooth e outros) para HTTP (camada de aplicativo da Web). Sendo assim, um *FoT-Gateway* oferece serviços Web RESTful para acesso as funcionalidades dos *FoT-Devices* (*Thing as a Service paradigm* – TaaS) e a outros serviços da IoT. Assim, as aplicações podem abstrair o protocolo de comunicação com os dispositivos e acessá-los através de uma API REST padronizada para cada tipo de dispositivo, como é o funcionamento atual da maioria dos aplicativos da Web;

- *FoT-Server* - servidores mais robustos que o *FoT-Gateway* e que podem ser de dois tipos: um tipo especial de *gateway* aprimorado com recursos de gerenciamento; um tipo especial de recurso fornecendo informações específicas que não são suportadas por dispositivos e *gateways*, como, por exemplo, dados históricos de longo prazo de dispositivos (servidor de dados) ou credenciais de identificação e autorização (servidor provedor de segurança);

- *Application* - são aplicações que utilizam a API REST baseada em HTTP (RESTful Web Services) para acessar os serviços IoT fornecidos pela FoT, via os *FoT-Gateways*, e que oferece a interação com serviços para os usuários. Os aplicativos podem ser Web, dispositivos móveis (Android, iPhone ou Windows Phone) ou mesmo aplicativos *desktop* tradicionais;

- *FoT-Profile* - define com quais tipos de serviços que os nós da FoT (FoT-Gateway e FoT-Servers) podem ser configurados. Como exemplo, alguns dos perfis que os nós da FoT podem assumir são: composição; descoberta; segurança; armazenamento; dentre outros.

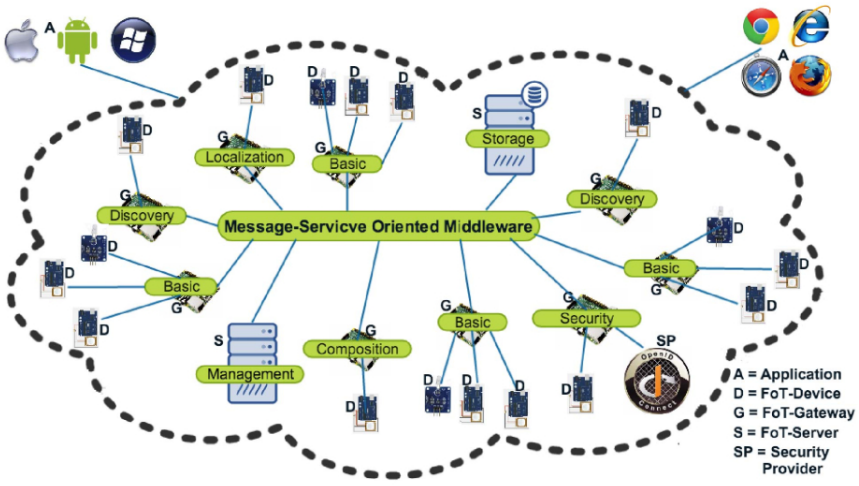


Figure 6.1. Visão Geral SOFT-IoT. Obtida de [Prazeres and Serrano 2016].

Como forma de validar a proposta do paradigna FoT, [Prazeres and Serrano 2016] apresentam a plataforma SOFT-IoT (*Self-Organizing Fog of Things for The Internet of Things*), que é uma implementação concreta do paradigma *Fog of Things*. Os *FoT-Devices* na plataforma SOFT-IoT são nós de sensores ou atuadores embutidos com um

driver (TATUDevice) que é implementado a partir de um protocolo leve denominado TATU (*The Accessible Thing Universe Protocol*). O protocolo TATU estende o protocolo MQTT através da padronização das mensagens para comunicação entre *FoT-Devices* e *FoT-Gateways*. Além disso, *TATUDevice* também implementa o protocolo *Zeroconf*[1] e uma descrição semântica do dispositivo baseada em uma taxonomia (ontologia) para dispositivos IoT.

O *gateway* da SOFT-IoT implementa um *middleware* composto por duas camadas, uma camada orientada a serviços e outra orientada a mensagens [Prazeres et al. 2017]. A camada do *middleware* orientada a serviços fornece comunicação entre os aplicativos e os serviços implantados nos *gateways*, utilizando a tecnologia ESB (Enterprise Service Bus) com base na especificação OSGi[2]. Por outro lado, a camada orientada a mensagens fornece comunicação entre os *FoT-Gateways* e os *FoT-Devices* da SOFT-IoT. Portanto, um *FoT-Gateway* na plataforma SOFT-IoT é, em geral, um dispositivo de baixo custo com processamento e recursos de memória limitados. Além disso, o *FoT-Gateway* utiliza versões reduzidas do sistema operacional Linux, uma máquina virtual Java, uma implementação da especificação OSGi (parte orientada para o serviço) e um servidor MQTT (parte orientada para mensagens). As tecnologias utilizadas no *FoT-Gateway* são mostradas na Figura 6.2. Na arquitetura mostrada na Figura 6.2, a camada do *middleware* orientada a serviços fornece interfaces para aplicativos acessarem os dispositivos via Web Services RESTful (TaaS), utilizando a tecnologia Apache CXF. Por sua vez, o *broker* MQTT serve para fornecer comunicação através de mensagens entre o *gateway* e os dispositivos e também entre os diversos gateways. Por fim, existe o componente *Mapping Devices* que tem como objetivo intermediar a comunicação entre os dispositivos e os *gateways*, sendo implementado seguindo a especificação OSGI e implantado no servidor Apache Karaf.

A plataforma SOFT-IoT propõe três tipos de servidores FoT, são eles: servidor de gerenciamento; servidor de armazenamento; e um servidor provedor de segurança. O primeiro é um servidor FoT que funciona como um tipo especial de *gateway* e implementa todos os serviços relacionados com a auto-organização dinâmica da plataforma SOFT-IoT. Os dois últimos são *FoT-Servers* que funcionam como um tipo especial de recurso e implementa serviços relacionados aos aspectos de armazenamento e segurança. A plataforma SOFT-IoT, por se basear em serviços, é uma plataforma suficientemente flexível para implantação dinâmica de novos tipos de servidores FoT, que, portanto, podem oferecer seus serviços para fins específicos. Por exemplo, os serviços IoT para análise de grandes volumes de dados (*Big Data Analytics*) podem ser desenvolvidos e implementados em *FoT-Servers* para análise de dados localmente (na borda da rede).

A plataforma SOFT-IoT também propõe recursos de auto-organização que devem ser implantados nos servidores da FoT, por exemplo, os servidores de gerenciamento devem implementar todos os serviços relacionados à auto-organização da plataforma SOFT-IoT. Os principais serviços que podem ser implementados nos servidores de gerenciamento são [Sousa and Prazeres 2018][Prazeres et al. 2017]: serviço de monitoramento auto-organizado; serviço de implantação de *gateways*; serviço de recuperação de falhas; serviço de gerenciamento e balanceamento dos perfis.

[1]http://www.zeroconf.org/

[2]https://www.osgi.org/developer/architecture/

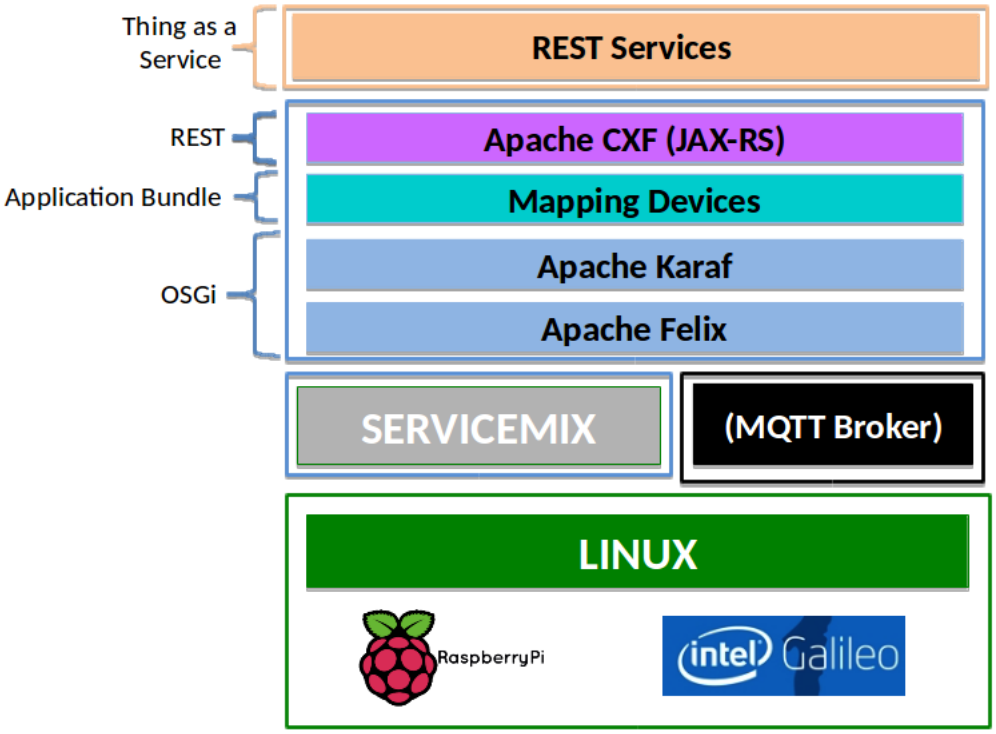


Figure 6.2. Gateway da Fog of Things para plataforma SOFT-IoT, adaptado de [Prazeres and Serrano 2016].

O paradigma *Fog of Things* e sua implementação concreta na plataforma SOFT-IoT suporta a implantação em diversos domínios da IoT, como, por exemplo, rede de sensores básicos, casas inteligentes, veículos autônomos e assistência médica. Além disso, permite a utilização de diferentes configurações de arquiteturas na Computação em Névoa e/ou na Computação em Nuvem [Andrade et al. 2018], como apresentado na Figura 6.3. Com a SOFT-IoT, podem ser criadas as seguintes arquiteturas computacionais: somente redes de sensores de área pessoal; rede de sensores de área pessoal combinada com gerenciamento de FoT-Gateways, criando assim uma rede local; rede de sensores de área pessoal com FoT-Gateways e FoT-Servers, criando assim uma infraestrutura mais robusta de rede local; somente Cloud Servers gerenciando os dispositivos locais; implantação de todo infraestrutura desde a rede global até a rede de área pessoal. Ou seja, a arquitetura é flexível e configurável de forma a atender requisitos de infraestrutura e de aplicações específicas ou genéricas.

6.3. Arquitetura IoT e Plataforma SOFT-IoT

Dado a diversidade de plataformas, infraestruturas, arquiteturas e aplicações propostas em IoT, tanto pela academia como pela indústria, alguns pesquisadores e mesmo a indústria têm trabalhado no sentido de propor padronizações e especificações compartilhadas. Khan et al. [Khan et al. 2012], Bassi et al. [Bassi et al. 2013] e Al-Fuqaha et al. [Al-Fuqaha et al. 2015] são exemplos dessas propostas. Tal como apresentado na Seção 6.2, a plataforma SOFT-IoT reusa os conceitos definidos no modelo conceitual de IoT proposto por Bassi et al. [Bassi et al. 2013]. Em relação à arquitetura, a plataforma SOFT-IoT utiliza conceitos definidos por Khan et al. [Khan et al. 2012] e Al-Fuqaha et al. [Al-Fuqaha

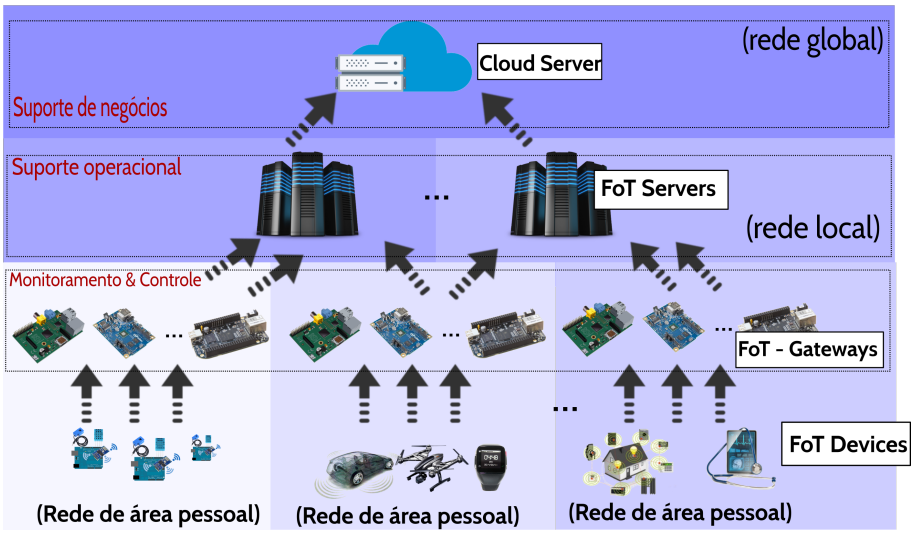


Figure 6.3. Paradigma Fog of Things, adaptado de [Andrade et al. 2018]

et al. 2015], que propuseram a divisão da estrutura da IoT em cinco camadas:

- A **Camada de Percepção** também é denominada camada de dispositivo. Essa camada é composta por objetos físicos e dispositivos como os sensores, atuadores e etiquetas, que trata basicamente da identificação e coleta de informações através de sensores. Dependendo do tipo de sensor, as informações podem ser sobre localização, temperatura, orientação, movimento, vibração, aceleração, umidade, mudanças químicas no ar, dentre outras. As informações coletadas são então passadas para a Camada de Rede para sua transmissão segura ao nó de rede responsável pelo processamento das informações.

- A **Camada de Rede** transfere com segurança as informações dos sensores para o sistema que processará as informações. O meio de transmissão pode ser com fio ou sem fio e a tecnologia pode ser 3G, 4G, UMTS, Wifi, Bluetooth, infravermelho, ZigBee, dentre outros, a depender da tecnologia utilizada no sensor. Assim, a Camada de Rede transfere as informações da camada de percepção para a camada de middleware.

- Na **Camada de Middleware** os dispositivos IoT implementam diferentes tipos de serviços. Cada dispositivo se conecta e se comunica apenas com outros dispositivos que possuam o mesmo tipo de serviço. A principal responsabilidade dessa camada é o gerenciamento de serviços e e o oferecimento de conexão com alguma base de dados. As informações obtidas da Camada de Rede são, dessa forma, armazenadas na base de dados. Além disso, essa camada realiza o processamento de informações e toma decisões automáticas com base nos resultados desse processamento.

- A **Camada de Aplicação** fornece serviços solicitados pelos usuários. Por exemplo, a Camada de Aplicação pode fornecer medições de temperatura e umidade do ar ao usuário que solicitar essas informações. A importância dessa camada para a IoT é que ela possui a capacidade de fornecer serviços inteligentes, com qualidade, visando atender as necessidades dos usuários. A camada de aplicação cobre

vários mercados verticais, como casas inteligentes, prédios inteligentes, transporte, automação industrial e saúde inteligente, dentre outros.

- A **Camada de Negócio** gerencia as atividades e serviços gerais de um sistema IoT. As responsabilidades dessa camada são construir um modelo de negócios, gráficos, fluxogramas, etc., com base nos dados recebidos da Camada de Aplicação. Assim, a Camada de Negócios possibilita o suporte a processos de tomada de decisão com base na análise de *Big Data*. Além disso, o monitoramento e o gerenciamento das quatro camadas subjacentes são obtidos nessa camada. Por fim, a Camada de Negócio também é responsável por comparar a saída de cada camada, com a saída esperada, com a finalidade de melhorar os serviços e manter a privacidade dos usuários.

