UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

RUAN ROCHA MARTINELLI

UMA ARQUITETURA ORIENTADA A SERVIÇO DE SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES PARA REDE DE SENSORES SEM FIO

VITÓRIA
2017

RUAN ROCHA MARTINELLI

UMA ARQUITETURA ORIENTADA A SERVIÇO DE SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES PARA REDE DE SENSORES SEM FIO

Monografia apresentada ao curso de Ciência da Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. José Gonçalves Pereira Filho
Coorientador: Prof. M.Sc. Sérgio Teixeira

VITÓRIA
2017

RUAN ROCHA MARTINELLI

UMA ARQUITETURA ORIENTADA A SERVIÇO DE SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES PARA REDE DE SENSORES SEM FIO

COMISSÃO EXAMINADORA

_____

Prof. Dr. José Gonçalves Pereira Filho
Departamento de Informática – UFES
Orientador

_____

Prof. M. Sc. Sergio Teixeira
Departamento de Informática – UFES
Universidade Católica de Brasília - UCB
Centro Universitário Católico de Vitória - UCV
Coorientador

_____

Prof. Ph.D. Eduardo Zambon
Departamento de Informática – UFES

_____

Prof. Dr. Adriano Francisco Branco
Membro externo

Vitória, 21 de março de 2017

RESUMO

ABSTRACT

1 – INTRODUÇÃO

A Internet das coisas ou *Internet of Things* (IoT) traz um novo conceito e uma visão global composta por redes de objetos inteligentes que interoperam através da Internet [1]. Nesse contexto, objetos embarcados com capacidade de processamento, armazenamento, comunicação sem fio e de sensoriamento, possibilitam a criação de aplicações de IoT que podem beneficiar empresas, governos ou qualquer interessado nesse tipo de aplicação. Por exemplo, a cidade de Barcelona na Espanha utiliza dispositivos autônomos para melhorar o sistema de trânsito, estacionamento e o uso eficaz de energia elétrica [2]. Outro exemplo interessante é o Amazon Echo, solução criada pela Amazon com o objetivo de fornecer informações sobre o clima, esportes, dentre outros inúmeros serviços [3].

Os conceitos de IoT não são novidade. No início da década de 90 Weiser vislumbrava uma sociedade que poderia viver de forma harmônica com sistemas computacionais inseridos no nosso cotidiano de maneira ubíqua e quase imperceptível. Essa visão de Weiser pode ser notada nos dias atuais, pois já existem diversas soluções de IoT que fazem uso de objetos inteligentes que facilitam a vida das pessoas. Uma pesquisa realizada pelo *International Data Corporation* (IDC) prevê um crescimento do mercado de IoT na ordem de 1,7 trilhões de dólares até o ano de 2020 [4]. Além disso, estudos de *Business Intelligence* (BI) [5] estimam que nesse mesmo ano teremos mais de 24 bilhões de dispositivos no planeta – aproximadamente 4 dispositivos para cada ser humano no planeta.

Apesar da IoT não ser um conceito novo e da existência de inúmeras aplicações e dispositivos em funcionamento é preciso ter conhecimento especializado de equipes multidisciplinares para desenvolver e colocar em operação as soluções de IoT. O Desenvolvimento de uma aplicação de IoT ainda é uma atividade complexa que exige conhecimento de programação e de aspectos de baixo nível relacionados às plataformas de hardware. Existem diferentes tipos de soluções disponíveis que utilizam diferentes tipos de hardwares e sistemas que são específicos para cada plataforma de hardware [6, 7, 8].

Um importante elemento utilizado no desenvolvimento de soluções de IoT são as Redes de Sensores Sem Fio – do inglês Wireless Sensor Networks (WSN). Nesse tipo de rede estão presentes uma variedade de plataformas que agregam os mais diversos tipos de hardware, sistemas e linguagens de programação. As WSN são capazes de medir grandezas por meio de pequenos nós com capacidades de processamento, armazenamento, comunicação e detecção para permitir, aos seus usuários, interagir e observar em detalhes e com boa capilaridade os mais variados ambientes ou objetos de interesse. O baixo custo de uma WSN cria uma atmosfera fértil e com enorme potencial para ser aplicada no monitoramento e controle das mais diversas áreas. [9, 10, 11, 12, 13]

O uso de aplicações de IoT pode ajudar uma empresa a melhorar seus objetivos estratégicos

ou fazer uso dessas aplicações para evoluir seus modelos de negócios, permitindo rápidas respostas para as questões competitivas. Para isso, é preciso desenvolver mecanismos e sistemas que permitam alterar rapidamente as aplicações de IoT, atendendo as demandas e mudanças que podem ocorrer nos requisitos dessas aplicações e que também podem estar relacionadas às forças competitivas ou mudanças no modelo de negócio das empresas.

No ambiente corporativo as WSN podem ser utilizadas em diversas aplicações que podem estar integradas ao sistema de gestão integrado da empresa – do inglês *Enterprise Resource Planner* (ERP) ou com alguma solução de Gerenciamento de processos de negócios do inglês Business Process Management (BPM). Por exemplo, Meyer [14] propõe um sistema baseado em BPM que permite a precificação dinâmica para reduzir automaticamente o preço de um produto, de acordo com a qualidade estimada. De acordo com o sensor, que pode detectar a temperatura no espaço ao redor das orquídeas, o sistema é capaz de estimar a qualidade e automaticamente reduzir e exibir o preço no painel eletrônico.

Uma aplicação de IoT/WSN normalmente é composta pelo código que é executado nos nós sensores de uma WSN e por algum sistema hospedado em algum computador ou servidor para processar os dados, gerar estatísticas ou informações que sejam úteis aos usuários interessados nos resultados dessa aplicação. Essa aplicação final é o que comumente interessa aos profissionais que trabalham em uma empresa ou qualquer instituição que faça uso de WSN para desenvolver soluções de IoT. Por exemplo, Tranquillini [15] propõe um sistema automatizado de ventilação prédios com eficiência energética, levando em consideração a temperatura, ocupação e níveis de CO_2 do ambiente. A automação é feita com o uso de sensores que monitoram temperatura, umidade, presença e níveis de CO_2 em ambientes fechados para acionar o sistema, permitindo considerável economia de energia. A aplicação final é o sistema que permite coletar esses dados e fornecer informações sistematizadas sobre o consumo de energia dos ambientes.

O modelo e infraestrutura de Tecnologia da Informação e Comunicação (TIC) das empresas na atualidade é heterogêneo e normalmente utiliza algum tipo de infraestrutura privada ou pública de computação em nuvem. Uma pesquisa realizada pelo IDC [16] em 2015 indica que 41% das indústrias dos Estados unidos preferem utilizar serviços via nuvem pública e quase 50% das indústrias da Europa utilizam seu sistema ERP hospedado em nuvem pública. Uma forma utilizada para permitir a integração entre a infraestrutura de TIC da empresa e os sistemas hospedados em nuvem é por meio do uso de arquiteturas e conceitos de orientação a serviço – do inglês *Service-Oriented Architecture (SOA)*. SOA é uma abordagem de computação distribuída que utiliza protocolos e padrões abertos para facilitar a interoperabilidade entre sistemas heterogêneos independente de plataforma [17, 18].

No ambiente corporativo as aplicações finais são as mais diversas, pois o ambiente é heterogêneo, entretanto, normalmente seguem padrões consagrados no mercado como o REST,

Publisher/Subscriver (Pub/Sub), dentre outros. Os profissionais que já atuam nas empresas comumente preferem integrar as WSN às aplicações já existentes na empresa. Dessa forma, não existe interesse ou tempo disponível para aprender novos conceitos referentes aos aspectos de baixo nível das WSN. O interesse normalmente é focado no dado que foi sensoreado e não nos detalhes de plataforma de hardware ou sistemas utilizados para obter uma medida de algum tipo de sensor. Capacitar cada profissional que desenvolve as aplicações finais de WSN pode ser custoso e demorado. Uma pesquisa realizada pela Cisco [19] estima um déficit de profissionais de TIC para trabalhar com aplicações de IoT de 8.2% em 2022 e evidenciam a necessidade de capacitação em IoT.

Diante disso, ganham destaque as soluções que visam flexibilizar ou facilitar a integração entre WSN e as aplicações finais típicas do ambiente corporativo de tal forma que, os profissionais responsáveis pelo desenvolvimento das soluções finais de IoT/WSN não tenham que conhecer os detalhes de baixo nível das WSN, evitando assim a necessidade de esforço cognitivo adicional ou de capacitação nessas novas tecnologias. Dessa forma, os desenvolvedores da empresa terão a possibilidade de utilizar interfaces padronizadas de comunicação como RESTful, *Publisher/Subscriber, Websocket,* dentre outras, para integrar as aplicações já existentes no ambiente corporativo com as WSN.

Diante desse contexto, este trabalho propõe uma solução para a integração de aplicações finais com uma ou mais WSN, por meio de interfaces de comunicação padronizadas, permitindo que os desenvolvedores das aplicações finais não precisem ter qualquer tipo de preocupação com aspectos de programação, sistemas ou tecnologias de baixo nível referentes ao funcionamento da WSN. Dessa forma, o desenvolvedor precisa apenas ser informado sobre as API disponíveis para que a aplicação final possa acessar os dados sensoreados da WSN.

Este trabalho foi motivado e desenvolvido com base em um projeto de doutorado em andamento proposto por Sergio Teixeira, que faz parte do Laboratório de Pesquisas em Redes e Multimídia (LPRM) do departamento de informática da Universidade Federal do Espírito Santo (UFES), que propõe uma arquitetura de suporte ao desenvolvimento de soluções para WSN cujo nome proposto inicialmente é **L**ean **AU**tomatic code generation for situation-awa**R**e **A**pplication (LAURA). O Projeto LAURA prevê uma arquitetura de suporte e integração entre as WSN e as aplicações finais típicas do contexto empresarial, incluindo aplicações situation-Aware que podem ser modeladas com o uso de sistemas BPM e uma arquitetura de geração de códigos dos nós das WSN. O serviço proposto nesse projeto de graduação visa implementar aplicações previstas na arquitetura LAURA, principalmente no desenvolvimento dos serviços Terra Gateway e Terra Core que serão detalhados mais adiante.

