Patrícia Teixeira Davet

Plataformas para a IoT: Desafios e Projetos

Trabalho Individual II apresentado ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação

Orientador: Prof. Dr. Adenauer Corrêa Yamin
Coorientadora: Profa. Dra. Ana Marilza Pernas Fleischmann

Pelotas, 2015

RESUMO

DAVET, Patrícia Teixeira. **Plataformas para a IoT: Desafios e Projetos**. 2015. 49 f. Trabalho Individual II (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2015.

Na Computação Ubíqua (UbiComp) os diversos sistemas computacionais interagem com o ser humano a todo o momento, e de forma o mais transparente possível. Como um meio de materializar as premissas da UbiComp, a Internet das Coisas (IoT) vem ganhando destaque mundial.

Na perspectiva da IoT qualquer "coisa" (pessoa, animal ou objeto) pode interoperar através da Internet com identificação única, constituindo assim um objeto inteligente por meio de dispositivo computacional embarcado que fornece funções de sensoriamento, processamento e comunicação.

No entanto, há uma série de desafios para a plena implantação da IoT. Um dos principais desafios está relacionado à concepção de infraestruturas capazes de integrar os inúmeros dispositivos da IoT, com caracteristicas heterogêneas, escaláveis, dinâmicas e restritas, de maneira que estes possam interoperar autonomicamente.

Como uma abordagem promissora para tratar desafios IoT, middlewares vem sendo utilizados na concepção de infraestruturas para IoT. Estes fornecem uma interface de alto nível, provendo um meio padronizado para o acesso aos dados e serviços fornecidos pelos dispositivos.

Desta forma, este trabalho apresenta os principais requisitos necessários para middlewares IoT, com o intuito de contemplar as principais características almejadas para as infraestruturas IoT. Também são revisadas estratégias de interoperação dos dispositivos IoT, bem como realizado análise de trabalhos que representam plataformas de middleware para IoT.

Palavras-chave: Middlewares IoT, Internet das Coisas, Computação Ubíqua.

ABSTRACT

DAVET, Patrícia Teixeira. **Platforms for IoT: Challenges and Projects**. 2015. 49 f. Trabalho Individual II (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2015.

In Ubiquitous Computing (UbiComp) various computer systems interact with the human being at every moment, and of so as transparent as possible. As a means to materialize the premises of UbiComp, the Internet of Things (IoT) is gaining worldwide prominence.

From the perspective of IoT any “thing” (person, animal or object) can interoperate across the Internet with unique ID, constituting like this an intelligent object through embedded computing device that provides sensing functions, processing and communication.

However, there are a number of challenges to the full implementation of IoT. A major challenge is related to the design of infrastructure able to integrate the many IoT devices with heterogeneous, scalable, dynamic and restricted features, so that they can interoperate autonomically.

As a promising approach to treat IoT challenges, middleware has been used in the design of infrastructures for IoT. These provide a high-level interface, providing a standardized means for access to data and services provided by devices.

Thus, this work presents the main requirements for IoT middleware, in order to contemplate the major required characteristics for IoT infrastructure. Also strategies of interoperation of IoT devices are reviewed, as well as analysis of works representing middleware platforms for IoT.

Keywords: IoT MIddlewares, Internet of Things, Ubiquitous Computing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environment
DLNA	Digital Living Network Alliance
DPWS	Device Profile for Web Services
EEML	Extended Environments Markup Language
EMML	Enterprise Mashup Markup Language
GENA	Generic Event Notification Architecture
HTML	HyperText Markup Language
HTTPU	HyperText Transfer Protocol Unicast
HTTPMU	HyperText Transfer Protocol Multicast
IETF	Internet Engineering Task Force
IoT	Internet of Things
JPA	Java Persistence API
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
MQTT	Message Queuing Telemetry Transport
MySQL	My Structured Query Language
NFC	Near Field Communication
OSGi	Open Services Gateway Initiative
RAM	Random Access Memory
RDF	Resource Description Framework
REST	Representation State Transfer
RFID	Radio Frequency Identification
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol