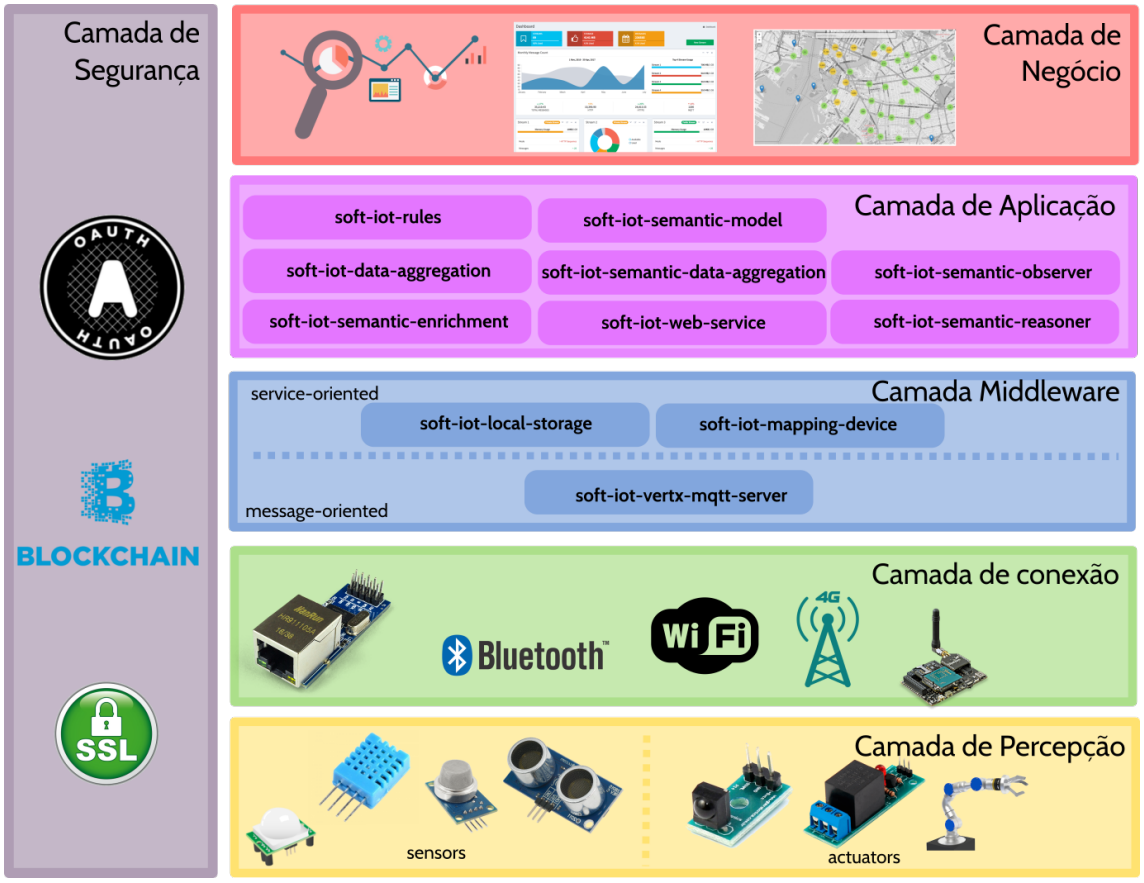


Figure 6.4. Arquitetura IoT com a plataforma SOFT-IoT

A Figura 6.4 mostra uma visão da plataforma SOFT-IoT implementada na arquitetura IoT descrita anteriormente. No restante desta seção, é explicado o desenvolvemos de aplicações IoT a partir da adoção da arquitetura IoT apresentada na Figura 6.4: são descritos os aspectos e os mecanismos de segurança em desenvolvimento na plataforma SOFT-IoT. Ainda no restante desta seção, são descritas as características e os detalhes de configuração e instalação da plataforma SOFT-IoT dentro de cada camada da Figura 6.4.

Para ter uma visão centralizada e resumida da instalação e configuração da plataforma acesse o link abaixo:

`https://github.com/WiserUFBA/soft-iot-platform`

6.3.1. Camada de Percepção

A camada de percepção contém os *FoT-Devices* que são responsáveis por captar informações do ambiente, por meio de sensores, ou atuar no ambiente, por meios de atuadores. Existem vários dispositivos que podem ser utilizados para esta finalidade. Na Figura 6.5 observa-se o nó de sensores Sonoff SC da ITEAD[3]. Esse nó de sensores é composto por cinco sensores: som, temperatura, umidade, poeira e luminosidade, que, na Figura 6.5, estão destacados em amarelo.

Além disso, o nó conta com o microcontrolador ATmega328, em destaque na cor vermelha na Figura 6.5. Esse microcontrolador, que foi desenvolvido pela Atmel Corporation (adquirida pela Microchip Technology), possui baixo custo, porém, também tem um baixo poder de processamento, por ter um microprocessador de 8 bits. No Sonoff SC ilustrado na Figura 6.5, o ATmega328 é responsável pelo interfaceamento entre os sensores com o ESP8266 (em destaque na cor rosa na Figura 6.5).

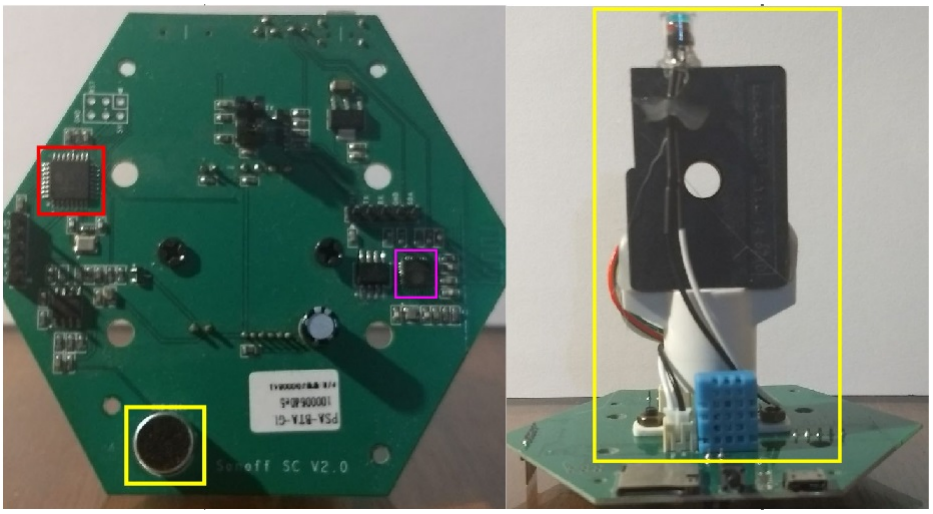


Figure 6.5. Sonoff SC da ITEAD.

O ESP8266 foi desenvolvido pela Espressif e possui poder de processamento maior que o ATmega328, por ter um microprocessador de 32 bits, além de possuir mais memória disponível. Por essas características e pelo seu baixo custo, o ESP8266 representa uma boa escolha como adaptador Wi-Fi e intermediador entre *FoT-Devices* e *FoT-Gateways*.

6.3.1.1. ESP8266

O primeiro passo para possibilitar o uso do ESP8266, presente no Sonoff SC, na plataforma SOFT-IoT, é trocar o *firmware*, que vem de fábrica, antes de sua primeira utilização. Essa

[3]https://www.itead.cc/

ação permite reprogramar o dispositivo utilizando um Ambiente de Desenvolvimento Integrado (Integrated Development Environment - IDE, em inglês), como por exemplo a Arduino IDE[4].

Para esse procedimento, é necessário um conversor serial/USB, como o apresentado na Figura 6.6, onde, em verde, estão destacados a chave seletora de tensão da comunicação serial e os pinos de alimentação (VCC), tanto 3.3V quanto 5V. A posição da chave seletora e qual pino utilizar para alimentar dependem da tensão de operação do dispositivo a ser programado.

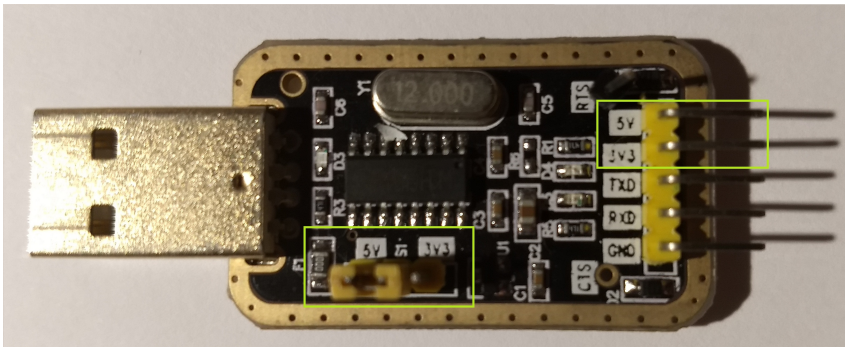


Figure 6.6. Conversor USB/Serial 3.3V/5V CH340G.

O importante, ao decidir qual conversor escolher, é observar o componente a ser programado. Por exemplo, o microcontrolador ATmega328 opera em uma tensão de 5V e necessita de 5 pinos para ser programado: 2 pinos para alimentação (VCC-5V e terra), 2 pinos para comunicação serial (RX, TX) e um pino para reiniciar(RESET). No caso do ESP8266, que opera em uma tensão de 3.3V e seria danificado caso alimentado com tensões maiores, são usados 4 pinos: 2 pinos para alimentação (VCC-3.3V e terra) e 2 pinos para comunicação serial (RX, TX). Portanto, o ideal é escolher um conversor que possa operar nas duas tensões e que possua o pino extra necessário para programar o ATmega328.

Na Figura 6.7, que apresenta a parte traseira do nó de sensores Sonoff SC, estão destacados os pinos de programação do ESP8266 (rosa), do ATmega328 (amarelo) e os dois jumpers (branco), que interligam e permitem a comunicação serial entre esses dois componentes. Esses jumpers são peças pequenas de plástico e metal que devem ser removidos no momento da programação.

Para a troca do *firmware* do ESP8266 é necessário baixar um programa chamado ESP8266 Flash Downloader. O ESP8266 Flash Downloader é um programa para o sistema operacional Windows. Porém, outros programas podem ser utilizados para outros sistemas operacionais, tal como o ESPtool para Linux. Além disso, é necessário um arquivo binário contendo o novo *firmware* (o arquivo e o programa estão disponíveis na página do Github[5]) e 5 fios jumpers Fêmea X Fêmea para interligar o conversor USB/Serial com os pinos de programação de cada componente, devido a necessidade de 5 pinos do microcontrolador ATmega328.

[4]https://www.arduino.cc/en/Main/Software

[5]https://github.com/WiserUFBA/soft-iot-devices/tree/master/Programa_Flash

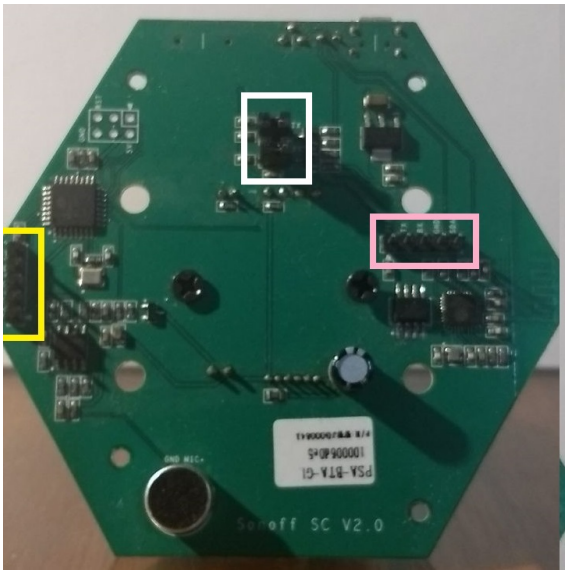


Figure 6.7. Pinos de programaçao do ESP8266 e do ATmega328, respectivamente e localização dos jumpers a serem removidos durante programação.

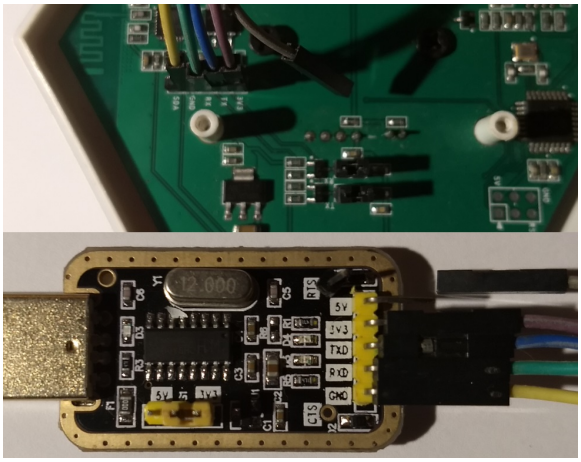


Figure 6.8. Interligação de fios do ESP8266 com conversor USB/Serial.

A Figura 6.8 ilustra a conexão entre o conversor USB/Serial e os pinos de programação do ESP8266 do Sonoff SC. O mais importante é se atentar aos pinos de alimentação e na mudança da chave seletora de tensão da comunicação serial para 3.3V. Os pinos de comunicação podem variar sua conexão a depender do conversor utilizado. Alguns seguem o padrão RX-TX e TX-RX, mas há casos de conversores que requerem TX-TX e RX-RX. Nesse caso, é importante observar a correta posição da chave seletora de tensão para o componente a ser programado e inverter os fios RX e TX em casos de falha ao programar ou trocar de *firmware*.

Outra questão importante sobre a programação dos componentes é que alguns necessitam de procedimentos extras para entrarem no modo de programação. O ESP8266, por exemplo, necessita que o pino GPIO0 seja conectado ao pino RESET. No nó de sensores Sonoff SC essa ação é realizada pressionando e mantendo pressionado o botão disponível na placa no momento de conexão do conversor USB/Serial com o computador.

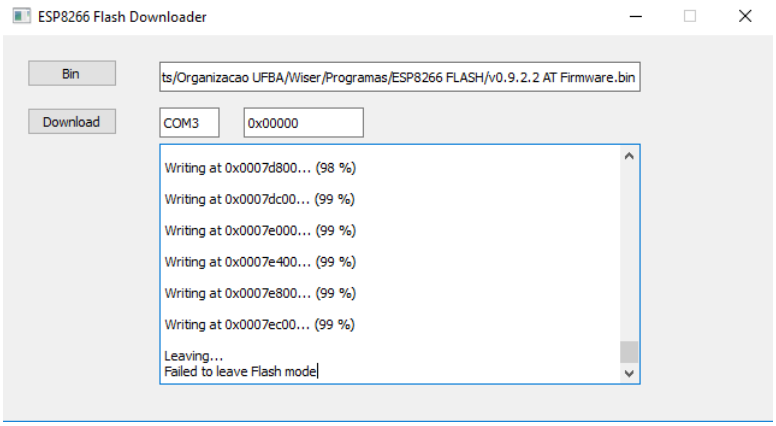


Figure 6.9. Programa de trocar de firmware ao finalizar ação.

Deve-se observar também a porta que o conversor está inserido e modificar para a porta correta no programa, por padrão o programa inicia com a porta "COM0". Na Figura 6.9, observa-se a janela do programa ao finalizar o processo de troca de firmware.

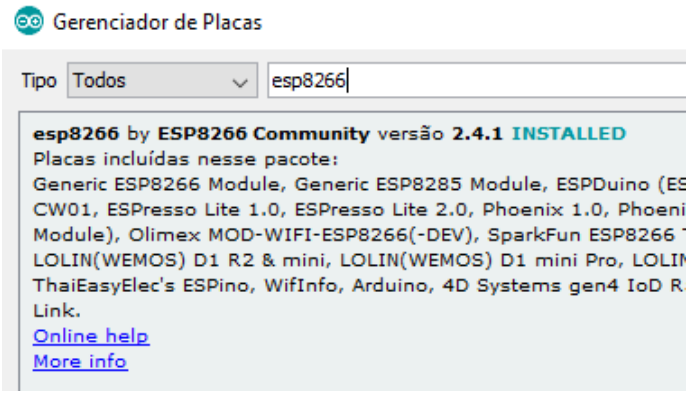


Figure 6.10. Gerenciador de placas da Arduino IDE.

Antes de programar os componentes é necessário configurar a Arduino IDE. O primeiro passo é instalar, por meio do gerenciador de placas da IDE, a placa ESP8266. Para tal, é preciso seguir as abas *Arquivo* > *Preferências*, e no campo "URLs Adicionais para Gerenciadores de Placas" adicionar o seguinte endereço: *http://arduino.esp8266.com/stable/package_esp8266com_index.json*. Dessa forma, a placa aparecerá na busca realizada em *Ferramentas* > *Placa:* > *Gerenciador de Placas...*, como visto na Figura 6.10.

O segundo passo é instalar as bibliotecas utilizadas pelos códigos dos dois componentes (ATmega328 e ESP8266), a Arduino IDE possui um gerenciador próprio para isso que pode ser visualizado na Figura 6.11. As bibliotecas do TATU precisam ser baixadas pela página do grupo de pesquisa WISER(Web, Internet and Intelligent Systems Research) disponível na página do Github[6]. As demais bibliotecas podem ser instaladas pelo gerenciador seguindo as seguintes abas: *Sketch* > *Incluir Biblioteca* > *Gerenciador Bibliotecas...*

[6]https://github.com/WiserUFBA/soft-iot-devices/tree/master/Sonoff_SC

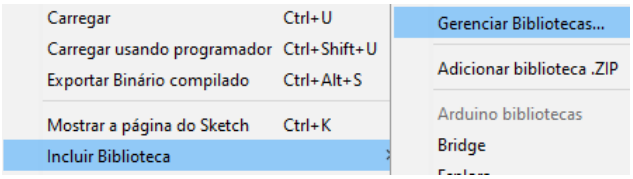


Figure 6.11. Gerenciador de bibliotecas da Arduino IDE.

Elas são utilizadas nas seguintes versões:

1. DHT sensor library da Adafruit - versão 1.0.0
2. PubSubCliente do Nick O'Leary - versão 2.6.0
3. Bounce2 do Thomas O Fredericks - versão 2.52.0
4. ArduinoJson do Benoit Blanchon - versão 5.13.2
5. WiFiManager de tzapu - versão 0.12.0

Com a Arduino IDE configurada, o passo seguinte é programar o ESP8266 com o código chamado "SonoffSC_ESP 8266"[7]. Mesma conexão entre pinos de programação do ESP8266 e conversor USB/Serial utilizada para trocar o firmware da placa, além de novamente usar o procedimento com o botão e a remoção dos jumpers que interligam ESP8266 e ATmega328. Na Arduino IDE, selecione a placa "Generic ESP8266" e defina a porta para a que está sendo utilizada pelo conversor no computador, conforme mostrado na Figura 6.12.