1.1 OBJETIVOS

O objetivo geral deste trabalho é propor uma arquitetura com aplicações e serviços para facilitar a integração e o desenvolvimento de aplicações finais para WSN desacopladas dos detalhes de mais baixo nível da WSN. Além disso, como objetivos específicos esse trabalho também irá:

- Implementar a aplicação *Terra Gateway* cujo objetivo é receber e encaminhar dados originários da WSN baseada em *Terra System* para a aplicação *Terra Core*, possibilitando o total desacoplamento das questões de mais baixo nível referentes a WSN, em relação às interfaces de comunicação que serão disponibilizadas para as aplicações finais;
- Implementar a aplicação *Terra Core* com o objetivo de receber os dados do Terra Gateway e disponibilizar interfaces de comunicação padronizadas para as aplicações finais interessadas nos dados sensoreados. O Terra Core executará funções complementares de suporte ao que foi sensoreado na WSN com o objetivo de reduzir o processamento do nó, fará a persistência dos dados em banco de dados relacional e disponibilizará interfaces API REST, Publisher/Subscribe e WebSocket para que as aplicações finais possam acessar dados da WSN com total desacoplamento dos detalhes de mais baixo nível da WSN em relação às questões de mais alto nível que são de interesse das aplicações finais.
- Implementar a aplicação *Terra Web Control* cujo objetivo é auxiliar na manutenção e gerência da WSN, identificando e monitorando os dados recebidos dos nós da WSN;

1.2 METODOLOGIA

O primeiro passo para o desenvolvimento do trabalho foi o estudo de propostas de integração entre redes de sensores sem fio e aplicações finais interessadas em dados sensoriados. Foi realizada pesquisa na literatura com o objetivo de identificar aplicações que apresentam interesse na leitura de dados sensoriados.

O estudo teve como base a busca por soluções que utilizassem a *internet* como principal forma de integração entre redes e aplicações finais. Por conta da distribuição física e até mesmo atemporal dos componentes em um cenário de redes de sensores sem fio, a pesquisa se concentrou em soluções mais flexíveis, desacopladas e com interfaces padronizadas. Ao longo da revisão da literatura, constatou-se que abordagens orientadas a serviço do tipo *Web Service* representam o estado da arte no assunto, direcionando ainda mais a pesquisa para esse tipo de arquitetura.

A fim de propor uma abordagem que levasse em conta a heterogeneidade de redes atuais, a modelagem da arquitetura baseou-se em redes que fizessem o uso de máquinas virtuais na programação de sensores. O trabalho de Branco [20] foi usado como referência para esses tipos de rede, cujo sistema Terra System, baseado no TinyOS é utilizado na configuração e atualização dinâmica dos nós da WSN.

Após definida uma arquitetura conceitual da solução, baseando-se no estudo de diferentes padrões de comunicação e protocolos de rede, foram encontradas ferramentas que auxiliassem a implementação da solução proposta. Na implementação do *Web Service* proposto, foi feito o uso da linguagem JavaScript com o auxílio do *framework* Node.js [21], voltado para a construção de *Application Program Interfaces* (APIs) e capaz de lidar com um grande número de requisições simultâneas. Algumas das razões para a escolha de JavaScript como linguagem foram a sua natureza assíncrona padrão, sem a necessidade de criar e gerenciar threads, a crescente popularidade [22], e a existência de bibliotecas que facilitam a comunicação em tempo real como a Socket.io [23] e ZeroMQ [24].

Para a comunicação com a rede Terra System utilizou-se a biblioteca TOSSAM [25]. Através desta é facilitada a troca de mensagens com sensores que utilizem o mesmo sistema adotado pelo Terra System.

1.3 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está estruturado como segue:

- O capítulo 2 apresenta os principais conceitos referentes a aplicações orientadas a serviço, padrões de arquiteturas e protocolos de comunicação que baseiam a arquitetura proposta;
- O capítulo 3 apresenta a arquitetura conceitual proposta, uma sugestão de sua implementação e um exemplo de aplicação-cliente, fazendo o uso da solução desenvolvida;
- O capítulo 4 apresenta um cenário real de aplicação e o resultado dos testes realizados;
- O capítulo 5 traz conclusões e sugestões de trabalhos futuros que podem dar prosseguimento ao trabalho proposto.

2 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo apresenta os principais conceitos relacionados aos tópicos de pesquisa relacionados ao trabalho proposto.

2.1 REDES DE SENSORES SEM FIO

Uma WSN é composta por um conjunto de nós capazes de monitorar cooperativamente uma região de interesse. Cada nó é composto de um transmissor de rádio, um processador e uma fonte de energia e pode estar acoplado a diversos tipos de sensores capazes de monitorar medidas de temperatura, pressão, movimento, som, dentre outros inúmeros tipos de sensores [27].

O dado sensoreado em uma WSN é repassado de nó em nó até alcançar o nó sorvedouro também conhecido como Sink Node que é responsável por estabelecer a comunicação dos dados da WSN com um computador ou servidor externo. Os nós de uma WSN são dispositivos de baixo custo com recursos de processamento e autonomia de energia limitados.

Por se tratarem de dispositivos de baixo custo, os recursos de processamento, armazenamento e autonomia de energia em cada nó sensor são bastante limitados. Além disso, devido ao grande número de sensores presentes em uma rede, é comum que os dispositivos sejam instalados em áreas remotas e de difícil acesso. A atualização manual do código do nó sensor é uma tarefa custosa, pois exige a retirada do nó do seu local de origem para a atualização manual do código. Quando a WSN tem muitos nós ou quando está instalado em locais de difícil acesso o ideal é utilizar alternativas de atualização dinâmica dos códigos dos nós por meio dos links de rádio [20, 28]. Dessa forma, o uso eficaz da energia e processamento são aspectos relevantes em qualquer solução de IoT/WSN [26].

Existem várias arquiteturas de WSN. O TinyOS é o sistema mais utilizado e pode ser utilizado em várias plataformas de hardware. Após o processo de compilação no TinyOS é gerado um único bloco monolítico que é transferido para o nó [29]. Outro tipo de arquitetura que vem atraindo cada vez mais o interesse dos pesquisadores é o uso de WSN cujos nós se baseiam em máquina virtual - do Inglês *Virtual Machine* (VM). A vantagem em relação código monolítico é percebida principalmente no processo de atualização dinâmica. Após as etapas de compilação o código de mais alto nível é utilizado como instruções da VM ou bytecode. No processo de atualização dinâmica basta transmitir o bytecode que pode ser dividido em blocos. Dessa forma, não sobrecarrega a rede no momento da atualização. Uma abordagem que merece destaque no uso de VM que foi adaptada com base no TinyOS é o Terra System proposto por Branco [20]. A programação do nó em Terra System é mais simples e se baseia em linguagem de programação

reativa. A VM é baseada em componentes que permite diferentes customizações. A programação em alto nível se transforma em um bytecode de tamanho reduzido que facilita muito o processo de disseminação. Devido aos recursos e facilidades oferecidas pelo Terra System foi adotado como WSN para o estudo de caso para demonstrar a arquitetura e aplicações propostas nesse trabalho. Apesar da possibilidade de uso de qualquer outro tipo de WSN desde que sejam feitas as devidas adaptações no componente que integra a WSN às aplicações propostas o Terra System foi escolhido pelas suas características, facilidades de atualização dinâmica e principalmente por ser a WSN adotada no estudo de caso proposto no trabalho em desenvolvimento da arquitetura LAURA.

2.2 ARQUITETURAS ORIENTADAS A SERVIÇO

No contexto de WSN, é prevista uma classe de redes em que diversas aplicações necessitem consultar dados de nós *sink* localizados de forma arbitrária em um determinado cenário [30]. Com o aumento da base de aplicações consumidoras, também cresce a demanda por implementações mais complexas de mecanismos de segurança, disponibilidade, privacidade, tolerância a falhas, entre outros requisitos.

Abordagens que propõem resolver essas questões fazem o uso de sistemas intermediários entre a WSN e uma aplicação final consumidora. Nesse tipo de sistema, geralmente são agregados processamentos de dados especializados, módulos de autenticação e gestão de usuários, entre outros mecanismos que contribuam para a interoperabilidade da rede, com aplicações que busquem qualquer tipo de integração com a WSN.

Um dos padrões estruturais mais utilizados no desenvolvimento de sistemas com essas características é o SOA. Em aplicações desse tipo, funcionalidades são expostas a outros componentes ou sistemas em forma de serviços, que podem ser invocados ou acessados remotamente, por meio de um protocolo de comunicação de redes. Dessa maneira, abordagens no contexto de WSN, desenvolvidas de acordo com princípios de SOA, conseguem estender as funcionalidades de uma rede sem aumentar sua complexidade. Por um sistema orientado a serviços com acesso direto aos dados da rede, é possível abstrair as regras de negócio a essa camada, tornando o sistema como um todo mais flexível e desacoplado.

Implementações modernas de sistemas SOA fazem o uso de uma gama de padrões e paradigmas para atingir o objetivo de um sistema orientado a serviço. Atualmente, é comum encontrar implementações da arquitetura SOA, fazendo o uso de *Web Services* para prover uma interface via *internet*.

Tipicamente em sistemas do tipo *Web Service,* é provida uma interação através de protocolos de rede com o HTTP ou HTTPS. Como essa especificação independe da linguagem de programação

entre as duas entidades que se comunicam, a troca de mensagens é feita por arquivos em um formato predefinido, como o XML ou o JSON.

Web Services podem ser implementados por meio de diversos padrões e arquiteturas. No contexto de WSN, abordagens atuais se baseiam nas especificações SOAP e REST para prover uma interface de comunicação de alto nível entre a rede e clientes interessados [31,32]. Na seção a seguir, será apresentado, em mais detalhes, o padrão REST para a construção de *Web Services*.

2.3 PADRÃO REST EM WEB SERVICES

Proposto no ano 2000 por Fielding [33], o padrão REST especifica uma série de requisitos e princípios para a construção de sistemas interoperáveis. Um subconjunto desses sistemas é chamado de *RESTful Web Services* – ou *RESTful APIs* – quando aplicado ao contexto de sistemas *Web* orientados a serviço.

Arquiteturas que obedecem a esse padrão disponibilizam recursos passíveis de serem consumidos por um cliente, numa comunicação típica cliente-servidor. Em *Web Services RESTful*, cada recurso é identificado por uma URI única, permitindo que cada aplicação evolua de maneira independente. As requisições baseiam-se no protocolo HTTP ou HTTPS e a troca de mensagens é feita de maneira similar a outros tipos de *Web Service*, utilizando os formatos XML, JSON e, em alguns casos, HTML.

Requisições a *APIs RESTful* são realizadas por métodos HTTP como, por exemplo, GET, POST, PUT e DELETE. Cada verbo especifica uma ação a ser tomada perante um recurso. Cada requisição é independente e nenhum estado que caracterize requisições anteriores é armazenado entre diferentes as trocas de mensagens.

Embora muito utilizado na integração de sistemas e na comunicação de navegadores com servidores, o padrão REST tem se mostrado um bom candidato para a construção de APIs que forneçam dados de WSN, devido a sua leveza e desacoplamento [31]. Dessa maneira, é possível fornecer interfaces de comunicação para as mais diversas plataformas, sem demandar um processamento intenso da rede.