SSDP	Simple Service Discovery Protocol
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	eXtended Markup Language

SUMÁRIO

1 INTRODUÇÃO

A Internet das Coisas, do inglês *Internet of Things* (IoT), é um termo utilizado para designar uma rede de objetos inteligentes conectados à Internet (ATZORI; IERA; MORABITO, 2010). Objetos estes equipados com dispositivos embarcados unicamente identificados e dotados de capacidade de sensoriamento, armazenamento e processamento de dados captados das mais diferentes "coisas". O que leva a um cenário onde tudo pode estar conectado, desde objetos do nosso dia a dia como cafeteiras, geladeiras, como até mesmo animais e o nosso corpo, gerando e trocando informações, com o mínimo de intervenção humana possível.

Nesta perspectiva a IoT, enquanto uma infraestrutura computacional, vem ganhando destaque como uma abordagem para materializar as premissas da Computação Ubíqua, a qual tem como objetivo fornecer serviços computacionais de forma o mais transparente possível, independente de infraestrutura e tecnologias empregadas, não necessitando que o usuário tenha conhecimento de todo o sistema e/ou elementos deste ambiente (LOPES et al., 2014).

Os avanços tecnológicos na miniaturização de dispositivos embarcados e a agregação de novos e diferentes sensores, atuadores e tags inteligentes, foram os principais fatores que possibilitaram o surgimento da IoT. Porém para a plena implantação deste novo cenário introduzido pela IoT é necessário a adoção de princípios que apoiam-se em infraestruturas baseadas em padrões abertos de hardware e software, escaláveis e seguras no âmbito das tecnologias de informação e comunicação.

Outrossim, estes princípios devem proporcionar uma convergência entre tecnologias sem fio, dispositivos eletrônicos e a Internet. O que vem exigindo avanços na concepção de novos dispositivos eletrônicos, evolução das redes sem fio, desenvolvimento de novas estratégias de middlewares, protocolos de comunicação, aplicações e *frameworks* (GOUVEIA, 2013).

1.1 Tema

Este Trabalho Individual tem como tema o estudo dos desafios inerentes na concepção de infraestruturas para a IoT, de forma a caracterizar os requisitos necessários e características desejadas para a concepção de middlewares, os quais vem sendo utilizados para endereçar tais desafios.

1.2 Motivação e Objetivos

Os recentes avanços na área da Internet das Coisas, têm proporcionado uma crescente incorporação de dispositivos computacionais com capacidades de processamento, comunicação e sensoriamento a objetos cotidianos, ditos objetos inteligentes, os quais constituem fontes geradoras de informações contextuais distribuídas (PERERA et al., 2013).

Os dispositivos ou objetos inteligentes da IoT devem ser capazes de detectar fenômenos (físicos ou lógicos) e/ou disparar ações sobre o ambiente, bem como possuir alguma capacidade computacional e de comunicação, que os capacitem a interoperarem de forma o mais autônoma possível (FORSSTRÖM; KANTER, 2014). Tendo em vista estas características, a IoT vem constituindo-se elemento essencial para consolidar a integração entre os sistemas computacionais e o ambiente físico. Por ser uma área relativamente nova, ainda possui problemas de pesquisa em aberto, o que estimula o envolvimento da comunidade científica no tema.

Um dos principais desafios está na concepção de infraestruturas capazes de interligar, reconhecer e armazenar o grande número de informações geradas pelos dispositivos da IoT, de forma a constituírem informações relevantes para as aplicações e seus usuários. Levando em consideração, o fato, que estes dispositivos possuem capacidade restrita e são constituídos por diferentes tecnologias, tanto de hardware como de software, bem como de protocolos de comunicação e formatos de dados. Registra-se uma tendência na utilização de middlewares para endereçar tais desafios (PIRES et al., 2015).

O desenvolvimento de plataformas de middleware especificamente voltadas para ambientes de IoT é uma área de pesquisa recente que tem atraído a atenção da indústria e da comunidade acadêmica.