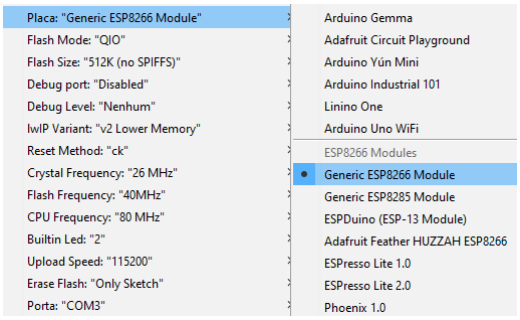


Figure 6.12. Definição da placa a ser utilizada.

Existem 2 pontos a serem modificados antes de carregar o código no ESP8266 do Sonoff SC que podem ser vistos na Figura 6.13. Primeiramente, é necessário definir: nome do dispositivo, usuário do MQTT, senha do MQTT e porta. E depois definir qual será o nome e senha da rede Wi-Fi a ser acessada (não deve ser uma rede existente). Essa rede servirá para configurações dinâmicas do dispositivo que são possíveis devido a biblioteca mencionada anteriormente, WiFiManager. Essas configurações permitem alterar dados, como rede de internet a se conectar, sem a necessidade de reprogramar o dispositivo. Com esses dois pontos definidos, pode-se enviar o código para a placa com o ícone "Carregar" no canto superior esquerdo ou pela combinação de teclas "Crtl + U". Assim que o código estiver carregado, a rede criada anteriormente irá aparecer como disponível.

[7]Disponível em: https://github.com/WiserUFBA/soft-iot-devices/tree/master/Sonoff_SC

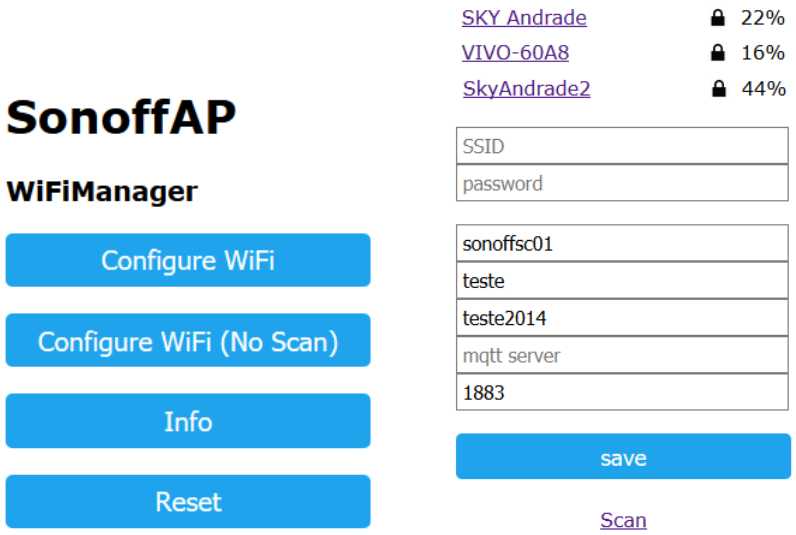
```
SonoffSC_ESP8266
//
char vector_response[1024];

#define stringSize 20
char device_name[stringSize]    = "sonoffsc01";
char mqtt_user[stringSize]      = "teste";
char mqtt_pass[stringSize]      = "teste2014";
char port[stringSize]      = "1883";
char mqtt_server[40];
(...)
wifiManager.startConfigPortal("SonoffAP", "teste2014");
```

Figure 6.13. Parte do código do Sonoff SC para o ESP8266.

Ao se conectar a esta rede de acesso, é preciso entrar em um navegador WEB e acessar o seguinte endereço: *http://192.168.4.1*. Dessa forma, aparecerá uma página como a Figura 6.15(a). As opções são: Configurar WiFi (nesta opção são listadas redes disponíveis), Configurar WiFi sem escaneamento (não são listadas as redes disponíveis), Informações e Reiniciar.

Figure 6.14. Configurando dinamicamente o dispositivo.



(a) Página inicial.

(b) Página exibida ao clicar em "Configure WiFi".

Após selecionar a opção "Configurar WiFi", é apresentada uma página como a Figura 6.15(b) onde as redes disponíveis e qualidade de sinal serão listadas. Assim, é só clicar na rede desejada e digitar a senha. É necessário também informar o endereço do FoT-Gateway com o qual o ESP8266 irá se comunicar. Outras opções de configuração possíveis envolvem alterar: nome do dispositivo, usuário MQTT, senha MQTT e porta. Finalizado todas as configurações a serem realizadas, basta salvar. Por fim, aparecerá uma página informando que os dados foram salvos.

Outro fato interessante da configuração dinâmica, é a possibilidade de não perder esses dados em casos de reinício do nó de sensores. Quando esse evento ocorre, o nó

se conecta automaticamente na última rede configurada. Esta rede pode ser alterada ao apertar o botão presente no Sonoff SC e entrando novamente na rede de acesso das configurações dinâmicas.

6.3.1.2. ATmega328

O microcontrolador ATmega328 também precisa ser reprogramado para que o Sonoff SC possa ser utilizado como um FoT-Device na plataforma SOFT-IoT. Esse processo é mais simples, para isso é necessário carregar código chamado "SonoffSC_Arduino"[8] nele. As interligações entre os pinos de programação do ATmega328 e do conversor USB/Serial estão mostradas na Figura 6.15.

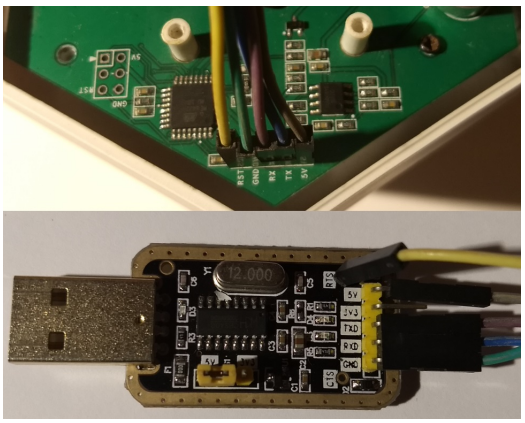


Figure 6.15. Interligação de fios do ATmega328 com conversor USB/Serial.

Com as interligações feitas, é preciso configurar a Arduino IDE para programar o ATmega328. É preciso ir em *Ferramentas* > *Placa:* e selecionar a opção "Arduino Mini". Lembrando-se sempre de confirmar se a porta está configurada corretamente para a porta que o conversor está utilizando no computador. Antes de enviar o código, é válido analisar parte do código do ATmega328, mostrado na Figura 6.16. O nome do dispositivo deve ser o mesmo configurado no ESP8266, assim como a porta MQTT. Observado essas informações, pode-se enviar o código. Finalizado esta tarefa, o Sonoff SC estará definitivamente pronto para exercer o seu papel de FoT-Device, bastando recolocar os jumpers removidos para programação e ligá-lo a uma fonte de alimentação.

SonoffSC_Arduino

```
#include <stdint.h>
#include <string.h>
#include <DHT.h>

// Constants to connection with the broker
#define DEVICE_NAME   "sonoffsc01"
#define MQTT_PORT     1883
```

Figure 6.16. Parte do código do Sonoff SC para o ATmega328.

[8]Disponível em: https://github.com/WiserUFBA/soft-iot-devices/tree/master/Sonoff_SC

6.3.1.3. Protocolo TATU

Como protocolo de comunicação entre FoT-Devices e FoT-Gateway é utilizado o protocolo TATU (The Acessible Thing Universe). Este protocolo é uma extensão do protocolo MQTT (Message Queuing Telemetry Transport) desenvolvida para atender algumas das necessidades de um sistema IoT, como por exemplo, métodos que facilitem a edição das variáveis e dos pinos, além de informar os valores destes pinos, métodos que retornem informações do dispositivo (nome, id, localização) e a possibilidade de atribuir nomes a variáveis.

Este protocolo contém 3 métodos:

(i) GET: retorna o valor de uma variável, pino ou propriedade do sistema.

(ii) SET: edita o valor de uma variável ou pino.

(iii) POST: é o retorno de qualquer um dos métodos descritos anteriormente, representado por uma mensagem no formato JSON.

O GET é utilizado em FoT-Devices que possuem sensores, de modo que os dados desses sensores possam ser capturados e processados por outros componentes da FoT. O SET é utilizado em FoT-Devices que possuem atuadores para editar pinos específicos desses atuadores com a finalidade de realizar a mudança definida por outros componentes da FoT.

Além disso, o protocolo TATU disponibiliza ao FoT-Gateway e FoT-Device dois tipos de fluxos de dados [Batista et al. 2018a][Batista et al. 2018b]:

(i) GET: requisita um dado de algum sensor, sem periodicidade definida. Ilustrado na Figura 6.18(a).

(ii) FLOW: requisita uma amostra de dados de algum sensor que deve ser enviada a cada Y unidades de tempo. Nesse fluxo, o FoT-Device irá coletar dados para formar uma amostra a cada X unidades de tempo e quando atingir Y unidades de tempo irá enviar esses dados em conjunto para o FoT-Gateway. Ilustrado na Figura 6.18(b).

As requisições de dados e as respostas obtidas seguem os padrões descritos na Figura 6.19(a) e 6.19(b), respectivamente. Além disso, exemplos de cada uma das formas de requisições no protocolo TATU são apresentadas no código 6.1. O tópico das requisições é o *"\dev"* e o dispositivo retornará respostas no tópico *"\dev\nome_dis e positivo\RES"*.

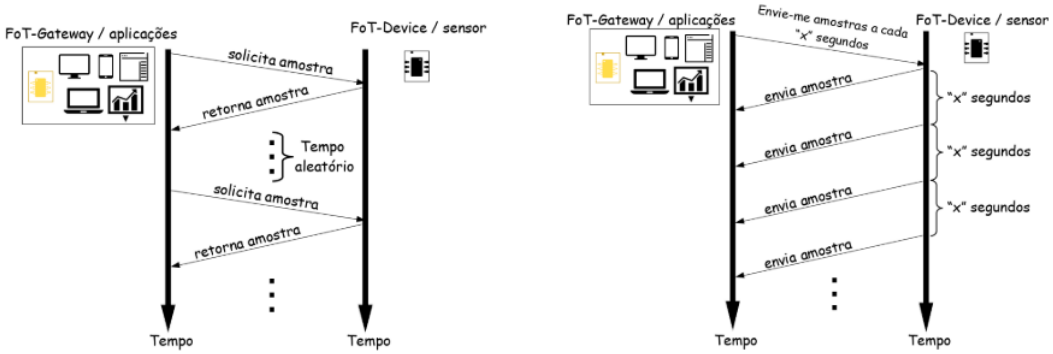
6.3.2. Camada Middleware

A camada middleware na SOFT-IoT é responsável por intermediar a comunicação com os dispositivos IoT (FoT-Device) com os demais serviços da plataforma. O middleware pode ser implantado em dispositivos de capacidade limitada (como Raspberry Pi[9] e BeagleBone[10]), e também servidores em rede local ou remota (Cloud). A SOFT-IoT define

[9]Raspberry Pi - https://www.raspberrypi.org/

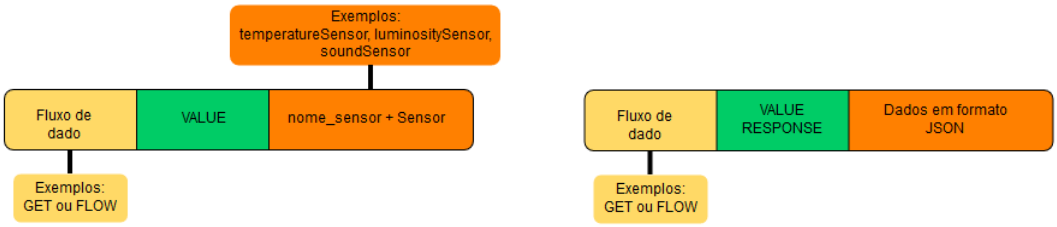
[10]BeagleBone - https://beagleboard.org/bone

Figure 6.17. Ilustrações dos fluxos de dados.



(a) Fluxos baseados em Requisições GET (b) Fluxos baseados em Requisições FLOW

Figure 6.18. Padrão de formato para requisições e respostas.



(a) Formato das requisições. (b) Formato das respostas.

```
//Exemplo de requisicao baseada em GET com sensor de temperatura
GET VALUE temperatureSensor
//Resposta para requisicoes GET
GET VALUE RESPONSE {"CODE":"POST","HEADER":{"NAME":"device_name"
    },"METHOD":"GET","BODY":{"temperaturaSensor":25}}

//Exemplo de requisicao baseada em FLOW, coletas a cada 200 ms e
     publicacoes a cada 1000 ms
FLOW VALUE temperatureSensor {"collect":200, "publish":1000}
//Resposta para requisicoes FLOW, uma aresta com dados da
    amostra obtida.
FLOW VALUE RESPONSE {"CODE":"POST","HEADER":{"NAME":"device_name
    "},"METHOD":"GET","BODY":{"temperaturaSensor"
    :[25,25,27,25,26],"FLOW":{"collect":200,"publish":1000}}}
```

Código 6.1. Exemplos de requisicoes e respostas no protocolo TATU

que o middleware é implementado no FoT-Gateway, e este quando alocado em dispositivos/servidores locais, não dependente do acesso à internet (somente comunicação de rede local) para coleta, armazenamento e acesso aos dados produzidos pelos sensores. Quando implantados em servidores remotos precisam de conectividade estabelecida para captura dos dados dos sensores.

A implementação do middleware na plataforma SOFT-IoT é realizada utilizando o Apache ServiceMix, que funciona como um Enterprise Service Bus (ESB – barramento de serviços), baseado na especificação OSGi. Além disso, o middleware é dividido em duas partes, as quais são [Prazeres et al. 2017]:

- **Orientada a mensagem**: provê o ponto de comunicação com os sensores e atuadores (FoT-Device) através de mensagens MQTT com protocolo de comunicação TATU;
- **Orientada a serviço**: tem a função de viabilizar e oferecer interfaces de acesso aos dados dispositivos pela camada de aplicação.

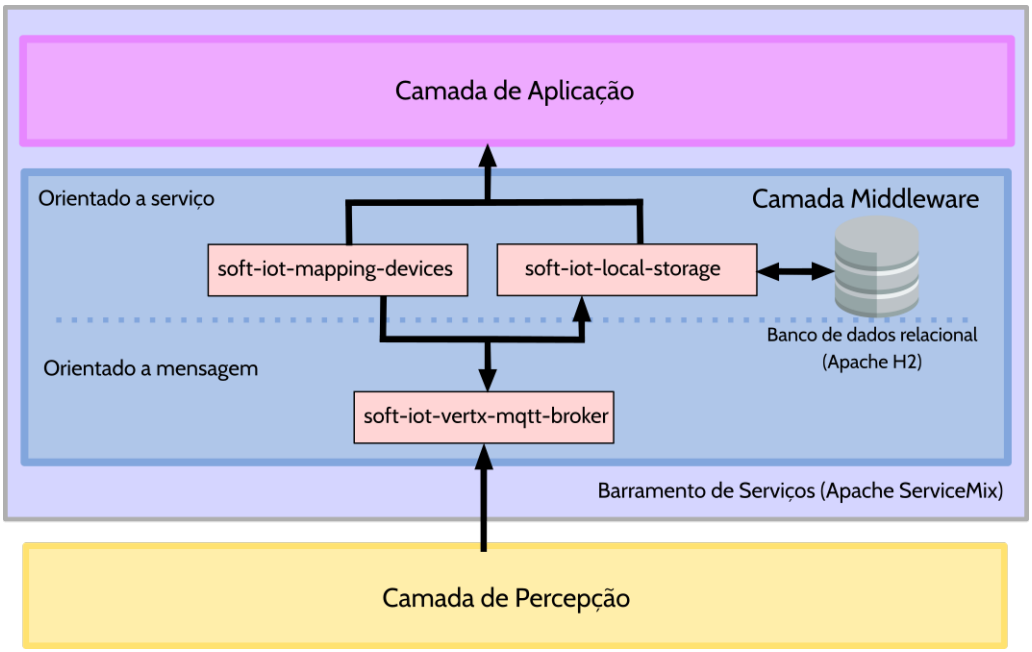


Figure 6.19. Componentes e interações da camada Middleware da SOFT-IoT

A Figura 6.19 mostra a camada middleware, seus módulos e interações. Nota-se que a camada middleware é implementada sobe o ESB com o Apache ServiceMix e possui módulos para prover a comunicação com os dispositivos da camada de Percepção (*soft-iot-vertx-mqtt-broker*), para mapeamento dos sensores (*soft-iot-mapping-devices*) e para coleta e armazenamento dos dados (*soft-iot-local-storage*). Além disso, o módulo *soft-iot-local-storage*, junto com o *soft-iot-mapping-devices*, fornece uma interface de acesso aos dados dos sensores para outros módulos implementados na camada de aplicação.

As subseções a seguir detalham o funcionando da infraestrutura e dos módulos que compõem a camada middleware, tal como um guia para sua instalação e configuração.