Apesar das características distribuídas de serviços *RESTful*, o padrão pode não ser adequado em alguns casos. Em WSN, no mundo real, são geradas dezenas e até centenas de mensagens em curtos intervalos de tempo. Em situações como essa, a requisição de cada mensagem pelo protocolo HTTP pode se tornar inviável. Na maior parte dos casos, o tamanho de cabeçalhos da requisição ultrapassa o tamanho do próprio conteúdo requerido. Também, da forma como é proposto, o padrão REST define o servidor como uma entidade passiva, que apenas responde a requisições. Na seção

seguinte, será abordada uma nova demanda de aplicações no âmbito de WSN e como diferentes tipos de *Web Services* podem atender a essa necessidade.

2.4 WSN E A COMUNICAÇÃO EM TEMPO REAL

Nos dias de hoje, é comum encontrarmos diversas aplicações de monitoramento em tempo real. Presentes em hospitais, meios de transporte, eventos, entre outras áreas, é comum nos depararmos com sistemas capazes de receber dados à medida que eles são gerados e, com base nisso, tomar decisões e invocar procedimentos.

Um subconjunto considerável na área de WSN se dedica ao monitoramento e observação de fenômenos. Como exemplo, podemos citar medicamentos que necessitam de refrigeração a uma temperatura constante, ou batimentos cardíacos de um paciente que precisam ser monitorados a cada instante. Para esses casos, mostra-se necessária a existência de um padrão de comunicação eficiente, capaz de trocar mensagens de maneira instantânea.

Em *Web Services RESTful*, a necessidade do envio de cabeçalhos em cada requisição somada ao *overhead* de *handshakes* do protocolo HTTP fazem com que a comunicação da maneira *Request-Response* não seja a mais adequada para o recebimento de dados em tempo real. Nesse tipo de serviço, o servidor exerce um papel passivo, apenas responde a requisições à medida que chegam.

Mecanismos de *polling* podem ser utilizados para simular uma comunicação instantânea em serviços desse tipo, porém a troca de mensagens se mantém custosa e atrasos podem ocorrer na comunicação. Para resolver esse problema, diferentes padrões de troca de mensagem específicos podem ser empregados na comunicação cliente-servidor.

2.5 PADRÃO *PUBLISHER/SUBSCRIBER*

Além do REST, diferentes padrões de troca de mensagens podem ser empregados para atingir objetivos específicos e adaptar sistemas a diferentes casos de uso. Enquanto os serviços *RESTful* focam o desacoplamento de componentes e o reúso, outros padrões abrem mão desse tipo de requisito para obter uma comunicação mais rápida e direcionada.

Um dos padrões de troca de mensagem mais conhecidos nesse segmento é o *Publish/Subscriber*, uma abordagem flexível, em que o servidor é responsável pelo controle de novos conteúdos.

No *Publish/Subscriber,* o cliente interessado assume o papel de "*Subscriber*", declarando interesse por um tópico específico sobre o qual deseja ser comunicado a respeito de atualizações. Os canais disponíveis são gerenciados pelo *Publisher*, representado pelo servidor. Cada nova

mensagem é direcionada a seu canal respectivo e enviada aos *Subscribers* conectados naquele momento.

Trazendo ao contexto de WSN, é possível atender, por meio desse padrão, a demandas de um cliente interessado em dados em tempo real. Cada novo dado sensoriado em uma rede percorre canais exclusivos de comunicação, até chegar a um ou mais clientes interessados.

Em uma comunicação *Publisher/Subscriber*, também é possível atender a demandas diferentes de maneira eficiente. Por exemplo, em uma rede de capacidade variada de sensoriamento, com sensores de pressão, temperatura, presença, etc., é possível que clientes se inscrevam apenas em tópicos de interesse. Em casos como esse, o servidor é responsável por filtrar as mensagens e enviá-las aos *Subscribers* inscritos em cada tópico.

Por ser um padrão abrangente, o *Publisher/Subscriber* pode ser implementado de diversas formas e utilizar diferentes protocolos de comunicação, variando conforme o ambiente onde executa e o tipo de aplicações que integra. Comunicações via *internet,* por exemplo, podem utilizar protocolos distintos para o envio de mensagens. Implementações do padrão Pub/Sub para comunicação via *internet* utilizam tanto de protocolos padronizados, como por exemplo o WebSocket, quanto através de protocolos implementados por terceiros.

No trabalho é possível encontrar referências ao padrão Pub/Sub de maneira mais genérica, sem definir tecnologias ou bibliotecas específicas para estabelcer esse tipo de comunicação. Nesses casos, fica livre a implementação do padrão através de tecnologias que mantenham os requisitos de interoperabilidade e facilidade de uso. Na proposta de implementação foi-se usado a biblioteca ZeroMQ, que além de atender tais requisitos possui implementação em diversas linguagens de programação.

Em outras partes do trabalho, a comunicação por meio do padrão Pub/Sub aparece de forma mais específica, sendo feita através do protocolo de comunicação WebSocket. Padronizado pela W3C, este protocolo vem sendo desenvolvido como forma de comunicação entre servidores e navegadores Web. A escolha do protocolo WebSocket é baseada em uma crescente classe de aplicações desenvolvidas com foco em plataformas Web. Para a implementação de uma comunicação Pub/Sub entre aplicações consumidoras – nesse caso, especificamente, navegadores Web – e servidores utilizou-se a biblioteca Socket.io, que fornece suporte na comunicação por WebSocket e utiliza *fallbacks* em navegadores mais antigos onde o protocolo não é implementado.

3 ARQUITETURA E APLICAÇÕES PROPOSTAS

Um dos grandes desafios associados à construção de aplicações para WSN é a diversidade e a heterogeneidade de tecnologias, protocolos, linguagens e padrões disponíveis [34, 35, 36, 37]. O ambiente e infraestrutura de Tecnologia da Informação e Comunicação (TIC) das empresas é heterogêneo e convive com constantes mudanças no modelo ou regras de negócios que forçam a existência de um ambiente com recursos capazes de reagir rapidamente a essas mudanças. [38, 39, 40, 41]. Entretanto, a diversidade e a heterogeneidade adicional que o desenvolvimento e a integração de aplicações de WSN trazem para o ambiente empresarial, torna o cenário ainda mais complexo e adiciona esforço cognitivo e a necessidade de capacitação da equipe de profissionais responsável pelo desenvolvimento dessas aplicações. Um dos aspectos que normalmente preocupa a equipe de desenvolvedores é a escolha da plataforma de *hardware* de WSN, tecnologias, linguagens e sistemas que fazem parte de uma WSN. Essas escolhas irão influenciar diretamente na forma como a solução de WSN pode se integrar aos sistemas empresariais já existentes e ao modelo de negócio da empresa.

Normalmente, uma solução de desenvolvimento de aplicações para WSN impõe o uso de determinadas tecnologias, sistemas e linguagens para a integração aos sistemas corporativos. Por exemplo, Corredor [37] apresenta uma arquitetura orientada a modelos baseada em ontologias com foco na abstração e reuso de componentes, propondo a interoperabilidade através de WebServices RESTful. Apesar de ter uma boa modularidade e abstrair ao máximo a complexidade da rede, abordagens como essa ainda não complexas para os profissionais que não possuem conhecimento mais profundo em WSN. Em contraste a abordagem de Corredor, o trabalho atual se preocupa em remover essa barreira e permitir que o conhecimento já presente nos profissionais atuais seja aproveitado na hora de integrar suas aplicações com uma WSN. Dessa forma, o consumo de dados de uma rede de sensores se torna tão simples quanto a integração com outras APIs presentes na *Web*.

A escolha de uma solução de WSN deve permitir flexibilidade para o desenvolvimento da aplicação final. É interessante para as empresas que a solução tenha a possibilidade de integração aos sistemas ERP, sistemas legados e soluções orientadas a serviços.

A arquitetura e as aplicações propostas nesse trabalho buscam flexibilizar e permitir o total desacoplamento no desenvolvimento de aplicações finais, eliminando as preocupações de baixo nível que devem ser de responsabilidade do especialista em WSN. Dessa forma, o analista responsável pelo desenvolvimento da aplicação final deve ter total flexibilidade para integrar sua aplicação orientada a serviço a WSN, sem qualquer preocupação sobre a plataforma e os detalhes de mais baixo nível.

A subseção seguinte apresenta uma visão geral da arquitetura e das aplicações propostas, caracterizando, de forma resumida, o funcionamento geral da arquitetura e das aplicações que fazem parte da arquitetura, incluindo as principais funcionalidades e possibilidades. As subseções seguintes irão detalhar todo o funcionamento da arquitetura, indicando o fluxo de interoperabilidade dos dados, desde a leitura do dado no nó sensor até a forma como os dados podem ser consumidos ou acessados pelas aplicações finais.

3.1 VISÃO GERAL DA ARQUITETURA E APLICAÇÕES PROPOSTAS

Antes da apresentação da solução proposta é importante fazer algumas considerações e esclarecimentos, pois a solução apresentada tem o objetivo principal de fornecer suporte ao desenvolvimento de aplicações finais orientadas a serviço. Apesar da existência de uma aplicação web intitulada *Terra Web* que será detalhada mais adiante, cujo objetivo principal é fornecer mecanismos de monitoramento dos dados que estão sendo recebidos e transmitidos para a WSN, o foco principal da solução é disponibilizar um ambiente de suporte ao desenvolvimento de aplicações finais para WSN, em um contexto corporativo, com o uso de WSN orientadas a VM. No estudo de caso utilizado nesse trabalho e como forma de servir como prova de conceito da solução proposta, será utilizada uma WSN baseada no Terra *System* proposto por Branco [20]. Portanto, a solução parte do pressuposto que existe uma rede WSN baseada no Terra *System* em funcionamento, conforme o cenário de aplicação que foi utilizado e será apresentado após a apresentação da arquitetura proposta.

A arquitetura apresentada é composta por três elementos principais: a WSN, o Gateway que será responsável pela interoperabilidade com o *Core* que é o elemento central e mais importante da arquitetura. O *Core* é o responsável pela persistência dos dados originários da WSN, executa funções complementares e necessárias às tarefas realizadas nos nós. Por exemplo, se a temperatura é uma medida analógica ou digital, a conversão dessa medida será feita por essa função, diminuindo o processamento no nó. Além disso, o *Core* disponibiliza as interfaces, por meio de WS, que permitem o acesso padronizado e em tempo real dos dados originários da WSN por meio de API REST, Publisher/Subscriber e WebSocket.