O presente trabalho tem como objetivos: (i) caracterizar os desafios inerentes na concepção de infraestruturas IoT, (ii) sistematizar estratégias para interoperação entre os dispositivos da IoT, (iii) apresentar os protocolos de comunicação para IoT que a literatura mostrar como mais usuais e consolidados e (iv) realizar revisão e análise de plataformas de middlewares para a IoT.

1.3 Estrutura do Texto

O texto é composto por 4 capítulos. No capítulo 1 são apresentados o tema, motivação e objetivos deste trabalho.

O Capítulo 2 caracteriza os desafios na concepção de infraestruturas IoT, apresenta o estilo arquitetural REST, o qual vem sendo amplamente utilizado no âmbito da IoT, bem como os principais protocolos de comunicação em uso atualmente na IoT, no que concerne a interoperabilidade de dispositivos restritos, como também características de zero configuração.

Quatro plataformas de middleware são descritas e analisadas no Capítulo 3, mediante a satisfação de critérios que constituem desafios IoT. Por fim o Capítulo 4 apresenta as considerações finais.

2 DESAFIOS E ESTRATÉGIAS DE INTEROPERAÇÃO NA IOT

A Internet das Coisas, do inglês *Internet of Things* - IoT, vem ganhando destaque como o novo paradigma de evolução da Internet (PERERA et al., 2013). A consequência desta evolução deve-se ao fato que a IoT preconiza a ideia do tudo conectado, ou seja qualquer "coisa", tais como utensílios, equipamentos pessoais, além de sistemas de transportes e agrícolas, redes de energia, como também animais e o nosso corpo podem possuir sensores capazes de gerar e compartilhar informações tendo como principal meio de comunicação, a Internet.

São várias as definições encontradas na literatura para o termo Internet das Coisas, diferenciando-se de acordo com a visão de pesquisa. Para o âmbito deste trabalho, o qual procura atender as premissas da Computação Ubíqua, que é prover computação de forma o mais transparente possível de acordo com as demandas do usuário, a definição que se mostrou mais alinhada, contemplando a visão de pesquisa requerida, é a seguinte: "A Internet das Coisas permite que pessoas e objetos possam se conectar a qualquer momento, em qualquer lugar, com qualquer coisa, de preferência usando qualquer caminho ou rede e qualquer serviço" (GUILLEMIN; FRIESS, 2009).

O tipo de informação disponibilizada por um cenário IoT, pode ser provida apenas para um domínio específico, como também compartilhado com outros domínios de aplicação, gerando dados de maior valor agregado e podendo tomar uma proporção considerável, tanto em termos de números de dispositivos integrados, como de dados gerados. A Figura 1 demonstra a dimensão deste fato.

O grande facilitador para a integração destes diversos dispositivos independentemente de sua localização, por ser uma rede global, foi a Internet. Porém o que possibilitou efetivamente a materialização da IoT foram fatores tais como melhora na mobilidade, conectividade e comunicação dos dispositivos computacionais, avanços tecnológicos na miniaturização de dispositivos embarcados e agregação de novos e diferentes sensores, atuadores e tags inteligentes. No entanto, há ainda uma série de desafios a serem superados para alavancar a ampla disseminação da IoT.