6.3.2.1. Barramento de Serviços - ESB

A utilização de um barramento de serviços permite a implantação dinâmica de softwares através de uma infraestrutura base, a qual fornece suporte a comunicação entre aplicações. O ServiceMix, baseado na especificação OSGi, é utilizado como barramento de serviço na plataforma SOFT-IoT.

O Apache ServiceMix[11] é um software código aberto, implementado em Java, no qual serviços nativos da arquitetura ESB/OSGi oferecem toda a infraestrutura necessária para o suporte dos outros serviços da plataforma SOFT-IoT. Com o ServiceMix é possível implantar serviços (também chamados de bundles) em tempo de execução e permite que estes partilhem dados e objetos através de relacionamentos e dependências.

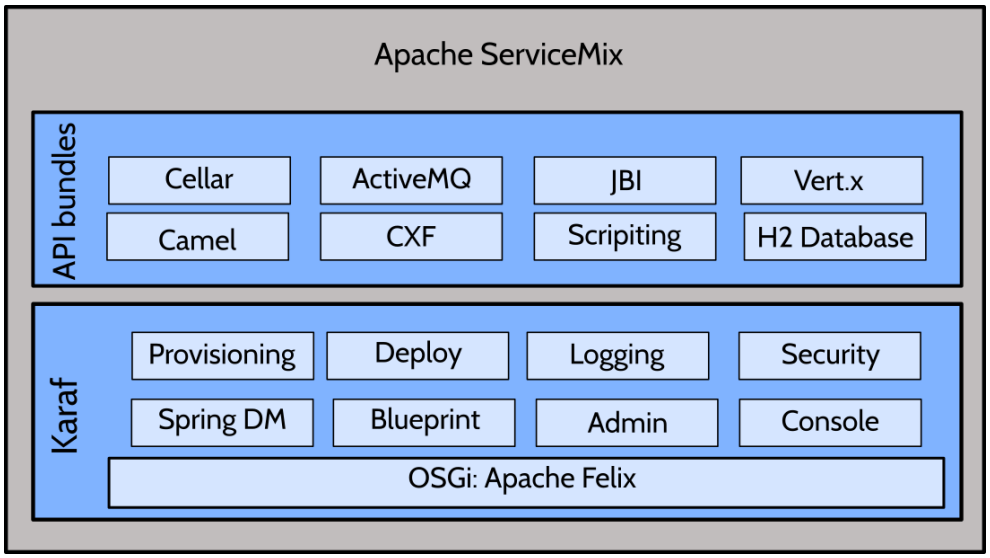


Figure 6.20. Componentes do Apache ServiceMix

A Figura 6.20 representa o ServiceMix e os seus principais módulos. O ServiceMix é leve e portável, tendo como requisito básico o suporte ao Java 1.7. Além disso, possui suporte ao Spring Framework e Blueprint e é compatível com o Java SE ou com um servidor de aplicativos Java EE. O Karaf é o núcleo principal do ServiceMix e oferece o conceito de recursos, onde coleções de pacotes podem ser instalados como um grupo em um ambiente OSGi em execução. Sobe o Karaf, o ServiceMix usa o ActiveMQ para fornecer serviço de mensagens, CXF para suporte serviços RESTful, Cellar para comunicação e interações entre diferentes instalações do ServiceMix e Camel para integração e troca de dados através de rotas. Por fim, o ServiceMix contém módulos adicionais como H2 Database, que gerencia o sistema de banco de dados relacional baseado em arquivos.

A plataforma SOFT-IoT foi baseada na versão 6.10 do ServiceMix. Para seu uso e instalação em um ambiente Linux é necessário somente baixar o pacote do programa, o qual pode ser encontrado em:

`https://servicemix.apache.org/downloads/servicemix-6.1.0.html`

[11] http://servicemix.apache.org/

Após a descompactação do pacote que contém o ServiceMix é possível observar uma conjunto de diretórios com funções específicas, dentre os quais podemos destacar: `etc/`: contém arquivos de configuração dos bundles (incluirá também os arquivos de configuração dos bundles da SOFT-IoT; `bin/`: contém os executáveis para inicializar o ServiceMix através do Karaf; `data/`: armazena dados produzidos pelos bundles, no caso do SOFT-IoT, nesse diretório que serão armazenados os dados obtidos dos sensores; `system/`: contém os arquivos básicos que suportam o ServiceMix.

Para inicializar o ServiceMix em sistemas Linux é preciso executar dentro do diretório do base o seguinte arquivo:

```
$ ./bin/servicemix
```

Para inicialização em sistemas operacionais Windows é preciso executar o arquivo *servicemix.bat*. A Figura 6.21 exibe o terminal do ServiceMix, no qual é possível executar comandos, como por exemplo para instalação e remoção de bundles, para listar dados, e também para verificar status das aplicações que estão operando. Ao completar a inicialização do ServiceMix o terminal é inicializado e disponibilizando para o usuário. Eventuais erros de execução nos bundles também são exibidos nesse terminal.

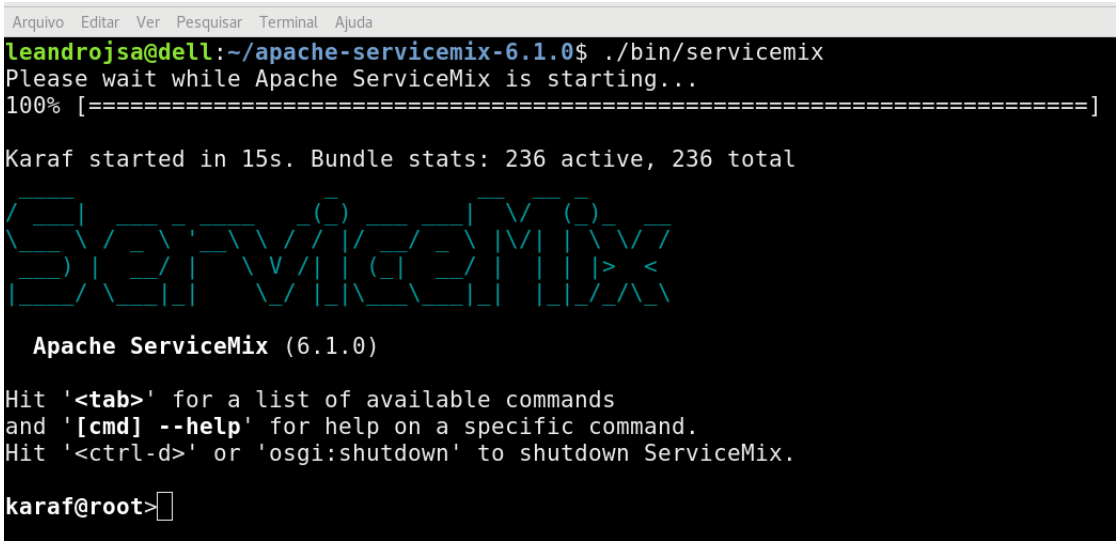


Figure 6.21. Terminal do Apache ServiceMix

O ServiceMix também pode ser gerenciado através de uma aplicação Web chamado webconsole. Nela é possível gerenciar por meio de uma interface gráfica, funções como instalação, atualização, remoção e informações de bundles e também funções de debug. Para instalar o webconsole no ServiceMix basta executar o seguinte comando no terminal:

```
karaf@root()> feature:install webconsole
```

Por padrão aplicação Web inicializada pelo webconsole é configurada na porta 8181 com login e senha `karaf` e pode ser acessada pelo endereço:

`http://localhost:8181/system/console`

A Figura 6.22 mostra um recorte do webconsole. Observe que é possível através dele ter um panorama geral dos bundles em execução e também alterar seus status. O

webconsole, além disso, permite instalar novos bundles e depurar o estado e eventuais erros das aplicações em execução.

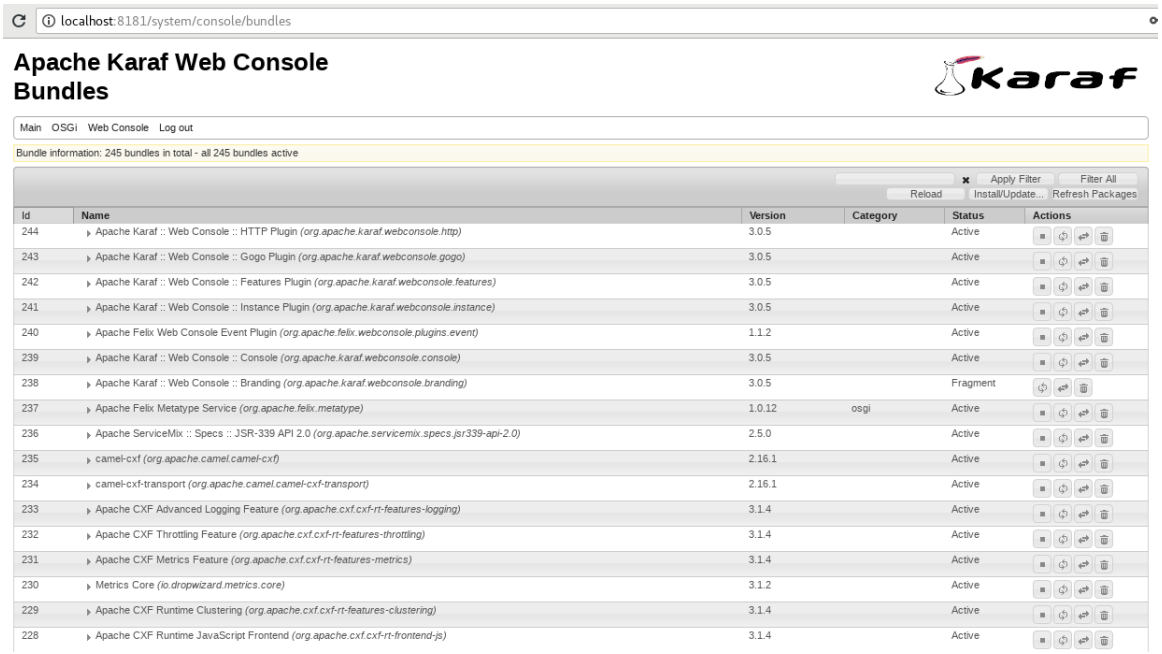


Figure 6.22. Webconsole do Apache ServiceMix (Karaf)

Por fim, é possível configurar o ServiceMix para permitir a instalação dos módulos da plataforma SOFT-IoT através do terminal. Isso é possibilitado pela inserção do repositório Maven do SOFT-IoT nas configurações do ServiceMix, habilitando a instalação dos bundles através de instruções de terminal. Para tal é preciso executar os seguintes comandos:

```
karaf@root()> config:edit org.ops4j.pax.url.mvn
karaf@root()> config:property-append org.ops4j.pax.url.mvn.repositories
",https://github.com/WiserUFBA/wiser-mvn-repo/raw/master/releases@
id=wiser"
karaf@root()> config:update
```

6.3.2.2. Módulo soft-iot-vertx-mqtt-broker

O componente MQTT Broker fornece um servidor que é capaz de lidar com conexões, comunicação e troca de mensagens com clientes MQTT remotos. Na plataforma SOFT-IoT o broker MQTT é implemetado no módulo **soft-iot-vertx-mqtt-broker**[12] uma implementação baseada em OSGI como um *bundle* do ServiceMix.

Este módulo é responsável por habilitar comunicações com clientes MQTT remotos, que no caso da plataforma SOFT-IoT são dispositivos conectados. Além das vantagens de usar uma arquitetura modular, o uso da API Vert.x MQTT Server torna possível escalar o agente MQTT reativo de acordo com, por exemplo, o número de núcleos do sistema e, assim, possibilitar a escalabilidade horizontal.

[12]https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker

Antes instalar o módulo soft-iot-vertx-mqtt-broker é necessário introduzir ao ServiceMix alguns módulos ao Vert.x. As dependências estão localizadas no endereço:

```
https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker/tree/master/
src/main/resources/dependencies
```

As dependências são bibliotecas .jar que devem ser incluídas no ServiceMix, podendo ser introduzidas através do webconsole, conforme pode ser visto na Figura 6.22 através do botão `Install/Update`. Além da instalação das dependências é necessário também incluir arquivos referente ao certificado de segurança do soft-iot-vertx-mqtt-broker. Tais arquivos estão disponíveis em:

```
https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker/tree/master/
src/main/resources/certificates
```

O certificado de segurança possui um arquivo de configuração disponível em:

```
https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker/blob/master/
src/main/resources/configuration/
br.ufba.dcc.wiser.soft_iot.gateway.brokers.cfg
```

Os arquivos do certificado de segurança, incluindo o arquivo de configuração devem ser armazenados em:

```
<diretorio_servicemix>/etc
```

Para instalação do soft-iot-vertx-mqtt-broker é necessário executar os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-vertx-mqtt-broker/1.0.0
```

6.3.2.3. Módulo soft-iot-mapping-devices

A plataforma SOFT-IoT tem um módulo específico para o mapeamento dos dispositivos e tradução das mensagens comunicação TATU entre o FoT-Device e FoT-Gateway. O **soft-iot-mapping-devices**[13] implementa as interfaces virtuais dos sensores e atuadores conectados na plataforma. Também funciona como tradutor das mensagens TATU trocadas entre os dispositivos sensoriais (FoT-Device) e o FoT-Gateway, as quais permitem requisitar e responder solicitações por dados dos sensores e ações dos atuadores.

O soft-iot-mapping-devices instancia um conjunto de objetos que representam os dispositivos, sensores e atuadores do FoT-Device. Esses objetos contém informações relacionadas aos dispositivos tais como identificador único, tipo do sensor/atuador e dados sobre localização. Além disso, estão disponíveis para acesso pelos demais módulos da plataforma, servindo como interface de acesso aos sensores e/ou atuadores conectados na plataforma.

Para instalação do soft-iot-mapping-devices é necessário executar os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-mapping-devices/1.0.0
```

[13]https://github.com/WiserUFBA/soft-iot-mapping-devices

Após a instalação do soft-iot-mapping-devices é necessário gerar um arquivo de configuração que irá conter informações referentes aos dispositivos conectados a plataforma. Um modelo do arquivo pode ser encontrado em:

```
https://github.com/WiserUFBA/soft-iot-mapping-devices/blob/master/src
/main/resources/br.ufba.dcc.wiser.soft_iot.gateway.mapping_devices.cfg
```

E deve ser armazenado no seguinte diretório:

```
<diretorio_servicemix>/etc
```

O código abaixo apresenta um exemplo do arquivo de configuração do soft-iot-mapping-devices para alguns dispositivos com sensores:

```
#JSON with array of devices:
DevicesConnected=[{"id":"mydevice01", "latitude":53.290411,
"longitude":-9.074406,
 "sensors":[{"id":"temp01", "type":"temperatureSensor",
 "collection_time":30000, "publishing_time": 60000},
 {"id":"humd01", "type":"humiditySensor",
 "collection_time":30000,  "publishing_time": 60000}]},
 {"id":"sonoff01", "latitude":53.290457, "longitude":-9.074423,
  "sensors":[{"id":"temp01", "type":"temperatureSensor",
  "collection_time":30000,  "publishing_time": 60000},
 {"id":"humd01", "type":"humiditySensor", "collection_time":30000,
 "publishing_time": 60000},
 {"id":"sound01", "type":"SoundSensor", "collection_time":30000,
 "publishing_time": 60000},
 {"id":"light01", "type":"LightSensor", "collection_time":30000,
 "publishing_time": 60000},
 {"id":"dust01", "type":"AirPollutantSensor", "collection_time":30000,
 "publishing_time": 60000}]}]

# Habilita ou desabilitar o debug mode
debugMode=false
```

A configuração descreve que existem dois dispositivos conectados a plataforma com seguintes identificadores: `mydevice01`, e `sonoff01`. Cada dispositivo possui dados em relação a localização (latitude e longitude) e seus respectivos sensores. Já cada sensor dos dispositivos conectados possui um identificador, um tipo e os tempos de coletas de dados e publicação no broker MQTT.

6.3.2.4. Módulo soft-iot-local-storage

O **soft-iot-local-storage**[14] é responsável por promover a coleta, armazenamento e acesso interno dos dados produzidos pelos sensores do FoT-Device. Neste módulo concentra-se as funcionalidades referentes ao armazenamento interno dos dados sensoriais, tal como o acesso a estes dados pelos demais módulos da plataforma SOFT-IoT.

Com o suporte do soft-iot-mapping-devices para tradução das mensagens TATU e identificação dos dispositivos conectados a plataforma, o soft-iot-local-storage requisita fluxo dados de cada sensor conectado na plataforma em uma frequência definida pelo

[14]https://github.com/WiserUFBA/soft-iot-local-storage

usuário. As mensagens de requisição e respostas são trocada no broker MQTT implementado pelo módulo soft-iot-vertx-mqtt-broker para por fim serem armazenadas no banco de dados relacional Apache H2.