A figura 1 apresenta a arquitetura e as aplicações propostas. A WSN pode ser caracterizada por qualquer tipo de plataforma de *hardware* de WSN, desde que exista um G*ateway* capaz de interoperar com a WSN, estabelecendo uma comunicação padronizada com o *Core*. O primeiro contato da arquitetura proposta com a WSN é feito por meio do *Gateway* que estabelece um canal de comunicação com o *Sink node*. Para executar as funções de *Gateway* e estabelecer comunicação com a WSN, por meio do Sink Node, foi desenvolvido o Terra *Gateway*, aplicação baseada na

linguagem Lua, cujo objetivo é interoperar com uma WSN baseada no Terra *System* através de um canal de comunicação *TOSSAM*. Toda comunicação com a WSN é feita por meio do Terra Gateway que mantém um canal de comunicação *Pub/Sub com o Sink Node*, permitindo interoperabilidade em tempo real WSN com o *Terra Core*. O *Terra Core* é o responsável pelo tratamento do dado, pela persistência em banco de dados relacional e executa os serviços que irão disponibilizar o consumo ou acesso dos dados da WSN via canais de comunicação pelas aplicações finais.

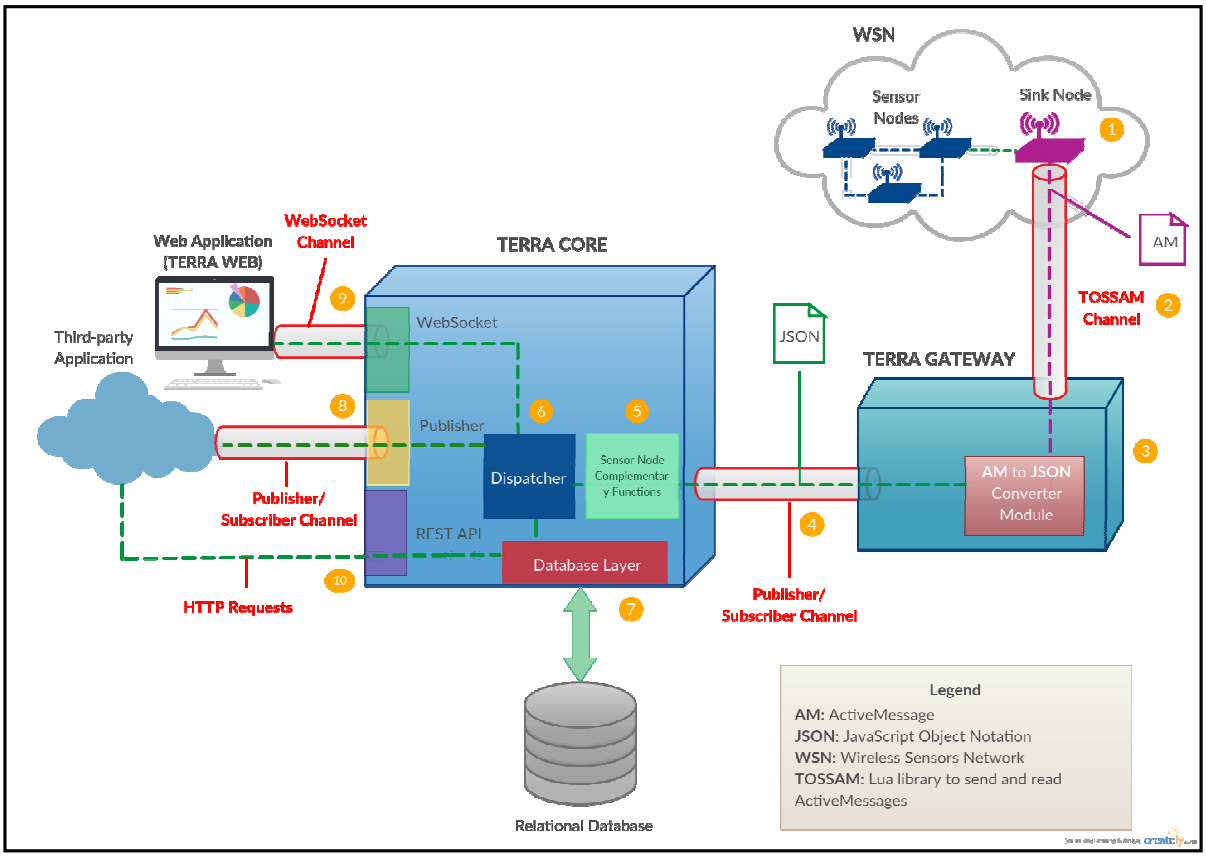

Figura 1. Visão geral da arquitetura e aplicações propostas

As subseções seguintes irão apresentar, em detalhes, a arquitetura e as aplicações propostas, de acordo com a numeração representada pelas bolinhas de cor laranja, indicando o fluxo do dado e o funcionamento da arquitetura e das aplicações.

3.2 WSN E A COMUNICAÇÃO DO *SINK NODE* - ITEM 1

Partindo da premisssa de uma rede pré-configurada e executando sobre a plataforma Terra System, os dados do fenômeno de interesse são sensoriados pelos nós sensores e enviados periodicamente ao nó sink. Por apresentar capacidades e recursos melhores do que os outros nós da rede, o nó sink fica responsável por empacotar e enviar as mensagens provenientes dos nós sensores. Cada mensagem trafega do nó sink ao *Terra Gateway* (item 3) através de um canal de comunicação dedicado, que faz o envio e recebimento das mensagens por meio de sockets TCP.

3.3 – TOSSAM CHANNEL POR MEIO DE MENSAGEM AM - ITEM 2

A comunicação entre a WSN e o *Gateway* deve ocorrer de forma eficaz, leve, padronizada e em tempo real. Para esses fins, sistemas TinyOS disponibilizam o protocolo AM, próprio para a comunicação entre WSNs. Como as redes Terra são baseadas em TinyOS, a opção de usar o protocolo AM para a troca de mensagens se torna uma consequência natural.

Além do protocolo AM ser utilizado internamente na rede, para a comunicação dos nós, ele também é usado para a comunicação com entidades externas à rede, como servidores ou aplicações. Aplicações que se conectem a rede podem se comunicar com a mesma através de interfaces como USB ou Ethernet, por onde passam a receber as mensagens via AM.

O TOSSAM é uma biblioteca baseada na linguagem Lua que dá suporte a diferentes tipos de comunicações que ocorrem através do protocolo AM. Desta forma, optou-se pelo uso da biblioteca devido a facilidade proporcionada na implementação da comunicação entre a rede Terra e o *Gateway.*

3.4 – TERRA *GATEWAY* - ITEM 3

Toda interoperabilidade entre a WSN e o Terra Core é feita pelo Terra Gateway que recebe e envia dados para a WSN, por meio de um módulo que utiliza um canal de comunicação TOSSAM para interoperar com o Sink Node. Além disso, se estabelece um canal de comunicação permanente do tipo Publisher/Subscriber com o Terra Core, realizando o envio e recebimento de mensagens no padrão JSON. O Terra Gateway apresenta também um módulo de conversão de mensagens AM para JSON de maneira a facilitar o processamento e manipulação das mensagens que serão processadas pelo Terra Core.

É comum que a leitura dos nós sensores apresentem medidas digitais ou analógicas, de baixa compreensão para pessoas comuns. O processo de conversão dessas medidas envolve cálculos mais complexos, envolvendo em alguns casos operações logarítmicas e exponenciais. Por conta da baixa capacidade de processamento dos nós sensores fica inviabilizado fazer a conversão no momento da leitura do dado, sendo melhor repassá-lo para o Gateway da maneira que foi lido pelo nó.

Como o papel do Gateway é o de servir como uma interface entre a rede e o Terra Core evitando sempre que possível a aplicação de alguma lógica ou processamento, optou-se por trabalhar com o dado em sua forma pura nessa etapa. Portanto, além da conversão para JSON, o Terra *Gateway* não faz nenhum tipo de tratamento ou modificação nos dados recebidos. O principal propósito é de converter as mensagens AM em um padrão, que no caso foi adotado o JSON, que possa disponibilizar os dados para serem tratados pelo Terra Core de uma forma estruturada e de fácil compreensão.

O Terra *Gateway* aparece como uma forma de manter alto o grau de desacoplamento e flexibilidade na arquitetura. Através de uma camada intermediária responsável pela comunicação da rede com o Terra *Core* é possível atender futuramente diferentes redes apenas conectando serviços, ao invés de realizar manutenções diretamente no Terra *Core*.

3.5 – *PUB/SUB CHANNEL* COM MENSAGENS JSON - ITEM 4

A comunicação entre o Terra *Gateway* e o Terra *Core* é feita por meio de um canal de comunicação direto e permanente do tipo *Publisher/Subscriber.* Nessa comunicação o Terra Core assume o papel de *Subscriber,* indicando que deseja receber em tempo real, na medida em que chegam, toda mensagem disponível no Terra Gateway que, nessa arquitetura, assume o papel de *Publisher.* Dessa forma, toda mensagem é recebida automaticamente sem a necessidade de trocas de mensagens para identificar se existe ou não o interesse em receber mensagens.

Após a inicialização do Terra Core é feita a inscrição em um tópico específico de notificações, indicando quais mensagens deseja receber do *Publisher* (Terra *Gateway*). Logo após o recebimento de uma mensagem da rede, o Terra Gateway as publica no canal de comunicação em que o Subscriber (Terra Core) é inscrito. Dessa forma, as mensagens da rede são repassadas imediatamente (em tempo real) após o seu recebimento.

O modulo de conversão de mensagens AM em objetos JSON segue os seguintes passos: 1) A mensagem AM é convertida para uma tabela em Lua. 2) A Tabela em Lua é transformada em um objeto JSON cujos dados são armazenados na forma de atributos e valores. A figura W apresenta uma visão geral do objeto JSON que foi gerado após o processo de conversão apresentado.


```
{
    gateway_time: '2016-11-23 09:34:11'
    port: 9004,
    id_mote: 11,
    value: -4
}
```


Figura 2. Exemplo do objeto no padrão JSON recebido e convertido pelo Terra Gateway

No estudo de caso proposto foi utilizada a biblioteca ZeroMQ para implementar o canal de comunicação Pub/Sub conforme indica o item 4. Essa implementação é feita em ambas as aplicações.

3.6 – TERRA *CORE – COMPLEMENTARY FUNCTIONS* - ITEM 5

Após receber os objetos JSON originários do Terra Gateway a primeira etapa é executar algumas funções complementares (item 5) e necessárias para que o dado sensoreado na WSN possa ser trabalhado e encaminhado de forma útil para as aplicações finais. Cada mensagem é recebida como uma string de caracteres que é, em seguida, convertida para um objeto JSON nativo da linguagem JavaScript. A primeira operação a ser feita é verificar se existe uma função pré-definida e vinculada à mensagem recebida que precisa ser executada. Por exemplo, quando um dado de medida de temperatura é recebido ele normalmente se encontra no formato nativo que foi lido no nó sensor, representado geralmente por um valor digital ou analógico. Esse dado não foi convertido para uma medida em graus célsius ou Fahrenheit no próprio nó sensor propositalmente, pois o objetivo é economizar energia.