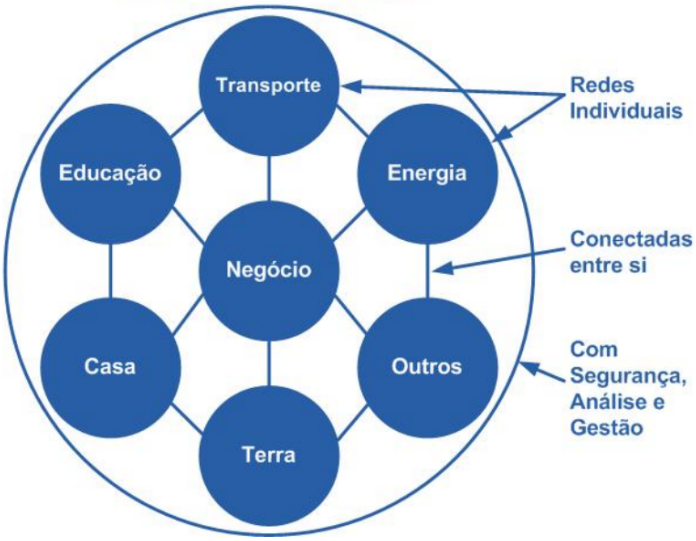

Figura 1: Dimensão de um Cenário IoT. Fonte: (EVANS, 2011)

Um dos principais desafios inerentes em um cenário IoT, está relacionado com a alta heterogeneidade decorrente da diversidade de tecnologias de hardware e software presentes neste ambiente. Esta situação demanda que haja uma busca de soluções que permitam a interoperabilidade e integração destes diferentes componentes. Uma alternativa de solução promissora que vem sendo amplamente utilizada, de forma a tratar desafios IoT, como o da heterogeneidade, está na utilização de plataformas de middleware.

As plataformas de middleware são inseridas entre as aplicações e a infraestrutura (de comunicação, processamento e sensoriamento) subjacente, provendo um meio padronizado para o acesso aos dados e serviços fornecidos pelos objetos por meio de uma interface de alto nível (BANDYOPADHYAY et al., 2011).

Dentre as pesquisas realizadas tendo como foco infraestruturas para IoT, o projeto europeu IoT-A (*Internet of Things - Architecture*) (IOT-A, 2014) destaca-se por tratar de maneira ampla os desafios para a concepção destas infraestruturas. Este teve como objetivo criar uma arquitetura de referência para a Internet das Coisas, que fosse capaz de trazer interoperabilidade entre dispositivos, entidades e sistemas distintos.

A visão do projeto IoT-A pode ser demonstrada como uma forma de caracterizar um cenário IoT, extremamente heterogêneo e escalável. Esta é apresentada na Figura 2 e caracterizada simbolicamente por uma árvore, onde as raízes abrangem todo um conjunto de tecnologias, como os de protocolos de comunicação (6LoWPAN, Zigbee, IPv6 e outros) e de dispositivos (sensores, atuadores, etiquetas RFID, etc.) representando a heterogeneidade presente em um cenário IoT, enquanto as flores/folhas da árvore demonstram todo o conjunto de aplicações em seus mais diferentes domínios, que podem ser construídos a partir da seiva (i.e., dados e informações) fornecidos pela raiz.

Nesta visão arquitetural percebe-se a necessidade de um elo de ligação entre os componentes representados pelas raízes e folhas, e que constitui a solução para grande parte dos desafios IoT. O tronco caracteriza este elo e pode ser representado por uma plataforma de middleware.

Figura 2: Árvore IoT-A - Visão do Projeto. Fonte: (IOT-A, 2014)

2.1 Requisitos de Middleware para IoT

Existem algumas propostas de middleware para IoT (MAIA et al., 2015), (PIRES et al., 2014), (FERREIRA, 2014), cada uma atendendo um subconjunto de requisitos necessários para viabilizar tais ambientes, conforme os objetivos e necessidades de aplicações e usuários a qual se destinam. Porém nenhuma que atenda a todos os requisitos inerentes à ambientes IoT.

De acordo com a literatura (CHAQFEH; MOHAMED et al., 2012), (BANDYOPADHYAY et al., 2011), (NAGY et al., 2009) os seguintes requisitos merecem destaque por serem considerados fundamentais para plataformas de middleware IoT:

- Interoperabilidade
- Descoberta e Gerenciamento de dispositivos
- Interfaces de Alto Nível

- Ciência de Contexto
- Escalabilidade
- Gerenciamento de Grandes Volumes de Dados
- Segurança e Privacidade
- Adaptação Dinâmica

Dentre os requisitos, o considerado primordial e que deve ser imperativamente endereçado por uma plataforma de middleware para a IoT, diz respeito a **interoperabilidade** entre os diversos dispositivos e plataformas disponíveis neste ambiente. A sua importância é devido a um número crescente de dispositivos a serem integrados neste novo cenário, sendo estes heterogêneos tanto em termos de hardware quanto de software, como também de protocolos (muitos deles proprietários) e formatos de dados, o que caracteriza a interoperabilidade em um cenário IoT um grande desafio.