A Figura 6.23 apresenta um diagrama de sequência das principais operações do soft-iot-local-storage com os seus relacionamentos. Com os dispositivos conectados na plataforma, através da configuração do usuário (*1.set a JSON with connected devices*, Fig. 6.23), o soft-iot-local-storage obtém do módulo soft-iot-mapping-devices os sensores conectados e seus respectivos fluxos de coleta de dados (*2. getConnectedDevices()*, Fig. 6.23) para publicar no soft-iot-vertx-mqtt-broker as requisições de dados do sensores (*3.setFLOWSenrsorsDevices()*, Fig. 6.23). Com a requisição dos dados, os sensores irão publicar no soft-iot-vertx-mqtt-broker as informações capturadas na frequência configurada. O soft-iot-local-storage por sua vez irá coletar tais dados (*4.getDataSensors()*, Fig. 6.23) e armazená-los no banco de dados Apache H2 (*5.StoreSensorData()*, Fig. 6.23). O soft-iot-local-storage também implementa um procedimento para remoção de dados antigos, o qual a frequência pode ser configurada pelo usuário (*6.cleanOldData()*, Fig. 6.23) e módulos da camada de aplicação podem solicitar dados dos sensores através do soft-iot-local-storage (7.*requestDataSensor(), Fig. 6.23*).

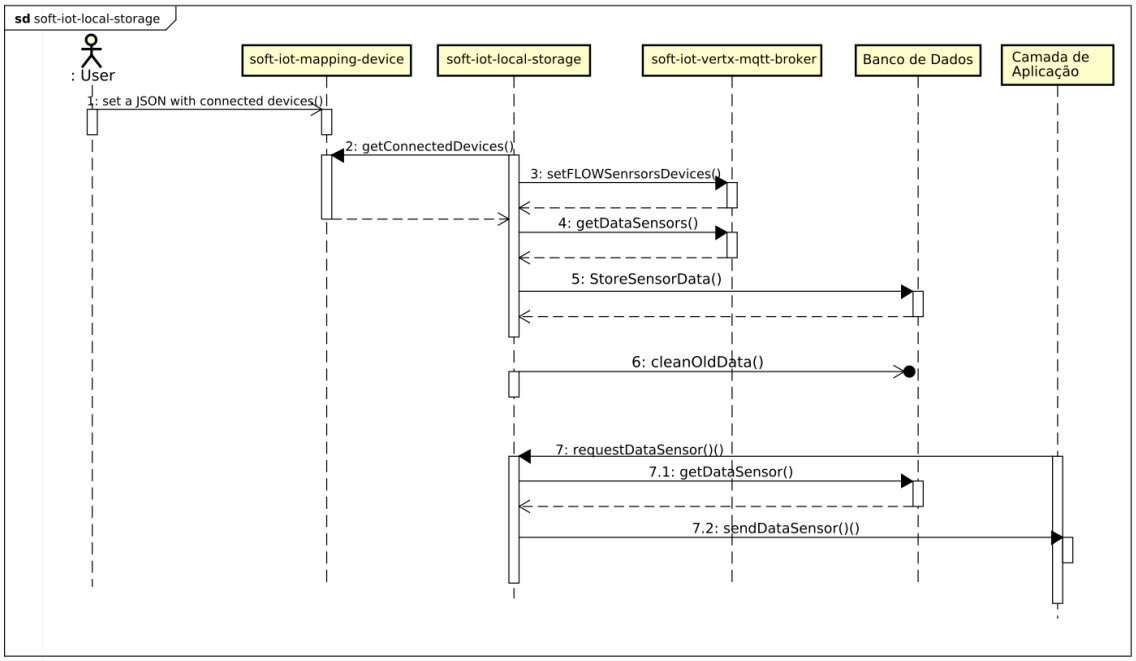


Figure 6.23. Diagrama de sequência com as operações e relacionamentos do soft-iot-local-storage

Antes de propriamente instalar o módulo soft-iot-local-storage é necessário introduzir ao ServiceMix alguns módulos que darão suporte ao banco de dados Apache H2. Assim é necessário executar os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> feature:repo-add mvn:org.ops4j.pax.jdbc/pax-jdbc
-features/0.8.0/xml/features
karaf@root()> feature:install transaction jndi pax-jdbc-h2 pax-jdbc
-pool-dbcp2 pax-jdbc-config
```

Com Apache H2 instalado é necessário também criar um arquivo de configuração para o banco de dados das informações coletadas nos sensores. O arquivo é:

```
<diretorio_servicemix>/etc/org.ops4j.datasource-gateway.cfg
```

E deve ter o seguinte conteúdo:

```
osgi.jdbc.driver.name=H2-pool-xa
url=jdbc:h2:${karaf.data}/soft-iot-local-storage
dataSourceName=soft-iot-local-storage
```

Este arquivo é responsável como indicar um driver para operar as transações no banco de dados, tal como o arquivo em que as inserções serão armazenadas.

É importante observar que o soft-iot-local-storage é dependente dos módulos soft-iot-mapping-devices e soft-iot-vertx-mqtt-broker, para correto funcionamento. Por fim, para instalação do soft-iot-local-storage é necessário executar os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-local-storage/1.0.0
```

Após a instalação do soft-iot-local-storage é necessário gerar um arquivo de configuração que irá conter informações referentes aos dispositivos conectados a plataforma. Um modelo do arquivo pode ser encontrado em:

```
https://github.com/WiserUFBA/soft-iot-local-storage/
src/main/resources/br.ufba.dcc.wiser.soft_iot.local_storage.cfg
```

E deve ser armazenado no seguinte diretório:

```
<diretorio_servicemix>/etc
```

O código abaixo apresenta um exemplo do arquivo de configuração do soft-iot-local-storage:

```
#Endereco do broker MQTT
MQTTHost=localhost

#Porta utilizada pelo broker MQTT
MQTTPort=1883

#Usuario e senha do broker MQTT. Se o broker MQTT
#aceita conexoes anonimas, eh possivel manter os campos vazios
MQTTUsername=
MQTTPassword=

#Nome da conexecao estabelecida com MQTT broker.
#Usado somente em opercoes internas.
MQTTServerId=FoTGatway

#Tempo padrao para coleta de dados, caso o sensor nao possua
#configuracao propria
DefaultCollectionTimeSensorData=300000
#Tempo padrao para publicacao de dados, caso o sensor nao possua
#configuracao propria
```

```
DefaultPublishingTimeSensorData=900000

#Numero de horas de dados que o sistema ira manter armazenado
NumberOfHoursDataStored=24

# Habilitar ou desabilitar o debug mode
debugMode=false
```

6.3.3. Camada de Aplicação e Negócios

A camada de aplicação na plataforma SOFT-IoT é responsável por fornecer os serviços solicitados pelos usuários. As aplicações podem ser implantados em dispositivos de capacidade limitada (e.g. Raspberry Pi (FoT-Gateway)) e servidores que estão localizados na borda rede (FoT-Server) ou em servidores localizados na Cloud (Cloud-Server).

Na SOFT-IOT as aplicações são construídas em linguagem de programação Java, empacotadas como um bundle OSGI e funcionam em um barramento de serviços conforme explicado na Seção 6.3.2.1. O bundle é a unidade de implementação principal ao usar o OSGi. É basicamente um arquivo jar contendo alguns cabeçalhos adicionais no MANIFEST usado pelo *framework* OSGi, no caso da SOFT-IOT, o Karaf[15]. A Figura 6.4 (veja camada de aplicação) ilustra as aplicações implementadas a partir da plataforma SOFT-IOT.

O módulo **soft-iot-data-aggregation** é responsável por agregar dados em FoT-Gateways. O módulo **soft-iot-semantic-enrichment** enriquece os dados do sensor, no FoT-Gateway, com descrições da Web Semântica. O módulo **soft-iot-data-aggregation** fornece agregação de dados a partir de dados semânticos. O módulo **soft-iot-web-service** fornece serviços como a API RESTful para acesso aos dispositivos IoT. O módulo **soft-iot-rules** permite aos usuários a criação de regras no padrão ECA (Event-Condition Action) para posterior execução de ações na borda rede. O módulo **soft-iot-semantic-observer** é responsável por observar alterações feitas em um modelo semântico para posterior tomada de decisão. O módulo **soft-iot-semantic-model** é responsável em obter informações semanticamente descritas por triplas de RDF que estão implantados no FoT-Server (Apache Fuseki). O módulo **soft-iot-semantic-reasoner** é responsável por fazer inferências sobre as regras criadas (a partir do módulo soft-iot-rules) pelo usuário.

Nas próximas seções explicaremos em detalhes essas aplicações e o processo de implantação na SOFT-IOT.

6.3.3.1. Módulo soft-iot-data-aggregation

Módulo da plataforma SOFT-IoT para agregar dados no FoT-Gateway. Esse módulo coleta dados não agregados do banco de dados, aplica funções de agregação e armazena os dados resultantes novamente no banco de dados.

Este módulo possui como pré-requisito a instalação dos módulos soft-iot-gateway-mapping-devices (veja Seção 6.3.2.3) e soft-iot-gateway-local-storage (veja Seção 6.3.2.4).

Para instalar este bundle execute os seguintes comandos no prompt do Karaf:

[15]https://karaf.apache.org/

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/soft-iot--
mapping-devices/1.0.0
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/soft-iot-lo
cal-storage/1.0.0
karaf@root()> bundle:install -s mvn:br.ufba.dcc.wiser.soft_iot.soft-iot
-data-aggregation/1.0.0
```

O módulo soft-iot-data-aggregation [16] possui um arquivo de configuração, onde é possível definir informações sobre o tempo de execução do procedimento de agregação e configurar a função de agregação para cada sensor.

Finalmente, para a correta execução do módulo você precisa copiar o arquivo:

```
https://github.com/WiserUFBA/soft-iot-data-aggregation/blob/master/src
/main/resources/br.ufba.dcc.wiser.soft_iot.data_aggregation.cfg
```

Para o diretório:

```
<servicemix_directory>/etc
```

6.3.3.2. Módulo soft-iot-semantic-enrichment

O módulo soft-iot-semantic-enrichment[17] possui como objetivo enriquecer os dados do sensor, no FoT-Gateway, com anotações semânticas. Possui como pré-requisito a instalação da biblioteca do Jena 3.1.0. Esta biblioteca não possui uma versão estável do bundle jena-osgi. Por essa razão, precisamos instalá-lo manualmente através da versão compilada no diretório *fot-gateway-semantic-enrichmen/jena-gateway.kar*.

Então, é necessário copiar:

```
https://github.com/WiserUFBA/soft-iot-semantic-enrichment
/tree/master/jena-gateway.kar
```

Para o diretório:

```
<servicemix_directory>/deploy
```

Além disso, este módulo possui também como pré-requisito a instalação dos módulos soft-iot-gateway-mapping-devices (veja Seção 6.3.2.3) e soft-iot-gateway-local-storage (veja Seção 6.3.2.4).

Para instalar este bundle execute os seguintes comandos no prompt do Karaf:

```
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot
/soft-iot-mapping-devices/1.0.0
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot
/soft-iot-local-storage/1.0.0
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot
/soft-iot-semantic-enrichment/1.0.0
```

Finalmente, para a correta execução do módulo você precisa copiar o arquivo:

[16] https://github.com/WiserUFBA/soft-iot-semantic-data-aggregation

[17] https://github.com/WiserUFBA/soft-iot-semantic-enrichment

```
https://github.com/WiserUFBA/soft-iot-semantic-enrichment
/blob/master/src/main/resources/
br.ufba.dcc.wiser.soft_iot.semantic_enrichment.cfg
```

Para o diretório:

```
<servicemix_directory>/etc
```

6.3.3.3. Módulo soft-iot-server-semantic-data-aggregation

O módulo soft-iot-server-semantic-data-aggregation[18] realiza a agregação de dados, onde são coletados dados de um banco de dados semântico. O resultado do procedimento de agregação também é armazenado na base de dados semântico. Além disso, o módulo soft-iot-server-semantic-data-aggregation permite configurar a função de agregação para cada tipo de sensor.

Para instalar este bundle, você precisa instalar previamente estas dependências no karaf:

```
karaf@root()>feature:repo-add mvn:org.ops4j.pax.jdbc/
pax-jdbc-features/0.8.0/xml/features
karaf@root()>feature:install transaction jndi
pax-jdbc-h2 pax-jdbc-pool-dbcp2 pax-jdbc-config
```

Esse módulo necessita da biblioteca do Jena 3.1.0 para funcionar corretamente. Esta biblioteca não possui uma versão estável do bundle jena-osgi. Assim, precisamos instalá-lo manualmente através da versão compilada no diretório soft-iot-server-semântica-data-aggregation/jena-gateway.kar. Então, necessitamos copiar:

```
https://github.com/WiserUFBA/soft-iot-semantic-data-aggregation
/tree/master/jena-gateway.kar
```

Para o diretório:

```
<servicemix_directory>/deploy
```

Depois disso, será necessário criar um arquivo com a configuração do banco de dados. O nome do arquivo é **etc/org.ops4j.datasource-server.cfg** (no diretório servicemix). O conteúdo do arquivo é:

```
osgi.jdbc.driver.name=H2-pool-xa
url=jdbc:h2:${karaf.data}/fot-server-control
dataSourceName=fot-server-control
```

Finalmente, para a correta execução do módulo é necessário copiar o arquivo:

```
https://github.com/WiserUFBA/
soft-iot-semantic-data-aggregation/blob/master/src/main/resources/
br.ufba.dcc.wiser.soft_iot.semantic_data_aggregation.cfg
```

Para o diretório:

```
<servicemix_directory>/etc
```

[18]https://github.com/WiserUFBA/soft-iot-semantic-data-aggregation

6.3.3.4. Módulo soft-iot-web-service

O módulo soft-iot-web-service[19] expõe os dados do sensor do sistema IoT por meio de um serviço web RESTful. Ele acessa os dados armazenados no banco de dados local, gerenciados pelo armazenamento local do módulo soft-iot-local-storage (veja Seção6.3.2.4), permitindo que os usuários obtenham dados e informações no formato JSON sobre os sensores.

Para instalar este bundle, é necessário instalar previamente estas dependências no karaf:

```
karaf@root()>bundle:install mvn:org.codehaus.jackson/jackson-jaxrs/
1.9.2
karaf@root()>bundle:install mvn:org.codehaus.jackson/jackson-core-asl/
1.9.2
karaf@root()>bundle:install mvn:org.codehaus.jackson/jackson-mapper-asl/
1.9.2
```

Este módulo possui também como pré-requisito a instalação dos módulos soft-iot-gateway-mapping-devices (veja Seção 6.3.2.3) e soft-iot-gateway-local-storage (veja Seção 6.3.2.4).

Para instalar este bundle execute os seguintes comandos no prompt do Karaf:

```
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-mapping-devices/1.0.0
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-local-storage/1.0.0
karaf@root()>bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-iot-service/1.0.0
```

Este módulo cria um Serviço Web RESTful que é possível coletar dados de sensores e informações sobre os dispositivos conectados.

Para obter os dispositivos conectados:

```
http://<servicemix-urls>:<servicemix-port>/cxf
/iot-service/device/{device_id}
```

Para obter os dados do sensor:

```
http://<servicemix-urls>:<servicemix-port>/cxf/iot-service/
device/{device_id}/{sensor_id}
```

Para obter dados do sensor em um intervalo de tempo:

```
http://<servicemix-urls>:<servicemix-port>/cxf/iot-service
/device/{device_id}/{sensor_id}/{start_datetime}/{end_datetime}
```

Para obter mais informações sobre a descrição sintática do serviço web RESTful, acesse:

```
http://<servicemix-urls>:<servicemix-port>/cxf/iot-service?_wadl
```

[19]https://github.com/WiserUFBA/soft-iot-iot-service

6.3.3.5. Módulo soft-iot-rules-editor

O módulo soft-iot-rules-editor[20] é uma aplicação construída em Android e possibilita ao usuário criar regras visualmente no formato EVENT, CONDITION e ACTION (ECA). Na Figura 6.24, o usuário escolhe em uma lista o sensor para o qual deseja criar regras e informa os valores apropriados.

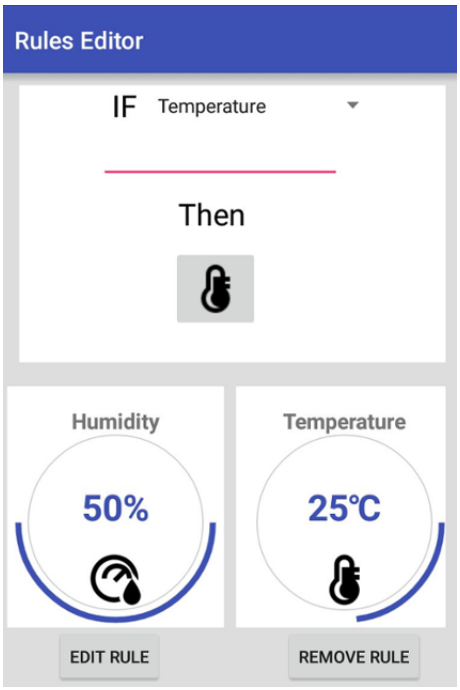


Figure 6.24. Aplicação mobile para criação de regras no formato ECA.

Essa versão do módulo soft-iot-rules-editor é uma versão de teste e está disponibilizada em:[21]. Este módulo possui como pré-requisito a instalação dos módulos soft-iot-semantic-observer (veja Seção 6.3.3.6), soft-iot-semantic-reasoner (veja Seção 6.3.3.7) e soft-iot-rules-execution (veja Seção 6.3.3.8).

6.3.3.6. Módulo soft-iot-semantic-observer

O módulo observador semântico[22] é responsável por observar as alterações feitas em um modelo semântico. Para fazer isso, o módulo soft-iot-semantic-observer usa consultas SPARQL, como mostra a Listagem 6.2. Conforme mostrado na Listagem 6.3, na linha 10, inicialmente, a propriedade isSettingFor é **false** e, após executar uma regra, essa propriedade pode ser alterada para **true** pelo módulo soft-iot-semantic-reasoner. Então, o módulo Semantic Observer, ao identificar mudanças feitas na propriedade isSettingFor, notifica o módulo de Execução de Regras (Seção 6.3.3.8), que é responsável por informar ao atuador e ativar o dispositivo (por exemplo, ativar um ar condicionado,ligar uma luz).

[20]https://github.com/WiserUFBA/Rules-Editor
[21]https://github.com/WiserUFBA/Rules-Editor/blob/master/Rules-Editor.apk
[22]https://github.com/WiserUFBA/Semantic-Observer

Código 6.2. Consulta SPARQL para observar mudanças realizadas no modelo semântico.