Do ponto de vista de um sensor individual a economia pode ser pouca, porém a preocupação com energia cresce quando escalamos esse gasto para uma rede com dezenas ou centenas de sensores. Dessa forma, existem funções previamente cadastradas e vinculadas aos IDs dos nós que são cadastradas de acordo com o tipo de função que deve ser executada para fazer a conversão de medida conforme o tipo de sensor que está presente no nó.

Após identificar a necessidade de conversão de medida do dado recebido é executada a respectiva função de conversão para transformá-lo na medida que normalmente é utilizada pelas aplicações finais. Por exemplo, transformas o dado digital em uma medida em Célsius. Feito isso o objeto JSON é atualizado com o novo valor de medida de temperatura.

Além da economia de energia a aplicação de funções complementares sobre os dados abre portas para um processamento de dados da rede mais complexo e especializado. Através desse mecanismo fica mais evidente a separação de preocupações e permite que além de funções de conversão, outras técnicas sejam aplicadas sobre dados mais específicos.

3.7 – TERRA *CORE – DISPATCHER* E *DATABASE LAYER* - ITEM 6 E 7

Após converter o dado para uma medida útil e atualizar o objeto JSON residente em memória a função Dispatcher (item 6) é invocada com o o objetivo de repassar a mensagem, contendo os dados sensoreados, para as API implementadas no Terra Core. Além disso, é feita a persistência do dado com a gravação (item 7) em algum banco de dados relacional. A persistência do dado em banco de dados se justifica pela necessidade de acesso por aplicações que necessitam acessar e manter o dado por um determinado período de tempo.

As próximas subseções irão apresentar as três formas de interoperabilidade por meio de API que o Terra Core disponibiliza para o acesso de diferentes modalidades de aplicações finais.

3.9 – *WEBSOCKET CHANNEL* – TERRA *WEB APPLICATION* - ITEM 8

Uma das formas de interoperabilidade oferecidas pelo Terra Core e por meio de canais de conexão que fazem o uso do protocolo *WebSocket*. Nestes, o objetivo é permitir a conexão com navegadores da atualidade que são compatíveis com esse padrão de comunicação orientado a eventos. A opção de oferecer esse tipo de API se dá pela crescente demanda por aplicações Web e por se tratar de um padrão do W3C.

3.11 – ACESSO POR APLICAÇÕES FINAIS VIA *PUB/SUB CHANNEL* - ITEM 9

Outra forma de comunicação de grande relevância no que diz respeito ao contexto de aplicações consumidoras de dados de WSN se dá através de canais de comunicação Pub/Sub. Dessa forma, o Terra Core disponibiliza uma interface de integração por meio desses canais como forma de permitir que aplicações recebam o dado da rede em tempo real.

A interoperabilidade é feita de forma similar ao item 4, só que dessa vez com o Terra Core assumindo o papel de Publisher na comunicação. Aplicações interessadas nos dados da rede comunicam seu interesse se inscrevendo em um dos canais fornecidos pelo Terra Core e, a partir desse ponto, passam a receber mensagens com os dados na medida em que eles chegam.

No estudo de caso também foi feito o uso da biblitoteca ZeroMQ para implementar o canal Pub/Sub. Nela são presentes implementações nas mais diversas linguagens, permitindo assim a integração com a grande parte das aplicações existentes.

3.10 – ACESSO POR APLICAÇÕES FINAIS VIA API REST - ITEM 10

Como forma de realizar consultas mais específicas a registros de dados anteriores, o Terra Core implementa uma API RESTful como sua terceira forma de integração. Através desta, aplicações podem realizar consultas de forma inteligente e sob demanda, buscando por exemplo dados de um sensor específico da rede ou de temperaturas que atingiram um determinado limiar.

A API REST possui comunicação direta com o a camada de persistência da aplicação, realizando consultas ao banco de dados salvos previamente pelo Dispatcher (item 6) conforme a necessidade.

4 CENÁRIO DE APLICAÇÃO

Em diversos ambientes corporativos, institucionais ou industriais é comum encontrar produtos ou processos que dependem do monitoramento permanente de algum fenômeno como temperatura, pressão, umidade, etc. Clínicas de tratamento estético, por exemplo, podem ter que fazer o monitoramento da temperatura de certos produtos pois, na eventual ocorrência de falta de energia, o produto pode estragar ou não produzir mais o efeito esperado e isso pode ocasionar uma série de problemas para o paciente e para o médico que pretende utilizar o produto.

O cenário de aplicação proposto visa apresentar o funcionamento e os resultados do uso da arquitetura e aplicações propostas para o suporte de aplicações final interessadas no uso da arquitetura proposta. Por exemplo, será apresentada uma aplicação final que deseja fazer o monitoramento e controle de temperatura de produtos de tratamento estético que precisam ser mantidos permanentemente em faixas de temperatura controladas. O produto a ser monitorado é o Botox [42] (toxina botulínica) que é utilizado por médicos dermatologistas para tratamentos estéticos ou de saúde. Apesar do foco específico do cenário aqui apresentado, essa aplicação pode ser utilizada por um grupo de aplicações com características similares que visam o monitoramento de medicamentos ou produtos que precisam ser mantidos em freezer ou geladeira.

É comum a realização de compras mensais ou programadas do Botox. Após receber o material, o Médico precisa manter o produto em freezer antes do uso e, em geladeira, após a diluição. Uma compra programada pode, por exemplo, conter aproximadamente 20 frascos de 100 unidades cada. Dessa forma, é preciso armazenar e monitorar a temperatura dos frascos de Botox que normalmente são recebidos por serviços rápidos de entrega de encomendas, pois na ocorrência de problemas, o prejuízo financeiro pode ser elevado.

Os produtos são recebidos em um isopor de tamanho pequeno com barras de gelo em gel. Com isolamento e lacres apropriados o produto pode ficar aproximadamente 24 horas em transporte sem afetar a qualidade. Esse período de transporte é garantido pelo fabricante do produto.

Após o recebimento do isopor, contendo aproximadamente 10 caixas com o Botox é preciso armazenar em freezer em uma temperatura menor ou igual a -5ºC. Após a diluição com o soro para o uso conforme bula e protocolo adotado pelo médico, o produto deve ser mantido somente em geladeira sob um intervalo de temperatura entre 2ºC e 8ºC por um período de no máximo de 3 dias.

Nesse estudo de caso será feito apenas o controle de temperatura de frascos de Botox a vácuo que ainda não foram utilizados ou diluídos. Dessa forma, a temperatura deve ser igual ou inferior a -5ºC. Normalmente o produto é recebido no consultório do médico dermatologista.

Assim que o isopor com os frascos é recebido o produto deve ser armazenado no freezer do consultório do médico para que o monitoramento seja iniciado imediatamente. Toda vez que os produtos atingirem uma temperatura maior que -5ºC o Médico, a secretária ou qualquer pessoa que esteja cadastrada para receber os alertas, deve ser informada sobre o fato para que tome alguma providência. Normalmente, quando se identifica algum problema no Freezer é preciso transportar o produto para outro Freezer.

É muito comum que o médico não deixe todos os frascos no freezer do consultório. Normalmente, fica armazenado no consultório apenas o suficiente para o uso semanal, pois o restante é armazenado no freezer da casa do médico que também deve ter um sistema de monitoramento. O Sistema pode ser programado para fazer alertas por Twitter, SMS, email, etc. Não importa o tipo de aplicação final a ser utilizada, pois o fundamental é que exista uma forma de alerta ao médico ou secretária do médico para que possa tomar alguma providência em caso de aumento da temperatura.

Como o objetivo principal desse cenário de aplicação é apresentar uma prova de conceito do funcionamento da arquitetura proposta, incluindo as aplicações desenvolvidas. Será desenvolvido e implementado um protótipo de uma aplicação final com foco no tempo de resposta, desde o momento em que o dado foi lido no sensor do nó até o momento em que a aplicação final recebe o dado tratado para ser processado de acordo com as regras de negócios que foram programas na aplicação final.

A WSN utilizada no estudo de caso é baseada em Terra System proposto por Branco [20] com a utilização do simulador TOSSIM que é integrado ao Terra System e que pode simular uma WSN orientadas a VM. Foram criadas duas WSN com 3 nós cada, sendo um nó Sink node e mais dois nós, contendo um sensor de temperatura cada um, simulando o monitoramento de um freezer.

As próximas subseções irão apresentar o passo a passo para colocar o protótipo em funcionamento e os detalhes de implementação ou ajustes necessários, caracterizando os elementos da solução da arquitetura proposta. Por questões didáticas e para facilitar o entendimento, serão apresentados os elementos da arquitetura conforme a ordem descrita na figura 1 que apresentou a visão geral da arquitetura e aplicações propostas. Alguns itens não estão na sequencia numerada apresentada na figura 1, pois são elementos que não fazem parte da arquitetura proposta, entretanto, foi necessário ajusta-los para atender às necessidades do cenário de aplicação. Em seguida, serão apresentados os resultados dos testes realizados.

4.1 – SIMULAÇÃO DAS WSN

O Simulador nativo da atual versão de Terra só tem a opção para criar uma única WSN com no máximo 81 nós. Para adequar o simulador TOSSIM integrado ao Terra System às necessidades

do cenário de aplicação foi necessário modificar o script de carga do simulador para permitir a execução de mais de uma WSN simultaneamente. Dessa forma, foi feita a alteração do script em Python que está disponível para a consulta no apêndice A.

4.2 – TERRA DIA - DISSEMINATION APLICATION

O Terra Dia ou Terra Dissemination Application é uma solução proposta por Ribeiro [43] de suporte à atualização remota de bytecodes de WSN em Terra que faz parte da arquitetura LAURA que já foi brevemente apresentada na introdução.

Esse serviço permite que a atualização seja feita remotamente por meio do upload do código em terra via aplicativo Web e comunicação via HTTP, evitando o contato físico com o sink node. Quando os nós de uma WSN em Terra são configurados pela primeira vez, não existe um bytecode residente. É preciso executar a atualização para disseminar o primeiro bytecode.