Em (PIRES et al., 2015) é destacado que a integração de dispositivos no contexto de IoT perpassa múltiplos níveis:

- em mais baixo nível, é necessário integrar, de maneira transparente, uma miríade de dispositivos físicos heterogêneos de modo a ocultar detalhes com relação à rede, aos formatos de dados empregados, e até mesmo à semântica das informações;
- em um nível intermediário, a fim de prover serviços de valor agregado aos usuários, é necessário integrar e disponibilizar dados providos por esses dispositivos, podendo incluir simples funções de processamento de dados ou mesmo aplicações Web mais complexas;
- por fim, em mais alto nível, um modelo padronizado de programação pode promover integração no que se refere à agregação e transformação de informações providas pelos dispositivos, de modo que desenvolvedores de aplicações não necessitam ter qualquer conhecimento acerca das especificidades dos dispositivos físicos e do ambiente de rede subjacente.

Com diferentes dispositivos interoperando, pode-se fazer uso de informações providas de diferentes meios e/ou aplicações, permitindo que sejam criadas aplicações com maior valor agregado para os usuários.

Uma outra característica comum à ambientes de IoT leva em conta o fato da sua infraestrutura de comunicação possuir uma topologia dinâmica e frequentemente desconhecida, visto que dispositivos podem ser integrados ao ambiente e utilizados de maneira oportunista e não previamente planejada (BANDYOPADHYAY et al., 2011).

Dessa forma, é importante que uma plataforma de middleware possibilite a **descoberta de dispositivos** presentes no ambiente em questão, realizada dinamicamente a fim de atender os requisitos das aplicações. Além disso, é necessário prover mecanismos para o **gerenciamento de dispositivos**, que diz respeito à capacidade de fornecer informações de localização e estado do dispositivo permitindo, dentre outras funcionalidades, desconectar algum dispositivo roubado ou não reconhecido, atualizar software embarcado, modificar configurações de segurança, modificar remotamente configurações de hardware, localizar um dispositivo perdido, apagar dados sensíveis de dispositivos, e até mesmo possibilitar a interação entre dispositivo.

A adoção de uma plataforma de middleware também pode contribuir para facilitar a construção de aplicações para IoT. Desta forma, o desafio reside no fato de que, a fim de permitir a criação de aplicações que combinem recursos do mundo físico disponibilizados via Web, são necessários modelos de alto nível que abstraiam os serviços e dispositivos físicos subjacentes. Com isso, usuários e aplicações consumidores dos dados originados dos dispositivos conectados, podem possuir acesso aos dados de forma padronizada, por meio de **interfaces de alto nível**, não necessitando assim lidar com funcionalidades de baixo nível para a manipulação de tais objetos.

Ciência de contexto é outro requisito importante para plataformas de middleware IoT. Uma definição de contexto clássica e ainda bastante referenciada é a que Dey propôs em (DEY, 2001) em que contexto é “qualquer informação que caracteriza a situação de uma entidade, sendo que uma entidade pode ser uma pessoa, um lugar ou um objeto considerados relevantes para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação. O contexto é tipicamente a localização, a identidade e o estado das pessoas, grupos ou objetos físicos e computacionais”.

Portanto dados de contexto constituem informações consideradas importantes, obtidas do mundo físico ou lógico por meio de sensores, tornando o contexto um instrumento de apoio à comunicação entre sistemas e/ou aplicações, denominados cientes de contexto, e seus usuários. Plataformas de middleware em IoT devem então ser responsáveis pela coleta, representação e processamento das informações de contexto providas por múltiplas fontes, liberando as aplicações e usuários da tarefa de manipulá-las e tornando transparente tal manipulação.