```
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX dul: <http://www.loa-cnr.it/.../DUL.owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?hasDataValue
WHERE { <http://wiser.dcc.ufba.br/smartUFBA/...>
#obsValue_14915308050001491530865086>
ssn: ObservationValue ;
dul: hasDataValue '37'^^xsd:double ;
dul: isSettingFor true . };
```

Código 6.3. Trecho das triplas RDF armazenadas no servidor Fuseki.

```
<http://wiser.dcc.ufba.br/smartUFBA/devices/temp01>
dul: hasIntervalDate "2017-04-03T20:14:11.054Z" .
<http://wiser.dcc.ufba.br/smartUFBA/devices/...>
a dul: TimeInterval ;
dul: hasIntervalDate "2017-04-03T20:16:11.207Z" .
 <http://wiser.dcc.ufba.br/smartUFBA/devices/...>
a ssn: ObservationValue ;
dul: hasDataValue "29"^^xsd:double .
dul: isSettingFor false.
```

A instalação desse bundle pode ser realizada a partir do console do karaf ou do web console do servicemix conforme ilustrado na Figura 6.22.

6.3.3.7. Módulo soft-iot-semantic-reasoner

O módulo soft-iot-semantic-reasoner[23] é responsável por fazer inferências, a partir do *framework* Jena, sobre as regras criadas pelo usuário.

Quando o módulo soft-iot-semantic-reasoner executa uma regra e infere que uma condição foi satisfeita, ela executa uma atualização no modelo. Esta atualização é executada apenas nos triplas de RDF que atendem à regra usando o SPARQL UPDATE de acordo com a Listagem 6.4. Na linha 4, as informações contidas no modelo são excluídas e, na linha 5, a mesma propriedade é inserida com um novo valor.Na linha 6, uma restrição é aplicada para identificar corretamente o valor a ser alterado.

A instalação desse bundle pode ser realizada a partir do console do karaf ou do web console do service mix conforme ilustrado na Figura 6.22.

Código 6.4. Atualização SPARQL para atualizar o modelo semântico.

```
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX dul:
<http://www.loa-cnr.it/ontologies/DUL.owl#>
PREFIX xsd:
<http://www.w3.org/2001/XMLSchema#>
DELETE { <%s> dul:isSettingFor false .}
INSERT { <%s> dul:isSettingFor true .}
WHERE { <%s> dul:isSettingFor false .}
```

[23]https://github.com/WiserUFBA/Rules-Semantic-for-IoT

6.3.3.8. Módulo soft-iot-rules-execution

O módulo soft-iot-rules-execution[24] executa os serviços Web RESTful para ativar os atuadores (por exemplo, condicionador de ar, janela, alarme, semáforo, etc.) enviando mensagens por meio do MQTT protocolo.

Para ativar um dispositivo:

```
http://<servicemix-urls>:<servicemix-port>/cxf/lamp/devices/actuator/lamp"
```

6.3.4. Camada de Segurança

Ao mesmo tempo que a IoT possibilita uma gama de novas aplicações, ela apresenta desafios em diferentes níveis de sua infra-estrutura. Os sistemas de IoT integram objetos físicos, dados de sensores e recursos de computação em redes amplas através da Internet. Particularmente, o grande volume de dados gerados e a exposição a que esses dados estão sujeitos ocasionam diferentes problemas de segurança e privacidade.

Nesse cenário, possíveis vulnerabilidades de segurança e violações de privacidade precisam ser resolvidas com base na confiança entre as partes envolvidas e pelo uso de mecanismos adequados. Dessa forma, os desenvolvedores podem empregar tais mecanismos para criar soluções escaláveis, distribuídas e confiáveis.

As arquiteturas de referência apresentadas em [Bassi et al. 2013] e [Pöhls et al. 2014] discutem os possíveis problemas de segurança e definem modelos de confiança, segurança e privacidade para sistemas IoT. Atualmente, os serviços de segurança IoT, como autenticação, controle de acesso e gerenciamento de identidade estão apoiadas em uma arquitetura cliente-servidor [Zhu and Badr 2018]. Eles se baseiam na suposição de que os usuários devem depositar toda a confiança em entidades terceiras confiáveis (provedores de identidade) que possuem dados pessoais de usuários e podem ver todas as transações entre os usuários e os provedores de serviços.

Neste sentido, Blockchain é uma tecnologia emergente que oferece suporte distribuído confiável e seguro para realização de transações entre membros que não necessariamente têm confiança entre si e que estão dispersos em larga escala numa rede P2P. É considerada uma tecnologia disruptiva, pois cria digitalmente uma entidade de confiança descentralizada, eliminando a necessidade de uma terceira parte de confiança. Os recentes avanços nessa tecnologia criam perspectivas para desenvolvimento de sistemas robustos e seguros em diversos escopos da computação, particularmente em IoT e Névoa.

Em [Greve et al. 2018], Blockchain é definida como o resultado de uma engenhosa combinação de técnicas robustas provenientes da computação distribuída confiável (tolerância a falhas bzyantinas, sistemas P2P), criptografia (chave assimétrica, funções hash, desafios criptográficos) e teoria dos jogos (mecanismos de incentivos). Agrega elementos eficazes para a implementação de um sistema de acordo em escala global, trazendo respostas para importantes desafios computacionais, dentre os principais: (i) realização de consenso numa rede aberta, com participantes desconhecidos, em escala planetária; (ii) tolerância a falhas byzantinas, com participantes anônimos; (iii) resistência ao ataque de duplo-gasto (*double-spending*), garantindo que ativos transacionados (como moedas

[24]https://github.com/WiserUFBA/Rules-Execution

digitais) não sejam gastos duplamente, para além do seu valor em posse; (iv) resistência a ataques Sybil, garantindo que usuários maliciosos não poderão se personificar em outros (a menos que tenham acesso a sua chave privada); (v) garantia da auditabilidade, autenticidade, não-repúdio e integridade em escala global de todas as transações validadas e armazenadas no livro-razão (*ledger*) distribuído.

O projeto SOFT-IoT vem investigado estratégias de confiança, segurança e privacidade que exploram a consistência eventual e as garantias oferecidas pelas tecnologias de livro-razão distribuído. Por exemplo, sem um gerenciamento de identidade apropriado para IoT, outros serviços de segurança voltados à computação em névoa não podem ser construídos corretamente. Os avanços na aplicação da tecnologia Blockchain em IoT podem revelar uma possível solução para garantir o gerenciamento de confiança em sistemas descentralizados [Abadi et al. 2018]. Pesquisas atuais vem fomentando soluções de gerenciamento de confiança baseadas em Blockchain com foco em sistemas de gerenciamento de identidade (IDMS) para a IoT [Zhu and Badr 2018]. No entanto, a integração da IoT com Blockchain é recente e carece de uma maior investigação.

A Figura 6.4 apresenta a integração da arquitetura SOFT-IoT com a Blockchain através de uma camada computacional para compartilhamento e análise segura dos dados, com garantias de privacidade. Essa camada pode ser utilizada para autenticar, autorizar, controlar e auditar os dados gerados pelos objetos inteligentes [Christidis and Devetsikiotis 2016]. Sua disposição vertical evidencia a possibilidade do provimento e integração de serviços baseados em Blockchain em diferentes níveis da arquitetura SOFT-IoT.

6.4. Tópicos de pesquisas relacionados a SOFT-IoT

Nesta seção, alguns trabalhos de pesquisa relacionados à arquitetura SOFT-IoT são apresentados. O objetivo é oferecer uma visão geral do desenvolvimento da plataforma visando uma aprendizagem dos conceitos e inspiração para novos projetos.

6.4.1. Microserviços Reativos

Os Microsserviços permitem a criação de um sistema a partir de uma coleção de serviços pequenos e isolados, capazes de gerenciar seus próprios dados [Lewis and Fowler 2014]. O REST, que produz comunicação síncrona entre os serviços, é frequentemente usado no desenvolvimento de Microsserviços, entretanto, a comunicação entre Microsserviços deve se basear no uso de mensagens assíncronas. Isso é necessário para estabelecer um limite assíncrono entre cada serviço, com o objetivo de desacoplar o fluxo de comunicação de serviços das perspectivas de tempo e espaço. O fluxo de comunicação desacoplado no tempo permite a simultaneidade, enquanto o desacoplamento no espaço permite a mobilidade e a distribuição que são indispensáveis para o desenvolvimento de aplicativos em cenários de IoT. Operações assíncronas e sem bloqueio reduzem o congestionamento de recursos no sistema e produzem escalabilidade e baixa latência. Além disso, a comunicação baseada em mensagens assíncronas produz sistemas com duas características:

- **Resiliência**, que está associada à capacidade do sistema de lidar com falhas;
- **Elasticidade**, que está associada à capacidade do sistema de dimensionar horizontalmente. Sistemas que têm seu design construído a partir dessas perspectivas são considerados reativos.

Este tópico estende a SOFT-IOT e propõe uma abordagem para modelar o design e a implementação de Microsserviços reativos em cenários IoT.

Para esse fim, definimos um modelo arquitetural para o desenvolvimento de Microserviços e criamos uma nova plataforma baseada no Vert.x[25]. Vert.x é um toolkit que permite a criação de aplicações reativas na JVM. Outra característica é a possibilidade de trabalhar com o Vert.x em diferentes linguagem de programação como Java, JavaScript, Groovy, Ruby, Ceylon, Scala and Kotlin.

6.4.2. Análise de Data Stream utilizando a SOFT-IoT

Os ambientes que são potencializados pela Internet das Coisas devem produzir uma enorme quantidade de dados, criando assim, uma nova área de atuação dentro do ecossistema da IoT que é denominada *IoT Analytics*. Essa área é caracterizada pela análise de dados de diversas fontes de dados IoT, que podem ser sensores, atuadores, dispositivos inteligentes e outros objetos conectados à Internet, sendo considerada como uma das principais partes que pode realizar todo potencial de mercado da IoT [Soldatos 2016].

Os dados de várias fontes e domínios produzidos pela IoT são em alguns casos fluxos de dados, como por exemplo, dados numéricos de diferentes sensores ou entradas de texto de mídias sociais. Os fluxos de dados comuns geralmente seguem a distribuição Gaussiana durante um longo período. Porém, os dados IoT são produzidos em curto espaço de tempo e em grandes quantidades, apresentando uma variedade de distribuições esporádicas ao longo do tempo. Além disso, em alguns casos em tempo real ou próximo do tempo real [Puschmann et al. 2017].

Uma tendência das aplicações para Internet das Coisas que tratam do conceito de *IoT Analytics* é a utilização da Computação em Névoa que pode descentralizar o processamento de fluxos de dados IoT e, apenas realizar a transferência de dados IoT filtrados dos dispositivos da borda da rede para a nuvem [Andrade et al. 2017]. Diante disso, a associação da análise de fluxos dados com a Computação em Névoa permite que as empresas explorem em tempo real os dados produzidos pela IoT e, com isso produzam valor de negócio. Essas análises na borda da rede podem resolver o problema das aplicações que necessitam de resposta em tempo real, como por exemplo, os sistemas que monitoram saúde de pacientes e devem tomar decisões em curto espaço de tempo. Outros tipos de análises podem descobrir preferências dos clientes, padrões ocultos nos dados, entre outras informações que podem ajudar as organizações a tomarem decisões mais fundamentadas e em curto espaço de tempo.

Diante disso, um módulo chamado **FoT-Gateway-StreamAnalytics-Soft_Iot**[26] foi desenvolvido para realizar mineração de fluxo de dados IoT na *Fog Computing* utilizando como base a plataforma SOFT-IoT. Uma arquitetura para análise de *data stream* com a SOFT-IoT tem como objetivo utilizar toda capacidade computacional dos dispositivos empregados na *Fog of Things* para o processamento e análise de dados, sem necessidade de utilização constante da *Cloud Computing*. Diante disso, esta proposta pode minimizar o volume de dados que precisam ser encaminhados para a nuvem, o que pode ter um grande impacto nos tempos de resposta e nos custos de transmissão dos dados na

[25]https://vertx.io/

[26]https://github.com/WiserUFBA/FoT-Gateway-StreamAnalytics-Soft_Iot

rede, além disso, permite análises de dados em tempo real na *Fog Computing*.

Para a instalação do FoT-Gateway-StreamAnalytics-Soft_Iot é necessário ter configurado o módulo soft-iot-mapping-devices como apresentado na Seção 6.3.2.3. A instalação do FoT-Gateway-StreamAnalytics-Soft_Iot pode ser feita executando os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot.analytics/
fot-gateway-streamanalytics-soft_iot/1.0.0
```

Logo após a instalação do FoT-Gateway-StreamAnalytics-Soft_Iot é necessário gerar um arquivo de configuração (seguindo o padrão da seção 6.3.2.3) que irá conter a configuração dos seguintes parâmetros do módulo: dispositivos que devem ser conectados no módulo; ativação do modo de depuração; endereço e configuração do *broker* MQTT; tempo de publicação e coleta de dados dos sensores. Um modelo do arquivo pode ser encontrado em:

```
FoT-Gateway-StreamAnalytics-Soft_Iot/src/main/resources/br/ufba/dcc/
wiser/soft_iot/analytics/
br.ufba.dcc.wiser.soft_iot.gateway.stream_analytics.cfg
```

O arquivo de configuração deve ser implantado no sub-diretório do ServiceMix:

```
<diretorio_servicemix>/etc
```

6.4.3. Ambientes de Testes para Computação em Névoa

É esperado que os ambientes de névoa suportem milhões de dispositivos IoT e de usuários finais [Coutinho et al. 2016]. Eles podem envolver um grande número de aplicativos, domínios de redes e nós de névoa. Além disso, suas diferentes aplicações podem ter uma grande quantidade de componentes que usam tanto recursos de nuvem quando de névoa para fornecer soluções inteligentes e avançadas.

Um problema atual nesta área é a falta de ambientes de suporte para prototipagem e teste de serviços, componentes e aplicações em larga escala [Mouradian et al. 2017]. Neste momento, simuladores de rede e *middleware* de nuvem são adaptados para permitir uma avaliação experimental de soluções de névoa em ambientes limitados, cenários específicos e condições restritivas. No entanto, testar e validar uma arquitetura com poucos dispositivos e nós não garante que ela seja propriamente executada sobre um ambiente em larga escala, no que diz respeito à qualidade e desempenho da solução desenvolvida.

Apoiado pelo projeto SOFT-IoT, o *framework* Fogbed [Coutinho et al. 2018a] foi desenvolvido para permitir a prototipagem rápida e o teste escalável de componentes e aplicativos de névoa. Seu projeto é compatível com tecnologias do mundo real e atende aos requisitos de configuração flexível e de baixo custo, suportando a integração de sistemas de terceiros por meio da implementação de interfaces padronizadas.

O Fogbed é baseado em tecnologias que são amplamente empregadas por desenvolvedores de software livre e pesquisadores como contêineres Docker[27]. e o emulador de rede Mininet [de Oliveira et al. 2014]. Usando uma abordagem *desktop*, o Fogbed estende o *framework* Mininet para criar ambientes de experimentação (*testbeds*) de névoa

[27]https://www.docker.com

virtualizados. Ele permite a implementação de nós de névoa como contêineres Docker sob diferentes configurações de rede. A API do Fogbed fornece funcionalidades para adicionar, conectar e remover contêineres dinamicamente da topologia da rede.

Essas características possibilitam a emulação da infraestrutura de nuvem e névoa, na qual é possível iniciar e interromper instâncias de computação em qualquer momento. Além disso, é possível alterar as limitações de recursos em tempo de execução para um contêiner, como o tempo da CPU e memória disponível. Utilizando a API de emulação Maxinet [Wette et al. 2014], esses contêineres podem ser executados em uma ou mais (*cluster*) máquinas hospedeiras (*hosts*) através de uma rede local [Coutinho et al. 2018b].

6.4.3.1. Instalação do Ambiente Fogbed

Na versão atual, existem duas opções para instalação do Fogbed em uma máquina hospedeira: (*i*) pode ser instalado nativamente (*bare metal*) sobre o sistema operacional; ou (*ii*) pode ser executado como um contêiner Docker privilegiado (*privileged Docker container*). Em ambas as opções é requerido no mínimo o Linux Ubuntu versão 16.04 LTS.