Terra Dia foi feito para disseminar códigos homogêneos na versão de terra cujo simulador previa apenas uma WSN com no máximo 81 nós. Dessa forma, também foi necessário fazer ajustes no código do serviço Terra Dia para se adequar ao cenário de aplicação proposto. Foi necessário alterar o serviço para funcionar em linha de comando com o objetivo de facilitar a integração com o simulador já comentado no item 4.1 acima que foi modificado. Devido à existência de mais de uma WSN foi necessário programar para parametrizar a porta de comunicação, pois para cada WSN tem que ser uma porta diferente, permitindo disseminar um bytecode específico para cada WSN.

4.3 – TERRA GATEWAY

As características principais do serviço Terra Gateway já foram apresentadas na seção “3.4 – TERRA GATEWAY - ITEM 3”. O objetivo aqui é descrever detalhes de implementação e de adequação ao cenário proposto. Para permitir a comunicação com várias WSN ao mesmo tempo foi necessário configurar portas de comunicação dinâmicas de acordo com as WSN existentes. Devido à função de entrega dos dados recebidos das WSN para o Terra Core, que estabelece uma comunicação Pub/Sub com o Terra Gateway, foi necessário instanciar vários “Publishers” de acordo com as portas de comunicação que recebem dados das WSN. Portanto, cada porta de comunicação que o Terra Gateway mantem com o Sink Node de uma WSN é preciso ter uma instância de “Publisher” executada que será associada à um tópico de interesse. O Terra Core terá várias instâncias de “Subscribers” que irão se inscrever no respectivo tópico de interesse no Publisher. Dessa forma, quando um dado é recebido pelo Terra Gateway em uma determinada porta, ele será automaticamente reencaminhado para um tópico de interesse correspondente a essa porta. O Subscriber inscrito nesse tópico receberá os dados sensoreados relacionados com o tópico.

4.4 – TERRA CORE – MÓDULO DISPATCHER

As características principais do serviço Terra Core já foram apresentadas na seção 3.6 e 3.7. O objetivo aqui é descrever detalhes de implementação e de adequação ao cenário proposto. Para permitir a comunicação com várias WSN foi necessário fazer com que o Terra Core e, principalmente, o módulo *Dispatcher* reencaminhassem corretamente os dados para os respectivos canais de interesse para que a aplicação final tenha facilidade em receber os tópicos de interesse e, decidir, se deseja processar apenas um tipo de dado. Por exemplo, quando o Terra Core (Subscriber) faz a inscrição em determinado tópico do Terra Gateway (Publisher), irá receber por meio da porta que foi alocada para essa conexão, todos os dados sensoreados referentes a essa conexão que normalmente pode vir de uma WSN.

Com isso, o papel de filtrar os dados conforme a necessidade passa a ser da aplicação final. Por padrão, essas aplicações recebem dados de todas as redes configuradas e, decidem dentro de seu próprio contexto, a melhor maneira de exibí-los ou filtrá-los. Portanto, em uma configuração com múltiplas redes transmitindo mensagens através de diferentes portas, é possível identificar a origem do dado através de porta, que indica de qual rede ele veio, e do ID do nó responsável pela leitura.

Sendo assim, caso a aplicação apresente um interesse em um ou mais nós, ela pode referenciá-los facilmente através da porta e dos IDs desses nós. Por exemplo, uma aplicação interessada em mensagens de uma WSN executando na porta 9004 e no conjunto de nós de ID 8 à 11 pode escolher ignorar as outras mensagens da rede que chegam e exibir apenas as que chegarem apresentando um ID na faixa determinada e a porta sendo igual a 9004.

4.5 – TERRA WEB CONTROL

Conforme a visão geral apresentada na figura 1 da seção 3 a arquitetura é capaz de suportar mais de um tipo de aplicação final consumindo dados de mais de uma WSN em execução simultaneamente. A aplicação final Terra Web é um exemplo de aplicação final que utiliza uma API websocket e uma RESTful para acessar as interfaces de comunicação disponíveis em Terra Core. Essa aplicação foi utilizada como prova de conceito da arquitetura proposta. A API websocket permite mostrar o funcionamento do acesso a dados em tempo real via Pub/Sub e a API RESTful recebe e apresenta dados que não possuem restrições de tempo. A figura 2 apresenta a tela principal do protótipo do Terra Web no instante em 20 minutos após o início da execução do simulador que foi programado para executar duas WSN com três nós cada, sendo um nó no papel de Sink Node e os demais nós executando a leitura de temperatura.

Nesse cenário, os dados são recebidos pelo Terra Gateway que reencaminha para o Terra Core que disponibiliza as interfaces de comunicação. Por meio dessas interfaces o Terra Web vai

exibir os dados que são apresentados na figura 2. O item “A” apresenta, em tempo real, os dados recebidos pela interface WebSocket via comunicação Pub/Sub, mostrando, da esquerda para a direita, os respectivos dados: Porta de comunicação utilizada, ID do nó que enviou o dado e a temperatura sensoreada no respectivo nó. O item “B” apresenta um gráfico de linha em tempo real das medições feitas na linha do tempo. O eixo “X” apresenta a hora da medição e o eixo “Y” apresenta a temperatura medida. O item “C” apresenta o tempo médio de atraso entre a leitura da temperatura e a recepção e visualização do dado pelo Terra Web. O tempo médio é importante para analisar a eficácia da solução, pois as aplicações finais não devem receber o dado com longo atraso, pois o processamento das aplicações que compõem a arquitetura devem processar e reencaminhar os dados com eficácia para suportar aplicações finais que demandam processamento e exibição de dados sensoreados em tempo real. O atraso deve ser mínimo. O item “D” apresenta os dados que foram recebidos por meio da interface RESTful. Da esquerda para a direita temos os seguintes dados: Quantidade de mensagens enviadas pelo Terra Core conforme consta no Banco de dados. Valor atualizado constantemente. Em seguida, a temperatura média de todas as medições feita até um determinado instante, temperatura máxima medida e, por fim, a menor temperatura registrada.

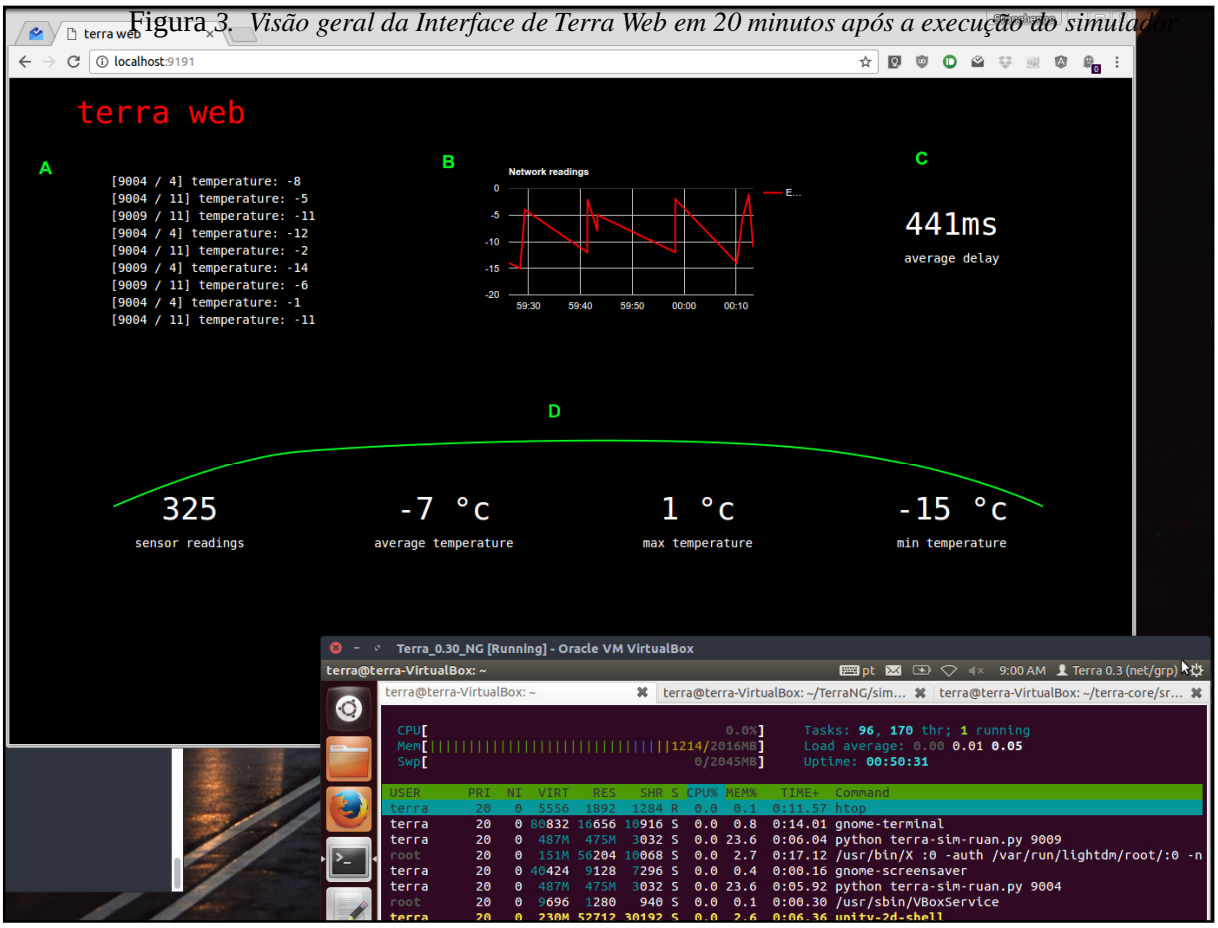

Figura 3. *Visão geral da Interface de Terra Web em 20 minutos após a execução do simulador*

Em 30 minutos após a captura da tela do protótipo do Terra Web, representado pela figura 2, foi feita a captura de uma nova tela que é apresentada na figura 3. Essa nova tela apresenta os dados atualizados e possibilita algumas análises. Uma observação importante antes da análise dos testes do protótipo é indicar a configuração do computador e da máquina virtual (VM) utilizada para executar o simulador e a aplicação final Terra Web. O protótipo (simulador e aplicação web estão rodando na mesma máquina) foi executado em um computador com processador x.y Ghz com 8 GB de RAM e com um disco SSD de 240GB. O protótipo foi executado em uma VM rodando Ubuntu versão 12.04 com 2 GB de RAM. Essa informação pode ser verificada logo abaixo da tela do Terra Web nas figuras 2 e 3.