Nos primeiros dias de 2015, o mundo registrou 25 bilhões de dispositivos conectados à Internet segundo uma das maiores feiras de tecnologias do mundo, a CES 2015 (*Consumer Electronics Show 2015*). Como consequência deste crescente número de dispositivos sendo conectados, as plataforma de middleware para IoT devem atender ao requisito de **escalabilidade**, i.e., possuírem capacidade de suportar requisições de inúmeros dispositivos, funcionando corretamente, mesmo em situações de uso intenso. Soluções de nuvem vem sendo utilizadas para este fim, devido à sua facilidade de provisão e uso de recursos computacionais, que podem ser alocados e liberados

sob demanda, proporcionando o surgimento da chamada “Nuvem das Coisas” (em inglês, *Cloud of Things* - CoT).

Com o aumento do número de dispositivos presentes em um ambiente IoT, cresce também o volume de dados providos e transmitidos pela Internet. O que acarreta a necessidade de um **gerenciamento de grandes volumes de dados**, que constitui-se um outro requisito importante para plataformas de middleware IoT. Neste aspecto, surgem desafios para a persistência, consulta, indexação, processamento e manipulação de transações, que podem ser realizadas na própria plataforma de middleware ou em um banco de dados relacional externo a ela. Soluções baseadas em Big Data e Computação em Nuvem têm surgido como uma potencial resposta a alguns desses desafios.

Segurança e Privacidade são requisitos que também devem ser considerados na IoT, pois os dados trafegados provenientes destes novos dispositivos podem ser privados e para tal a plataforma de middleware deverá fornecer estratégias de segurança, a fim de manter a integridade e privacidade dos dados disponibilizados, além de proteger tanto os dispositivos envolvidos quanto os recursos expostos à rede.

Considerando também a alta dinamicidade dos ambientes IoT, nos quais dispositivos podem tornar-se indisponíveis pelos mais diversos motivos (e.g., falha, capacidade energética, indisponibilidade de conexão à rede, mobilidade de usuário, etc.), as plataformas de middleware devem prover meios para que ocorra a **adaptação dinâmica**, garantindo assim a disponibilidade e qualidade das aplicações durante a sua execução. Esse requisito é especialmente crítico em determinadas aplicações como *health care*, devido ao fato que falhas ou degradação de parâmetros de qualidade podem acarretar algum risco aos pacientes monitorados.

Um fator comum em aplicações IoT é a inerente inteligência, que reflete na denominação dos seus domínios, como por exemplo *smart home, smart health care, smart agriculture* e outros. Sendo parte destas “*smart*” aplicações, os dispositivos podem coletar dados automaticamente, compartilhar informações entre si, e iniciar e executar serviços com o mínimo de intervenção humana. Um dos principais desafios em IoT está em como gerenciar e manter um grande número de dispositivos, fazendo-os reagir de forma inteligente às informações captadas. Com base neste cenário, algumas características são almejadas para infraestruturas IoT (LEE; KIM, 2010).

- Automatização: componente chave que implica que uma infraestrutura IoT deva suportar coleta de dados, processamento e inferência contextual de forma autônoma.

- Inteligência: dispositivos que possuam inteligência devem estar capacitados a operar de forma adaptativa a diferentes situações. A Ciência de Contexto vem se mostrando um componente chave para viabilizar sistemas inteligentes na IoT.

- Dinamicidade: um dispositivo pode se mover de um lugar a outro, necessitando que sua infraestrutura esteja apta para reconhecer esta mudança e consequentemente realizar a adaptação necessária baseada no seu ambiente.
- Zero Configuração: para suportar a fácil integração de dispositivos, recursos *plug and play* devem estar disponíveis, otimizando a gerência dos dispositivos e possibilitando o crescimento descentralizado de sistemas IoT.

Desta forma, plataformas de middleware que consigam endereçar os principais requisitos que constituem desafios na concepção de infraestruturas IoT, de forma a contemplar as suas características almejadas, atingem um importante passo para a plena implantação de cenários IoT.