Na opção (*i*), uma instalação automática é fornecida através de um *playbook* da ferramenta Ansible. A seguinte sequência de comandos pode ser aplicada em um terminal do sistema operacional Linux hospedeiro, onde a senha do usuário administrador pode ser solicitada para permitir a execução do comando *sudo*:

```
$ sudo apt-get install ansible git aptitude
$ git clone https://github.com/fogbed/fogbed.git
$ cd fogbed/ansible
$ sudo ansible-playbook -i "localhost," -c local install_metis.yml
$ sudo ansible-playbook -i "localhost," -c local install_docker.yml
$ sudo ansible-playbook -i "localhost," -c local --skip-tags \
"notindocker" install_fogbed.yml
```

A menos que se esteja tentando obter o melhor desempenho de emulação, é recomendado realizar a instalação em uma máquina virtual (*Virtual Machine* - VM) ao invés de uma máquina física. Este cuidado deve-se ao fato do processo de instalação alterar as configurações do sistema, podendo afetar seu funcionamento caso seja mal sucedido.

Na opção (*ii*), o ambiente Fogbed é executado em um contêiner Docker privilegiado. Como qualquer aplicação baseada em Linux, o Fogbed também pode ser conteinerizado como uma imagem Docker. Essas imagens do Fogbed podem então ser replicadas, distribuídas e instanciadas em diferentes máquinas hospedeiras. Uma imagem Docker do Fogbed pré-compilada está disponível pela Internet através da biblioteca pública de imagens Docker Hub. O seguinte comando pode então ser aplicado em um terminal do sistema Linux hospedeiro para construção de uma imagem do Fogbed localmente:

```
$ docker build -t fogbed .
```

Também é possível utilizar a imagem construída mais recentemente:

```
$ docker pull fogbed/fogbed
```

Por fim, o seguinte comando é usado para instanciar um contêiner do Fogbed:

```
$ docker run --name fogbed -it --rm --privileged --pid='host' -v \
/var/run/docker.sock:/var/run/docker.sock fogbed /bin/bash
```

6.4.3.2. Configurando o Ambiente de Névoa

Uma emulação no *Fogbed* é configurada através da definição de instâncias virtuais, nós virtuais, comutadores virtuais e conexões virtuais. Estas definições são empregadas para a criação de um ambiente de névoa virtual executado em uma máquina hospedeira. A configuração flexível do ambiente é alcançada através do uso de imagens de contêiner *Docker*. Cada nó virtual é instanciado a partir de uma imagem de contêiner pré-configurada e que compreende parte de uma aplicação distribuída, bem como seus serviços e protocolos. Diferentes tipos de imagens de contêiner podem ser usados para instanciar nós virtuais. Perceba que se o ambiente Fogbed estiver sendo executado como um contêiner privilegiado, os nós virtuais estarão sendo executados como contêineres dentro de contêineres ou contêineres aninhados (*nested container deployment*).

Antes de iniciar uma emulação, é necessário criar um *script* que define a topologia da névoa e seus elementos virtuais. Um *script* de topologia pré-definido é fornecido para demostrar a configuração básica do ambiente. Assim, após a instalação do Fogbed em uma máquina virtual ou contêiner Docker, é possível iniciar uma emulação através do seguinte comando aplicado em um terminal da máquina convidada (*guest*):

```
$ python examples/virtual_instance_example.py
```

No *script* de topologia, a definição do ambiente é realizada usando a linguagem Python. Primeiramente, todos os elementos virtuais são descritos e, após sua inicialização, um experimento predefinido pode ser executado sobre o ambiente. Se os procedimentos de instalação do Fogbed foram realizados corretamente, a execução do *script* deve instanciar a topologia em névoa nele definida e executar alguns comandos básicos de teste (*ifconfig* e *ping*) em nós virtuais desta topologia. No exemplo, verificando o conteúdo do *script* de topologia, encontramos em suas primeiras linhas as seguintes declarações:

```
topo = FogTopo()
c1 = topo.addVirtualInstance("cloud")
f1 = topo.addVirtualInstance("fog")
e1 = topo.addVirtualInstance("edge")
```

Nas primeiras declarações temos o instanciamento de uma topologia em névoa seguida pela definição de três instâncias virtuais. Uma instância virtual é uma abstração do Fogbed que permite o gerenciamento de um conjunto de nós virtuais e comutadores virtuais relacionados como uma única entidade. Por exemplo, uma instância virtual pode ser formada por um ou mais nós virtuais conectados por um único comutador virtual, onde durante a emulação novos nós virtuais podem ser alocados em uma instância virtual.

A seguir, nas próximas linhas do *script* de topologia, são definidos três diferentes modelos de recursos. Cada modelo é atribuído para uma das instâncias virtuais:

```
erm = EdgeResourceModel(max_cu=20, max_mu=2048)
frm = FogResourceModel()
crm = CloudResourceModel()
e1.assignResourceModel(erm)
f1.assignResourceModel(frm)
c1.assignResourceModel(crm)
```

Cada instância virtual segue o modelo de recursos associado a ela, que define quantos recursos essa instância possui para distribuir entre os seus nós virtuais (contêineres) e qual a estratégia de gerenciamento desses recursos. Cada modelo de recurso permite definir um valor *max_cu* e *max_mu*, que representam as unidades máximas de processamento e de memória atribuídas à instância virtual. Assim, é possível determinar quantas unidades de processamento *cu* e de memória *mu* os contêineres podem consumir durante a emulação, calculando seus valores como frações proporcionais ao total de recursos disponíveis na instância virtual onde os contêineres estão sendo executados. Esses valores são convertidos em tempo real de CPU e limite disponível de memória. Por exemplo, se para um contêiner *a* for atribuído quatro unidades de processamento, e para um contêiner *b* duas unidades de processamento, e se ambos estiverem na mesma instância virtual, o contêiner *a* terá duas vezes mais tempo de CPU do que o contêiner *b*.

Existem três tipos de modelos de recursos pré-definidos no Fogbed: *EdgeResourceModel*, *FogResourceModel* e *CloudResourceModel*, onde os valores *defaults* de *max_cu* e *max_mu* são respectivamente 32 e 2048, respectivamente. A estratégia de gerenciamento utilizada em*EdgeResourceModel* possui um limite fixo em que, se um contêiner solicita recursos mas toda a capacidade da instância virtual já foi alocada, uma exceção é gerada alertando que a instancia virtual não pode alocar novos contêineres. Em *FogResourceModel* e *CloudResourceModel*, é utilizada uma estratégia de super provisionamento (*overprovisioning*) semelhante ao modelo empregado em nuvem onde, se um contêiner solicita recursos mas toda a capacidade da instância virtual já foi alocada, o novo contêiner é iniciado de qualquer maneira. Entretanto, em *FogResourceModel* o valor de *max_cu* e *max_mu* continuam os mesmos após o instanciamento do novo contêiner, e o limite de tempo de CPU e de memória de cada contêiner é recalculado.

O trecho seguinte do *script* de topologia é semelhante a exemplos de outros emuladores baseados em Mininet e define os outros elementos virtuais da névoa:

```
d1 = c1.addDocker('d1', ip='10.0.0.251', dimage="ubuntu:trusty")
d2 = f1.addDocker('d2', ip='10.0.0.252', dimage="ubuntu:trusty")
d3 = e1.addDocker('d3', ip='10.0.0.253', dimage="ubuntu:trusty")
d4 = topo.addDocker('d4', ip='10.0.0.254', dimage="ubuntu:trusty")
d5 = e1.addDocker('d5', ip='10.0.0.255', dimage="ubuntu:trusty",
    resources=PREDEFINED_RESOURCES['medium'])
d6 = e1.addDocker('d6', ip='10.0.0.256', dimage="ubuntu:trusty",
    resources=PREDEFINED_RESOURCES['large'])
```

Como mostrado acima, cada novo nó virtual é iniciado passando como parâmetro suas configurações (identificador, endereço IP, e a imagem de contêiner Docker utilizada), onde cada um, exceto o nó d4, é iniciado dentro de uma instância virtual. O parâmetro de recursos em d5 e d6 descreve a quantidade de recursos da instância virtual que o contêiner deve receber. Se não for especificado, o recurso predefinido como *small* é definido como *default*. No seguinte trecho do *script* de topologia é encontrada a lista predefinida de recursos:

```
PREDEFINED_RESOURCES = {
    "tiny": {"cu": 0.5, "mu": 32},
    "small": {"cu": 1, "mu": 128},
    "medium": {"cu": 4, "mu": 256},
    "large": {"cu": 8, "mu": 512},
```

```
    "xlarge": {"cu": 16, "mu": 1024},
    "xxlarge": {"cu": 32, "mu": 2048}
}
```

Se nenhum dos valores predefinidos forem adequados para um determinado nó virtual, é possível definir uma nova entrada para o contêiner específico. Após a criação de todos os nós virtuais, dois comutadores virtuais são instanciados, seguidos pelas definições de suas conexões virtuais usando os parâmetros de classe (*cls*), atraso (*delay*) e largura de banda (*bw*) entre os nós virtuais, as instâncias virtuais e os comutadores virtuais:

```
s1 = topo.addSwitch('s1')
s2 = topo.addSwitch('s2')
topo.addLink(d4, s1)
topo.addLink(s1, s2)
topo.addLink(s2, e1)
topo.addLink(c1, f1, cls=TCLink, delay='200ms', bw=1)
topo.addLink(f1, e1, cls=TCLink, delay='350ms', bw=2)
```

6.4.3.3. Executando Experimentos no Fogbed

Uma aplicação de névoa e seus serviços são executados em um ou mais nós virtuais dentro de uma instância virtual. Na arquitetura do Fogbed, existem maneiras diferentes de interagir com o ambiente durante um experimento. A primeira maneira é definindo procedimentos no próprio *script* de topologia. Esses procedimentos serão executados após a inicialização do ambiente de névoa em:

```
exp = FogbedExperiment(topo, switch=OVSSwitch)
exp.start()
```

Onde os procedimentos determinam os passos que o experimento deve realizar:

```
try:
    print exp.get_node("cloud.d1").cmd("ifconfig")
    print exp.get_node(d2).cmd("ifconfig")
    print "aguarde 5 segundos para os algoritmos de roteamento do
           controlador convergirem"
    time.sleep(5)
    print exp.get_node(d1).cmd("ping -c 5 10.0.0.252")
    print exp.get_node("fog.d2").cmd("ping -c 5 10.0.0.251")
finally:
    exp.stop()
```

No exemplo, estamos executando o comando *ifconfig* no nó virtual $d1$ que está sendo executado na instância virtual *cloud*. Em seguida, é executado o mesmo comando dentro do nó virtual $d2$, desta vez usando uma variável de referência em vez passar como parâmetro uma *string* com o identificador do nó virtual. Após 5 segundos de espera, o experimento executa o comando *ping* de $d1$ para $d2$ e vice-versa.

Além de permitir procedimentos no próprio *script* de topologia, as imagens de contêineres instanciados como nós virtuais podem incluir *scripts* pré-configurados de forma que, após a sua inicialização, executem serviços, aplicações e procedimentos sobre o ambiente de emulação. Para facilitar o gerenciamento, uma interface de linha de

comando (*Command Line Interface* - CLI) interativa no Fogbed permite que os desenvolvedores interajam com os nós emulados, alterem configurações, visualizem arquivos de log ou executem comandos arbitrários enquanto a plataforma Fogbed executa seus aplicativos e os seus serviços.

Outra forma de interagir com um experimento é implementando um ou mais processos responsáveis pela inicialização e gerenciamento das instâncias virtuais no ambiente de emulação *Fogbed*. O sistema de gerenciamento precisa interagir com o ambiente emulado para controlar os nós virtuais em instâncias virtuais. Neste caso, a comunicação entre uma instância virtual e o sistema de gerenciamento é realizada através de uma API de instância padronizada e extensível. A API de instância fornece uma semântica de infraestrutura como serviço (IaaS) para gerenciar nós virtuais de maneira adaptável. Ela é projetada como uma interface abstrata para permitir a integração e o teste de sistemas de terceiros. Um desenvolvedor pode implementar sua própria interface de gerenciamento sobre uma API de instância virtual. Uma vez implementada, a interação do sistema de gerenciamento com a instância virtual acontece através de uma porta de comunicação definida no *script* de topologia. O exemplo a seguir adiciona uma API de instância específica a cada tipo de instância virtual:

```
api1 = EdgeApi(port=8001)
api1.connectInstance(e1)
api1.start()
api2 = FogApi(port=8002)
api2.connectInstance(f1)
api2.start()
api3 = CloudApi(port=8003)
api3.connectInstance(c1)
api3.start()
```

A implementação de diferentes APIs de instância permite que cada instância virtual use um processo ou sistema diferente de gerenciamento. Com essa concepção flexível, é possível a execução de diferentes estratégias de gerenciamento para cada instância virtual no ambiente emulado.

6.4.3.4. Emulação Distribuída

O exemplo anterior explorou uma emulação em uma única máquina hospedeira. O Fogbed pode ser adaptado, usando uma interface como a fornecida pelo Maxinet, para executar uma topologia distribuída sobre diferentes máquinas em rede local. O MaxiNet é executado em um conjunto de máquinas físicas chamadas trabalhadoras *(*workers). Cada um desses *workers* executa o Fogbed e apenas emula uma parte de toda a rede virtual. Os nós virtuais e comutadores virtuais em diferentes *workers* são interconectados usando túneis GRE. O MaxiNet fornece uma API centralizada para controlar a emulação, onde esta API é invocada em um *worker* especializado chamado de *frontend*. O *frontend* particiona e distribui a rede virtual para os *workers* e mantém uma lista de qual nó virtual reside em qual *worker*. Desta forma, podemos acessar todos os nós através do *frontend*.

Porém, antes de executar a configuração distribuída, é preciso verificar: (*i*) se todas as máquinas hospedeiras que vão executar o ambiente distribuído possuem o Fogbed instalado; e (*ii*) se as máquinas hospedeiras conseguem se comunicar através da rede local.

Depois é necessário iniciar o controlador POX em uma das máquinas presentes na rede. O controlador POX é um aplicativo em redes definidas por software (*Software-Defined Networking* - SDN) que gerencia o controle de fluxo. Ele é instalado na subpasta *pox* no diretório do Fogbed. No caso da execução do Fogbed como um contêiner Docker privilegiado, o caminho e os comandos para executar o controlador POX são:

```
$ cd /pox
$ ./pox.py forwarding.l2_learning
```

Após executar o controlador SDN em uma maquina na rede, é necessário configurar o Maxinet. Uma máquina da rede local será o *frontend* da emulação distribuída, e as outras máquinas serão os *workers* da rede. No *frontend* da rede copie o conteúdo do arquivo */usr/local/share/maxinet/config.example* para o diretório */etc/MaxiNet.cfg*. No arquivo copiado, substitua o endereço IP do atributo *controlador* pelo endereço IP da máquina que está executando o POX:

```
controller = 172.17.0.2:6633
```

No atributo *ip* do *[FrontendServer]* preencha o endereço da máquina que você pretende executar o servidor *frontend*:

```
[FrontendServer]
ip = 172.17.0.2
```

Abaixo da configuração do *[FrontendServer]*, insira o nome e o endereço IP de cada máquina *worker* da rede. No exemplo a seguir são configuradas duas máquinas:

```
[worker1-hostname]
ip = 172.17.0.2
share = 1
[worker2-hostname]
ip = 172.17.0.3
share = 1
```

Para verificar o nome e o IP de uma máquina, basta executar no terminal:

```
# print hostname
$ hostname

[worker1-hostname]
# print ip
```

Na máquina definida como o *frontend*, execute o comando:

```
$ sudo FogbedFrontendServer
```

E em cada máquina *worker* da rede, execute o comando:

```
$ sudo FogbedWorker
```

É possível verificar o status do cluster de emulação com o comando:

```
$ FogbedStatus
MaxiNet Frontend server running at 172.17.0.2
Number of connected workers: 2
--------------------------------
worker1-hostname            free
worker2-hostname            free
```

Em seguida, é necessário ajustar o *script* de topologia da seção anterior para permitir a execução de maneira distribuída. Para isso, é apenas necessário mudar a classe do experimento no *script* de topologia para *FogbedDistributedExperiment* e salvar o arquivo:

```
exp = FogbedDistributedExperiment(topo, switch=OVSSwitch)
```

Por fim, execute o *script* de topologia na máquina *frontend*:

```
$ python examples/virtual_instance_example.py
```

Após a execução, A partir de uma CLI executada na máquina *frontend* é possível verificar quais nós e comutadores são emulados em qual máquina física. A CLI interativa do *frontend* também ajuda na depuração dos experimentos, sendo possível executar comandos arbitrários e de forma centralizada em qualquer um dos nós virtuais emulados.