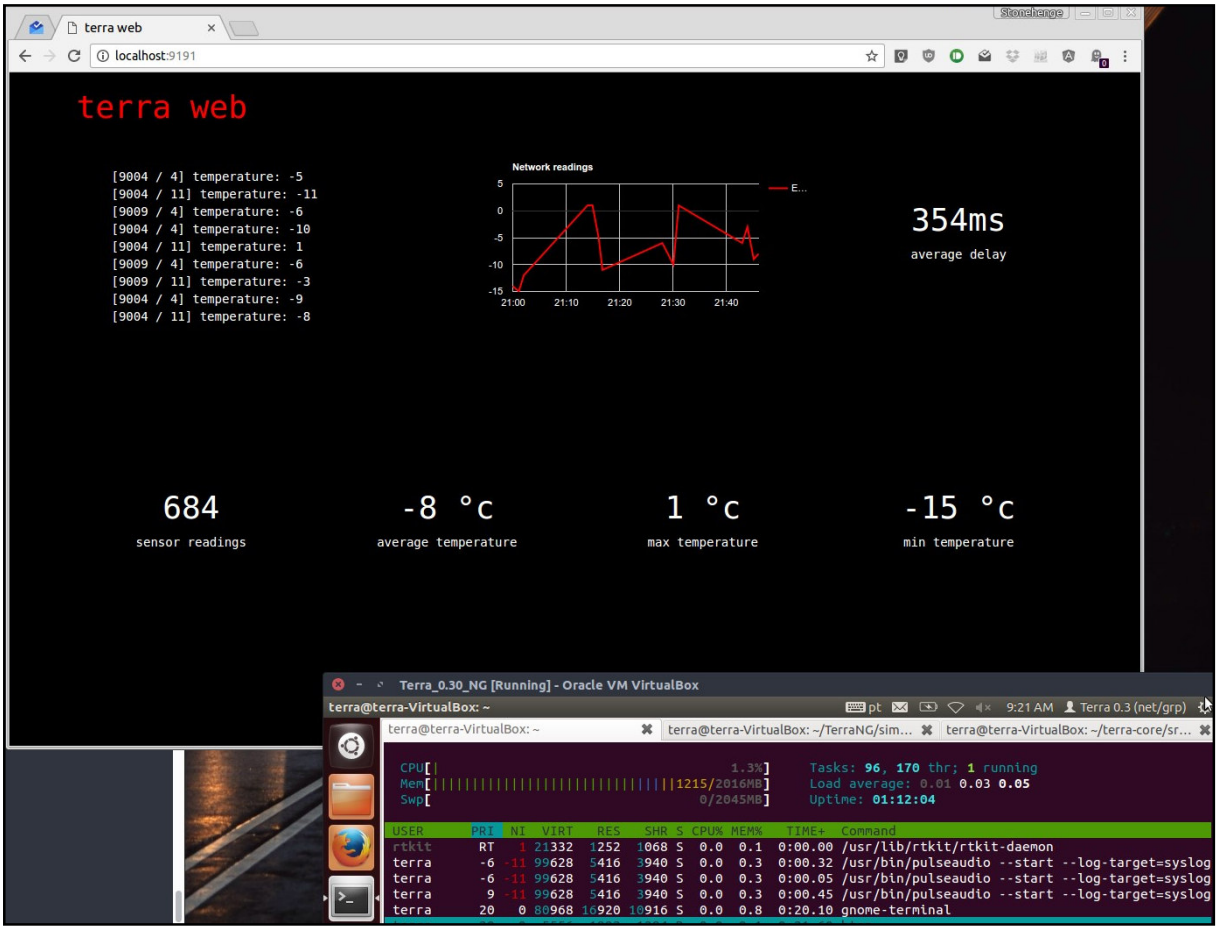

Figura 4. Visão geral da Interface de Terra Web em 50 minutos após a execução do simulador

O intervalo de leitura de temperatura definido do simulador foi entre -15 e 1 grau célsius. A emissão do alerta de temperatura será feita para a temperatura igual ou maior que -5 graus. Mesmo com o simulador rodando na mesma máquina que está executando os aplicativos da arquitetura e a aplicação final Terra Web foi possível verificar que o atraso entre a leitura da temperatura e a exibição da temperatura na aplicação final é aceitável para aplicações interativas de tempo real. Após 20 minutos de execução da aplicação e até 50 minutos foi constatado que o atraso foi se

estabilizando, juntamente com o uso do processador e da memória da VM. Conforme mostra a figura 3 o uso do processador ficou muito próximo de zero e o uso da memória ficou em 1.215MB. De acordo com a recomendação do ITU G.114 [44] para aplicações interativas, o atraso no pior caso deve ser de 400ms. O Atraso médio ficou estabilizado em aproximadamente 354ms, entretanto, rodando na mesma máquina que está rodando o simulador. Além disso, o atraso no pior caso recomendado pelo ITU se aplica a aplicações multimídia. Para aplicações de monitoramento o atraso não precisa ser tão rígido quanto aplicações de voz ou vídeo.

5 – CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Sistemas que busquem prover interoperabilidade entre WSN e aplicações finais devem levar em consideração que o ambiente das empresas é heterogêneo e com demandas de aplicações finais que devem ser compatíveis com o contexto das empresas. Com o aumento da demanda por aplicações WSN/IoT e com a previsão de carência de profissionais capacitados para trabalhar nessa área será, cada vez mais necessário, prover soluções que sejam capazes de abstrair os detalhes técnicos de baixo nível, provendo soluções que ofereçam interfaces padronizadas que possam facilitar o desenvolvimento de aplicações finais de WSN/IoT sem precisar de capacitação adicional ou conhecimentos dos detalhes de mais baixo nível.

A separação de preocupações também é outro fator vital para a escalabilidade de aplicações WSN/IoT, pois permite abstrair em níveis específicos as preocupações, facilitando o trabalho de equipes multidisciplinares para que o sistema seja capaz de responder rapidamente as demandas de mudanças ocasionadas por forças competitivas ou por mudança no modelo de negócio das empresas.

A solução proposta permite que o desenvolvedor da aplicação final não tenha que se preocupar ou conhecer os detalhes de baixo nível, pois pode simplesmente acessar as interfaces padronizadas disponibilizadas pela aplicação Terra Core. O tempo de resposta da solução é adequado para aplicações de monitoramento em tempo real.

Como proposta de trabalho futuro sugere-se o gerenciamento de clientes dos mais variados tipos, definindo mecanismos de autenticação com permissões e papéis específicos nos acessos às interfaces de comunicação disponibilizadas. Dessa forma um administrador poderia gerenciar e restringir o acesso à dados para clientes específicos.

REFERÊNCIAS

[1] - WEISER, M. The computer of the 21st century. Scientific American, pp. 94–104, 1991.

[2] - How Smart City Barcelona Brought the Internet of Things to Life. Disponível em <http://datasmart.ash.harvard.edu/news/article/how-smart-city-barcelona-brought-the-internet-of-things-to-life-789>. Acesso em: 20 jan. 2016

[3] - Is Amazon Echo the next enterprise IoT platform? Disponível em <http://www.cio.com/article/3047524/internet-of-things/is-amazon-echo-the-next-enterprise-iot-platform.html>. Acesso em: 20 jan. 2016

[4] - MACGILLIVRAY, C.; TURNER, V.; SHIRER, M. Explosive Internet of Things Spending to Reach $1.7 Trillion in 2020. IDC Corporate USA: Framingham, Massachusetts: IDC, 2015. Disponível em: <http://www.idc.com/getdoc.jsp?containerId=prUS25658015>. Acesso em: 04 dez. 2016.

[5] What is the Internet of Things (IoT)? Disponível em <http://www.businessinsider.com/what-is-the-internet-of-things-definition-2016-8?IR=T>. Acesso em: 20 jan. 2016.

[6] - AKYILDIZ, I. A Survey on sensor networks. IEEE Communications Magazine, vol.40, nº 8, p. 102-114, ago. 2002.

[7] - YICK, J.; MUKHERJEE, B.; GHOSAL, D. Wireless sensor network survey. Computer Networks, vol. 52, p.2292-2330, ago. 2008.

[8] - SU, K.; LI, J.; FU, H. Smart city and the applications. Electronics, Communications and Control (ICECC), 2011 International Conference on IEEE; 2011. pp. 1028–1031.

[9] - SOUZA, S. C. Uma abordagem baseada em regras e computação em nuvem para desenvolvimento de aplicações em redes de sensores sem fio. 2013. Dissertação (Mestrado em Informática) - UFES/CT/PPGI, Vitória -ES.

[10] - ALAMRI, A.; ANSARI, W.S; HASSAN, M. M.; HOSSAIn, M. S.; ALELAIWI, A.; Hossain, M. A. A Survey on Sensor-Cloud: Architecture, Applications, and Approaches in International Journal of Distributed Sensor Networks, 2013. Disponível em: <http://dx.doi.org/10.1155/2013/917923>. Acesso em: 04 dez. 2016.

[11] - HUGHES, A.; MURRAY, S. IDC Survey Reveals Majority of Manufacturers Worldwide Using Public or Private Cloud. IDC Corporate USA: Framingham, Massachusetts: IDC, 2015. Disponível em: <https://www.idc.com/getdoc.jsp?containerId=prUS25558515>. Acesso em: 04 dez. 2016.

[12] - DUAN, Q.; YAN, Y.; VASILAKOS, A.V. A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing in IEEE Transactions on Network and Service Management, Vol. 9, No. 4, December, 2012.

[13] - CHUTE, C. Benchmarking Cloud Maturity Against Line-of-Business Growth. Regional 2015