2.2 Arquitetura REST: Conceitos e Características

REST (*Representational State Transfer*) é um termo que foi utilizado pela primeira vez por Roy Fielding, em sua tese de doutorado publicada no ano 2000 (FIELDING, 2000).

Enquanto estilo arquitetural de software o REST não descreve protocolos específicos, formatos de dados ou sequências de interação, apenas apresenta os princípios de arquitetura e componentes que levaram ao sucesso e ampla utilização de sistemas distribuídos, como o da *World Wide Web*.

O REST trabalha em conjunto com o protocolo HTTP, que é o principal protocolo utilizado hoje na Web, embora seja possível desenvolver um sistema de software baseado nas definições do estilo arquitetural REST sem usar HTTP e sem interagir com a Web. Também se torna possível projetar interfaces HTTP + XML que não condizem com os princípios REST de Fielding (DAL MORO; DORNELES; REBONATTO, 2009). Sistemas que seguem os princípios REST são referenciados como "*RESTful*".

2.2.1 Princípios do Estilo Arquitetural REST

Segundo (FIELDING, 2000), o estilo arquitetural REST é construído sobre determinados princípios básicos da Internet. Existem cinco princípios que devem ser utilizados para a criação de serviços REST.

- Tudo são recursos: um recurso é qualquer componente de uma aplicação que valha a pena ser unicamente identificável e que possa ser transmitido na rede através das suas representações (binárias ou textuais). Em REST o pensamento não está focado em arquivos, mas sim em recursos.
- Todos os recursos são identificáveis através de um identificador único: os identificadores são utilizados para permitir a fácil manipulação do recurso. Em REST, utilizam-se *Uniform Resource Identifiers* (URIs) para identificar recursos.

- Utilização de uma interface simples e uniforme: REST utiliza HTTP como protocolo base, sendo este um protocolo bem conhecido e aceito em todo o mundo.
- Comunicação efetuada através de representações: cada recurso pode possuir várias representações diferentes de si próprio. Quando se realiza uma determinada operação sobre um recurso, o que acontece na realidade é uma troca de representações desse recurso.
- Interações *stateless* (sem estado): todas as interações realizadas sobre um recurso são independentes, isto é, a comunicação deve ser feita sem o armazenamento de qualquer tipo de estado no servidor, ou seja, cada requisição do cliente para o servidor deve conter todas as informações necessárias para que ela seja entendida. Portanto, estados de sessão, quando necessários, devem ser totalmente mantidos no cliente.

Um dos pontos chave deste estilo arquitetural tem a ver com a utilização de URIs ou Identificador Uniforme de Recursos para identificação de recursos na web. Segundo este estilo, os serviços são abstraídos através de uma interface uniforme (HTTP e respetivos métodos) que fornece mecanismos para que as aplicações cliente possam escolher a melhor representação possível das respostas que recebem do serviço. Estes são alguns dos motivos que posicionam o estilo REST no contexto IoT para o desenvolvimento de uma arquitetura global e APIs para dispositivos inteligentes.

O protocolo de comunicação base associado ao estilo REST é, como referido anteriormente, o protocolo HTTP. A arquitetura base deste protocolo segue o modelo de comunicação cliente-servidor. Este modelo é amplamente utilizado na Internet e de fácil compreensão: um cliente que deseje consultar um determinado recurso envia ao servidor um pedido HTTP, ao qual o servidor responde enviando uma mensagem ao cliente com a respetiva resposta.

Em relação ao formato da resposta enviada pelo servidor ao cliente e uma vez que os dispositivos inteligentes possuem recursos limitados, a informação pode ser enviada ao cliente tomando a forma de um documento estruturado XML ou ainda a forma de um documento *JavaScript Object Notation* (JSON).

Por sua vez, caso um usuário final queira consultar informações relacionadas com um determinado recurso, é preferível obter essa informação por exemplo através de um *browser* que renderize a resposta dada pelo servidor na forma de uma página web. Uma vantagem inerente a estas várias representações reside na possibilidade de poderem ser interpretadas e processadas por máquinas e ainda por pessoas permitindo uma maior flexibilidade e eficiência no processo de comunicação.