6.4.3.5. Coleta de Dados

Assim como em ambientes de névoa IoT do mundo real, a aplicação na coleta de dados no Fogbed depende do experimento executado. Porém, um esquema geral para observar o comportamento do ambiente pode ser implementado através do monitoramento dos fluxos de dados nas interfaces dos nós virtuais e comutadores virtuais. Por exemplo, é possível empregar o NetFlow[B. Claise, Ed. 2004] como tecnologia de monitoramento de fluxo e o NFDUMP[28] como a ferramenta de coleta e análise. Os registros de fluxo são enviados para um *daemon* de captura de fluxo *nfcapd* especificado usando o comando *fprobe* em nós virtuais e o comando *ovs-vsctl* em comutadores virtuais. A ferramenta de análise de fluxo *nfdump* pode ser então usada para estudar o tráfego coletado em cada interface virtual. Neste esquema, os contêineres podem ser configurados para executar as *daemon* de captura e monitoramento durante a sua inicialização.

Em uma emulação distribuída, as funcionalidades do MaxiNet registram e avaliam automaticamente o uso dos recursos físicos ao longo do experimento. Para isso, o MaxiNet monitora a utilização da CPU, o consumo de memória e o uso da rede de todas as máquinas físicas. Após o término de um experimento, esses dados podem ser avaliados para garantir que nenhum recurso físico tenha sido sobrecarregado durante o experimento.

6.4.3.6. Emulação da Plataforma SOFT-IoT

Neste capítulo, abordamos aspectos básicos de como instalar e configurar o *framework* Fogbed para uma emulação em névoa local e distribuída. No entanto, nos exemplos práticos não foi instanciada nenhuma arquitetura IoT, mas apenas um ambiente ou infraestrutura virtual onde essas arquiteturas podem ser projetadas, executadas e testadas. Embora este *framework* tenha como motivação a plataforma SOFT-IoT, seu projeto foi elaborado de forma aberta, genérica e extensível, podendo ser adaptado para testes de outras arquiteturas de névoa e IoT. Como suplemento a este material, em [Fogbed 2018] é apresentado um estudo de caso utilizando imagens pré-configuradas para demonstrar a emulação de componentes e serviços da plataforma SOFT-IoT. Mais informações sobre o Fogbed e seu código fonte podem ser encontrados na página do projeto no GitHub[29].

[28] https://github.com/phaag/nfdump

[29] https://github.com/fogbed/fogbed

6.5. Considerações Finais

Este capítulo apresentou a plataforma SOFT-IoT, que é uma implementação do modelo *Fog of Things*. A SOFT-IoT é uma plataforma de IoT distribuída que usa a infraestrutura do *Fog of Things* para implantar módulos em FoT-Gateways, FoT-Servers e servidores em nuvem. Além disso, o SOFT-IoT é uma plataforma de IoT agnóstica e que fornece uma infraestrutura flexível e configurável.

A SOFT-IoT busca desenvolver avanços tecnológicos que permitirão promover e viabilizar o paradigma *Fog of Things*. Muitas questões desafiadoras ainda precisam ser abordadas para uma ampla utilização desse paradigma, como visto nos tópicos de pesquisa delineados neste capítulo. Esses desafios vão desde soluções que permitam a interoperabilidade e a integração dos diversos componentes e serviços que compõem os ambientes de IoT, passando pelo processamento de uma grande quantidade de dados, à escalabilidade por conta do grande número de objetos (coisas) envolvidos e pela facilitação no desenvolvimento e no teste de aplicações para esses ambientes.

Nesse contexto, as plataformas de *middleware* para IoT precisam considerar diferentes questões e satisfazer a um conjunto de requisitos a fim de cumprir as demandas dos desafios apresentados. Pesquisas adicionais sobre arquiteturas IoT mais escaláveis e que tirem proveito das características do modelo FoT para tratar aspectos como interoperabilidade, descoberta e gerenciamento de dispositivos, interfaces de comunicação, ciência de contexto, escalabilidade, gerenciamento de dados, segurança e adaptação dinâmica podem permitir o uso da arquitetura SOFT-IoT na construção de ambientes de produção de fácil configuração, eficientes, seguros e confiáveis.

References

[Abadi et al. 2018] Abadi, F. A., Ellul, J., and Azzopardi, G. (2018). The blockchain of things, beyond bitcoin: A systematic review. In *2018 IEEE Cybermatics, 2018 IEEE International Conference on*, pages 1666–1672. IEEE.

[Abdelshkour 2015] Abdelshkour, M. (2015). IoT, from cloud to fog computing. `http://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing`. [Online; accessed 26-September-2018].

[Al-Fuqaha et al. 2015] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376.

[Andrade et al. 2017] Andrade, L., Rios, R., Nogueira, T., and Prazeres, C. (2017). Applying classification methods to model standby power consumption in the internet of things. In *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*, pages 537–542.

[Andrade et al. 2018] Andrade, L., Serrano, M., and Prazeres, C. (2018). The data interplay for the fog of things: A transition to edge computing with iot. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE.

[B. Claise, Ed. 2004] B. Claise, Ed. (2004). Cisco systems netflow services export version 9. Disponível em: https://tools.ietf.org/html/rfc3954. Acesso em: 20 set. 2018.

[Bassi et al. 2013] Bassi, A., Bauer, M., Fiedler, M., Kramp, T., van Kranenburg, R., Lange, S., and Meissner, S., editors (2013). *Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model*. Springer-Verlag Berlin Heidelberg, 1 edition.

[Batista et al. 2018a] Batista, E., Andrade, L., Dias, R., Andrade, A., Figueiredo, G., and Prazeres, C. (2018a). Characterization and modeling of iot data traffic in the fog of things paradigm. In *2018 17th IEEE International Symposium on Network Computing and Applications TO APPEAR*.

[Batista et al. 2018b] Batista, E., Figueiredo, G., Peixoto, M., Serrano, M., and Prazeres, C. (2018b). Load balancing in the fog of things platforms through software-defined networking. In *Proceedings of the IEEE International Conference on Computer and Information Technology (CIT)*, pages 1785–1791. IEEE.

[Bonomi et al. 2014] Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. (2014). Fog computing: A platform for internet of things and analytics. In Bessis, N. and Dobre, C., editors, *Big Data and Internet of Things: A Roadmap for Smart Environments*, volume 546 of *Studies in Computational Intelligence*, pages 169–186. Springer International Publishing.

[Bonomi et al. 2012] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA. ACM.

[Christidis and Devetsikiotis 2016] Christidis, K. and Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303.

[Coutinho et al. 2018a] Coutinho, A., Greve, F., Prazeres, C., and Cardoso, J. (2018a). Fogbed: A rapid-prototyping emulation environment for fog computing. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE.

[Coutinho et al. 2018b] Coutinho, A., Rodrigues, H., Greve, F., and Prazeres, C. (2018b). Scalable fogbed for fog computing emulation. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7. IEEE.

[Coutinho et al. 2016] Coutinho, A. A. T. R., Greve, F. G. P., and Carneiro, E. O. (2016). Computação em névoa: conceitos, aplicações e desafios. In *Minicursos - XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 266–315. SBC.

[de Oliveira et al. 2014] de Oliveira, R. L. S., Shinoda, A. A., Schweitzer, C. M., and Prete, L. R. (2014). Using mininet for emulation and prototyping software-defined networks. In *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pages 1–6. IEEE.

[Fogbed 2018] Fogbed (2018). How to setup a soft-iot testbed environment. Disponível em: https://github.com/fogbed/softiot. Último acesso: 20 jul. 2018.

[Greve et al. 2018] Greve, F., Sampaio, L., Abijaude, J., Coutinho, A., Valcy, I., and Queiroz, S. (2018). Blockchain e a revoluçao do consenso sob demanda. In *Minicursos do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, pages 1–52. SBC.

[Khan et al. 2012] Khan, R., Khan, S. U., Zaheer, R., and Khan, S. (2012). Future internet: The internet of things architecture, possible applications and key challenges. In *2012 10th International Conference on Frontiers of Information Technology*, pages 257–260.

[Lewis and Fowler 2014] Lewis, J. and Fowler, M. (2014). Microservices: a definition of this new architectural term. *Mars*.

[Mouradian et al. 2017] Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., and Polakos, P. A. (2017). A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*.

[Pöhls et al. 2014] Pöhls, H. C., Angelakis, V., Suppan, S., Fischer, K., Oikonomou, G., Tragos, E. Z., Rodriguez, R. D., and Mouroutis, T. (2014). Rerum: Building a reliable iot upon privacy-and security-enabled smart objects. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2014 IEEE*, pages 122–127. IEEE.

[Prazeres et al. 2017] Prazeres, C., Barbosa, J., Andrade, L., and Serrano, M. (2017). Design and implementation of a message-service oriented middleware for fog of things platforms. In *Proceedings of the Symposium on Applied Computing*, SAC '17, pages 1814–1819, New York, NY, USA. ACM.

[Prazeres and Serrano 2016] Prazeres, C. and Serrano, M. (2016). Soft-iot: Self-organizing fog of things. In *Advanced Information Networking and Applications Workshops (WAINA), 2016 30th International Conference on*, pages 803–808. IEEE.

[Puschmann et al. 2017] Puschmann, D., Barnaghi, P., and Tafazolli, R. (2017). Adaptive clustering for dynamic iot data streams. *IEEE Internet of Things Journal*, 4(1):64–74.

[Serrano et al. 2015a] Serrano, M., Barnaghi, P., Carrez, F., Cousin, P., Vermesan, O., and Friess, P. (2015a). IoT Semantic Interoperability: Research Challenges, Best Practices, Recommendations and Next Steps. Technical report, IERC: European Research Cluster on the Internet of Things.

[Serrano et al. 2015b] Serrano, M., Quoc, H., Le Phuoc, D., Hauswirth, M., Soldatos, J., Kefalakis, N., Jayaraman, P., and Zaslavsky, A. (2015b). Defining the Stack for Service Delivery Models and Interoperability in the Internet of Things: A Practical Case With OpenIoT-VDK. *IEEE Journal on Selected Areas in Communications*, 33(4):676–689.

[Soldatos 2016] Soldatos, J. (2016). *Building Blocks for IoT Analytics:*. River Publishers Series in Signal, Image and Speech Processing. River Publishers.

[Sousa and Prazeres 2018] Sousa, N. R. and Prazeres, C. V. S. (2018). M2-fot: a proposal for monitoring and management of fog of things platforms. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE.

[Sundmaeker et al. 2010] Sundmaeker, H., Guillemin, P., Friess, P., and Woelffle, S., editors (2010). *Vision and Challenges for Realising the Internet of Things*. European Commission.

[Wette et al. 2014] Wette, P., Draxler, M., Schwabe, A., Wallaschek, F., Zahraee, M. H., and Karl, H. (2014). Maxinet: Distributed emulation of software-defined networks. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE.

[Zhu and Badr 2018] Zhu, X. and Badr, Y. (2018). A survey on blockchain-based identity management systems for the internet of things. In *2018 IEEE Confs on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, Congress on Cybermatics*, number 6, pages 1568–1573.

Biografia Resumida dos Autores

Leandro Andrade, é doutorando em Ciência da Computação pela Universidade Federal da Bahia (UFBA). Obteve o título de Mestre em Ciência da Computação (2014) e também o de Bacharel em Ciência da Computação (2012) pela UFBA. É pesquisador do grupo WISER-UFBA, atuando em projetos relacionados a Internet das Coisas, Computação em Névoa e Big Data. Realizou doutorado sanduíche no The Insight Centre for Data Analytics (NUI Galway - Irlanda). Leandro é membro da Sociedade Brasileira de Computação (SBC), da Communications Society e possui publicações em conferências nacionais e internacionais. É professor substituto da UFBA, vinculado ao Departamento de Ciência da Computação. Possui interesse em pesquisa nas áreas de Internet das Coisas, Computação em Névoa, Serviços Web, Web Semântica, Big Data, Aprendizado de Máquina, Informática na Educação e Software Livre.

Cleber Santana é doutorando em Ciência da Computação pela Universidade Federal da Bahia (UFBA). Obteve o título de Mestre em Sistemas e Computação pela Universidade Salvador (UNIFACS) em 2015. É pesquisador do grupo WISER e NUMAC, atuando em projetos relacionados a Microserviços, Internet das Coisas, Serviços Web Semânticos e Educação. Santana é membro da IEEE Communications Society e possui publicações em conferências nacionais e internacionais. Desde 2013 é professor do Instituto Federal da Bahia e possui interesse em pesquisa nas áreas de Web Semântica, Serviços Web, Microserviços, Internet das Coisas, Computação em Névoa, Inteligência Artificial e Informática na Educação.

Brenno de Mello é mestrando em Ciência da Computação da Universidade Federal da Bahia (UFBA). Atualmente, é participante do grupo de pesquisa WISER, pesquisando principalmente os seguintes temas: Internet das Coisas, Mineração de Fluxo de Dados, Fog Computing, Smart Water. Mello possui graduação em Análise e Desenvolvimento de Sistemas pelo Instituto Federal da Bahia (2016). Tem experiência na área de Ciência da Computação, com ênfase em Sistemas de Informação, atuando como Analista de Sistemas.

Andressa Andrade é graduanda em Engenharia de Computação na Universidade Federal da Bahia (UFBA) e bolsista de iniciação científica pela CNPq. Atuou como monitora na disciplina de Robótica Inteligente na UFBA. Desde 2015, é membro do grupo de pesquisa WISER (Web, Internet and Intelligent Systems Research). Sua área de interesse são tecnologias da camada de percepção, trabalhando principalmente no desenvolvimento de dispositivos físicos (Nó de Sensores/Atuadores) baseados na arquitetura da Internet das Coisas.

Antonio Coutinho é doutorando na UFBA. Ele recebeu o título de Mestre em Informática (2000) e Bacharel em Ciência da Computação (1998) pela Universidade Federal de Campina Grande (UFCG), Paraíba. Desde 2004, é professor assistente no Departamento de Tecnologia (DTEC) da Universidade Estadual de Feira de Santana (UEFS), Brasil. Desde 2016, é membro dos grupos de pesquisa WISER e GAUDI da UFBA. Seus principais interesses de pesquisa envolvem a aplicação da tecnologia blockchain em sistemas de Computação em Névoa e IoT. No SBRC 2016, ele foi autor e ministrou o minicurso "*Computação em Névoa: Conceitos, Aplicações e Desafios*". No SBRC 2018, ele foi autor do minicurso "*Blockchain e a Revolução do Consenso sob Demanda*". Atualmente, participa do projeto FOGGY, cujo objetivo é desenvolver soluções confiáveis para o ambiente de névoa.

Fabíola Greve é pesquisadora e professora Associada do Departamento de Ciência da Computação (DCC) da Universidade Federal da Bahia (UFBA). Ela é membro permanente do Programa de Pós-Graduação em Ciência da Computação (PGCOMP), onde atua como o líder do grupo de computação distribuída GAUDI. Realizou seu doutorado em Ciência da Computação pela Université Rennes I e Laboratórios IRISA-INRIA, França e pós-doutorado no LIP6, Université Pierre et Marie Curie (Paris-Sorbonnes Universités), França. Ao longo de sua carreia científica tem contribuído nas áreas de algoritmos e sistemas distribuídos, tolerância a falhas, desenvolvimento de sistemas confiáveis e blockchain, sendo especialista em problemas de concordância e consenso distribuído, onde tem diversas publicações; particularmente, no SBRC 2005, ministrou o minicurso "*Protocolos Fundamentais para o Desenvolvi-*

mento de Aplicações Robustas". No SBRC 2018, ela é autora e ministrou o minicurso ""*Blockchain e a Revolução do Consenso sob Demanda*". Profa. Fabíola coordena atualmente a Comissão Especial de Redes e Sistemas Distribuídos da SBC (Sociedade Brasileira de Computação), é membro e presidente do Comitê Consultivo da Conferência SBRC e membro do Conselho Administrativo da Rede Nacional de Pesquisa (RNP), representando a SBC. Realizou visitas, como pesquisadora convidada, a laboratórios de pesquisa na França: LIP6 - Laboratoire de Recherche en Informatique de l'Université Paris 6 (2009, 2010); LRI - Laboratoire de Recherche en Informatique, Université Paris-Sud, Orsay (2007); IRISA - INRIA, Université de Rennes I (2004, 2005).

Cássio Prazeres é Doutor em Ciências - Área de Ciências de Computação e Matemática Computacional - pela Universidade de São Paulo (2009), é professor Associado I na Universidade Federal da Bahia (UFBA) nas áreas Internet/Web e é orientador permanente no Programa de Pós-graduação em Ciência da Computação (PGCOMP-UFBA). Prazeres é membro: da Sociedade Brasileira de Computação (SBC); do ACM SIGWEB (Special Interest Group on Hypertext the Web); do IEEE Computer Society Technical Committee on Services Computing; do IEEE Smart Cities Technical Community; do IEEE Internet of Things Technical Community; e do W3C Web of Things Community Group. É co-fundador e líder do Laboratório e Grupo de Pesquisa CNPq WISER (Web, Internet and Intelligent Systems Research Group). Tem interesse em pesquisas envolvendo tópicos de: Internet of Things, Web of Things, Web Services, Semantic Web, Microservices, Fog Computing, Fog of Things, Web of Data. Em 2015, Prazeres realizou estágio pós-doutoral como professor visitante no DERI (Digital Enterprise Research Institute) na National University of Ireland (Galway) nas áreas de Internet das Coisas e Web Semântica.