SMB Cloud Adoption Survey. IDC Corporate USA: Framingham, Massachusetts: IDC, 2015. Disponível em: <http://www.idc.com/getdoc.jsp?containerId=254023>. Acesso em: 04 dez. 2016.
[14] - Meyer, S.; Ruppen, A.; Magerkurth, C. Internet of Things-aware Process Modeling: Integrating IoT Devices as Business Process Resources in 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013
[15] - Tranquillini, S.; Spieß, P.; Daniel, F.; Karnouskos, S.; Casati, F.; Oertel, N.; Mottola, L.; Oppermann, F.J.; Picco, G.P.; Römer, K.; Voigt, T. Process-Based Design and Integration of Wireless Sensor Network Applications. Business Process Management 2012.
[16] - Hughes, A.; Murray, S. IDC Survey Reveals Majority of Manufacturers Worldwide Using Public or Private Cloud. IDC Corporate USA: Framingham, Massachusetts: IDC, 2015. Available online: https://www.idc.com/getdoc.jsp?containerId=prUS25558515 (accessed on 12 dez 2016).
[17] - Alamri, A.; Ansari, W.S; Hassan, M. M.; Hossain, M. S.; Alelaiwi, A.; Hossain, M. A. A Survey on Sensor-Cloud: Architecture, Applications, and Approaches in International Journal of Distributed Sensor Networks, 2013. http://dx.doi.org/10.1155/2013/917923
[18] - Dai, C. ; Wang, Z. A flexible extension of WSDL to describe nonfunctional attributes 2nd International Conference on e-Business and Information System Security (EBISS), 2010.
[19] Portal Fator Brasil. Cisco impulsiona formação de cientistas, engenheiros e inventores em torno da Internet das Coisas, 2013. Disponível em: <http://www.revistafatorbrasil.com.br/ver_noticia.php?not=251796> Acesso em: 20 jan. 2016.
[20] Adriano Branco, Francisco Sant'anna, Roberto Ierusalimschy, Noemi Rodriguez, and Silvana Rossetto. Terra: Flexibility and safety in Wireless Sensor Networks. ACM Transactions on Sensor Networks, 11(4):59:1-59:27, September 2015.
[21] Node.js. Disponível em <https://nodejs.org/en/>. Acessado em 20 jan. 2016
[22] Stack Overflow Developer Survey Results 2016. Disponível em <http://stackoverflow.com/research/developer-survey-2016> Acesso em 20 jan. 2016.
[23] Socket.io. Disponível em <http://socket.io/>. Acesso em 20 jan. 2016.
[24] ZeroMQ. Disponível em <http://zeromq.org/> Acesso em 20 jan. 2016.
[25] TOSSAM. Disponível em <http://www.inf.ufg.br/~brunoos/tossam/>. Acesso em 20 jan. 2016.
[26] Delicato, F.C., "Middleware Baseado em Serviços para Redes de Sensores sem Fio", (In Portuguese), PhD Thesis, Federal University of Rio de Janeiro, Brazil, June, 2005
[27] Alamri, A.; Ansari, W.S; Hassan, M. M.; Hossain, M. S.; Alelaiwi, A.; Hossain, M. A. A Survey on Sensor-Cloud: Architecture, Applications, and Approaches in International Journal of Distributed Sensor Networks, 2013. http://dx.doi.org/10.1155/2013/917923
[28] BOULIS, C. A.; Han, C.; SRIVASTAVA, M. B. Design and implementation of a framework for efficient and programmable sensor networks. International Conference On Mobile Systems, Applications And Services, pp. 187–200, 2003.

[29] LEVIS, P. et al. TinyOS: An operating system for sensor networks. In Ambient Intelligence. [S.l.]: Springer Verlag, 2004. 1.2, 2.1, 2.1.1, 3.1
[30] A Flexible Web Service based Architecture for Wireless Sensor Network (6)
[31] D. Guinard and V. Trifa, “Towards the Web of Things: Web Mashups for Embedded Devices,” Proc. Workshop Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM’09), Apr. 2009 (7)
[32] Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services (8)
[33] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine.
[34] - Akyildiz, I. “A Survey on sensor networks” in IEEE Communications Magazine, vol.40, nº 8, p. 102-114, ago. 2002.
[35] - Yick, J.; Mukherjee, B. ; Ghosal, D. "Wireless sensor network survey" in Computer Networks, vol. 52, p.2292-2330, ago. 2008.
[36] - Su, K.; Li, J.; Fu, H. Smart city and the applications. In Electronics, Communications and Control (ICECC), 2011 International Conference on IEEE; 2011. pp. 1028–1031
[37] - Corredor, I., Bernardos, A. M.,Iglesias, J. Model-Driven Methodology for Rapid Deployment of Smart Spaces Based on Resource-Oriented Architectures in SENSORS, v 12, 7, p. 9286-9335, 2012.
[38] - Alamri, A.; Ansari, W.S; Hassan, M. M.; Hossain, M. S.; Alelaiwi, A.; Hossain, M. A. A Survey on Sensor-Cloud: Architecture, Applications, and Approaches in International Journal of Distributed Sensor Networks, 2013. http://dx.doi.org/10.1155/2013/917923
[39] - Hughes, A.; Murray, S. IDC Survey Reveals Majority of Manufacturers Worldwide Using Public or Private Cloud. IDC Corporate USA: Framingham, Massachusetts: IDC, 2015. Available online: https://www.idc.com/getdoc.jsp?containerId=prUS25558515 (accessed on 4 Apr 2016).
[40] - Duan, Q.; Yan, Y.; Vasilakos, A.V. A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing in IEEE Transactions on Network and Service Management, Vol. 9, No. 4, December, 2012.
[41] - Moghe, U.; Lakkadwala, P.; Mishra, D.K. Cloud Computing: Survey of different utilization techniques in CSI Sixth International Conference on Software Engineering (CONSEG), 2012.
[42] Botox. Available online: http://www.allergan.com.br/Bulas/Documents/Botox_profissional.pdf (accessed on 20 jan 2016).
[43] – Ribeiro, F. Terra Dia: Uma Solução de Suport a Atualização Dinâmica de Bytecodes em Redes de Sensores Sem Fio Baseadas em Máquina Virtual. 2016. Dissertação (Projeto de Graduação) - UFES/CT/PPGI, Vitória -ES.

[44] - Brosh, E.; Baset, S.A.; Misra, V.; Rubenstein, D. The Delay-Friendliness of TCP for Real-Time Traffic in IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 18, NO. 5, OCTOBER 2010.

APÊNDICE A – CÓDIGOS FONTES MAIS RELEVANTES DA APLICAÇÃO

Terra Gateway

O código abaixo apresenta o loop principal do Terra Gateway. Neste loop, que executa até que o nó se desconecte do canal TOSSA, o programa se mantém escutando por mensagens da rede e as publica no canal criado pela biblioteca ZeroMQ.

```
local tossam = require("tossam")
local context = zmq.init(1)
local publisher = context:socket(zmq.PUB)
local exit = false

publisher:bind("tcp://*:"..port)

while not(exit) do
  local mote = tossam.connect {
    protocol = "sf",
    host     = host,
    port     = port,
    nodeid   = 1
  }
  if not(mote) then print("  ! connection error\n  ! aborting"); return(1); end

  // ...


  while (mote) do
    local stat, msg, emsg = pcall(function() return mote:receive() end)
    if stat then
      print(emsg)
      if msg then
        msg.port = port
        msg.gateway_time = os.time() * 1000

        publisher:send("event", zmq.SNDMORE)
        publisher:send(json.encode(msg))

      else
        if emsg == "closed" then
          print("\nConnection closed!")
          exit = true
          break
        end
      end
    else
      print("\nreceive() got an error:"..msg)
      exit = true
      break
    end
  end
```

```
  mote:unregister()
  mote:close()
end

publisher:close()
context:term()
```

Terra Core – Listener

O código a seguir é responsável por receber os eventos emitidos pelo Terra Gateway. Nele o Subscriber é se conecta ao Terra Gateway através de portas pré-definidas e se mantém escutando as mensagens. Após chegar uma nova mensagem da rede é feito o “parse” e a mesma é imediatamente despachada para o módulo Dispatcher.

```
const zmq = require('zmq')
const subscriber = zmq.socket('sub')
const moment = require('moment')
const sample = require('lodash/sample')
const random = require('lodash/random')
const dispatcher = require('./event-dispatcher')

const ports = [9002, 9003, 9004, 9005, 9006, 9007, 9008, 9009, 9010, 9011, 9012]

const init = (app) => {

  subscriber.subscribe('event')

  ports.map(port => subscriber.connect('tcp://localhost:' + port))

  subscriber.monitor(500, 0)

  subscriber.on('subscribe', (fd, ep) => console.log('-- connected to publisher'))

  subscriber.on('message', (channel, message) => {
    let m = JSON.parse(message.toString())

    dispatcher.dispatchEvent({
      port: m.port,
      id_mote: m.source,
      gateway_time: m.gateway_time,
      value: getTemperature()
    })

    console.log(' -- new message:', channel.toString(), message.toString())
  })

}
```

Dispatcher

O módulo dispatcher concentra as funções que devem ser executadas para cada mensagem. No trecho de código abaixo é possível ver que, para cada mensagem, ela é enviada para o cliente web e salva no banco de dados – para que possa ser obtida através da API RESTful posteriormente.

```
const dispatchEvent = (event) => {
  // send to web client
  io.emit('message', event)

  // parse gateway_time
  event.gateway_time = new Date(event.gateway_time)

   eventModel
     .addEvent(event)
     .catch(console.log)

}
```

Terra Core - API RESTful

No trecho a seguir, que é executado assim que a aplicação começa a executar, são iniciados os módulos da aplicação Terra Core (linhas 7 e 9) e logo em seguida os métodos da API RESTful (linhas 11 à 15). Através dos endpoints implementados é possível realizar requisições à API e receber dados em retorno.

Requisições ao endpoint da linha 11 retornam todas as mensagens recebidas pelo Terra Core até aquele instante. Em seguida, requisições ao metodo da linha 13, recebem como resposta estatísiticas sobre as mensagens recebidas como a temperatura máxima, o total de mensagens, entre outras. Caso o interesse seja em uma única mensagem, ela pode ser referenciada através de seu ID na linha 15, onde são retornadas apenas as informações de uma determinada leitura.

```
1.  const listener = require('./event-listener')
2.  const dispatcher = require('./event-dispatcher')
3.  const eventController = require('./event-controller')
4.
5.  const init = (app) => {
6.
7.   dispatcher.init(app)
8.
9.   listener.init(app)
10.
11.  app.get('/api/event', eventController.getEvents)
12.
13.  app.get('/api/event/stats', eventController.getStats)
14.
15.  app.get('/api/event/:id', eventController.getEvent)
16.
17. }
```

Script de Simulação

O trecho abaixo foi extraído do script usado para inicializar a simulação das redes Terra. Para adequar a simulação ao estudo de caso proposto, foram necessárias alterações ao script de simulação já existente, elaborado inicialmente por Branco [20].

Na função “main” abaixo são inializados os 3 nós da rede, de IDs 1, 11 e 4, sendo o nó 1 o *Sink node*. Na função inicMote são instanciados os nós com base no ID fornecido e conectados ao nó Sink.

```
1.  def main(nLines,nColumnsA,nColumnsB,timeB,runTime, port):
2.
3.    # instantiate the network
4.    net = Network(maxNode, port)
5.    if net.portStat == 0 :
6.      time.sleep(10)
7.      return
8.
9.    # initialize Terra Viewer
10.   wcmd("init",nColumns,nLines,runTime)
11.   net.inicMote1()
12.   net.inicMote(11)
13.   net.inicMote(4)
14.
15.   net.execTime(timeB)
16.
17.   remainTime = runTime - net.getLocalTime()/1000
18.   net.execTime(remainTime)
19.
20.
21. def inicMote(self,mote):
22.     Gain = 85.0
23.     """Switch-on a mote"""
24.     moteId = self.MoteInic+mote
25.     self.tossim.getNode(moteId).bootAtTime(self.tossim.time())
26.
27.     self.radio.add(1,moteId,Gain)
28.     self.radio.add(moteId,1,Gain)
29.
30.     # need at least one step to excute the command
31.     self.execSteps(1)
32.
```