Dados dinâmicos acerca do mundo real podem ser apresentados em páginas web e depois processados com o auxílio de ferramentas Web 2.0. Por exemplo, os dispositivos podem ser indexados como páginas web através das suas representações

para que os utilizadores possam aceder a estas páginas através de um *browser* com o auxílio de um motor de busca. Estes endereços podem também ser enviados para outros utilizadores (através de *email* por exemplo) além de também poderem ser adicionados aos favoritos do *browser* pelo utilizador final. A ideia base passa pela utilização da web como sistema de informação descentralizado para que seja fácil expor novos serviços e aplicações, direta ou indiretamente por dispositivos inteligentes.

A ideia central de REST está intimamente relacionada com a noção de recurso. Cada recurso possui um identificador global (URI em HTTP). Estes recursos são acedidos por componentes na rede que comunicam através de um protocolo (ex. HTTP) e trocam conteúdo (representações) desses recursos. Para interagir com um recurso, uma aplicação tem que possuir o identificador do recurso bem como o método que quer aplicar sobre o mesmo.

Portanto no REST, não é necessário conhecer a implementação e a configuração do sistema, ou seja, não interessa saber se existem *proxies, firewalls* ou qualquer outro componente entre a aplicação e o servidor que aloja esses recursos. A aplicação deve ser capaz de interpretar os dados enviados (como resposta) pelo servidor, sejam eles texto simples, XML, imagens, documentos, etc. Para isso, é possível a qualquer cliente especificar no pedido qual o formato da representação que deseja receber por parte do servidor.

De forma simples, o primeiro passo para permitir ligar dispositivos inteligentes na web passa por criar a rede de recursos. Dois aspetos centrais são o identificador de um recurso e as suas relações com outros recursos.

Em REST, a interação com os recursos e a recepção das representações por parte dos clientes acontece através de uma interface uniforme que especifica um contrato de serviço entre clientes e servidores. Existem 3 partes distintas e fundamentais para que se realize esta interação: operações, negociação de conteúdo e estado da resposta.

Quanto às operações, REST utiliza quatro métodos principais do protocolo HTTP para permitir a interação com recursos, sendo estes muitas vezes conhecidos como verbos HTTP: GET, PUT, POST e DELETE. O método GET é utilizado para obtenção de representações de recursos. O verbo PUT serve para atualizar o estado de um recurso ou para criar um recurso utilizando um identificador. O método POST cria um novo recurso sem ser necessário especificar o identificador. Por fim, o método DELETE serve para remover um recurso. Um exemplo do uso das operações através de métodos HTTP identificadas por URIs pode ser visto na Tabela 1.

Em relação à negociação de conteúdo, isto é, ao mecanismo que permite aos clientes e servidores comunicarem para chegarem a acordo sobre qual a representação a devolver pelo servidor, esta negociação está embutida na interface HTTP subjacente à arquitetura REST. No que diz respeito ao estado da resposta, são utilizados os códigos *standard* do protocolo HTTP. Existem vários códigos disponíveis (200 representa uma

Tabela 1: Interações com Recursos Através de Métodos HTTP

URI	Método HTTP	Descrição
/user	GET	Lista de usuários.
/user/id	GET	Recupera um usuário com todas as suas informações.
/user/id	PUT	Altera os dados do usuário informado.
/user/id	DELETE	Remove o usuário informado.
/user/	POST	Cria um novo usuário.

operação efetuada com sucesso enquanto que o 405 significa que o método utilizado não é permitido pelo recurso referenciado pela URI).

A flexibilidade e facilidade de compreensão do estilo arquitetural REST, torna-o um estilo atrativo e de ampla utilização no cenário da IoT.

2.3 Protocolos de Comunicação na IoT

Os protocolos de comunicação são parte importante na implantação de infraestruturas IoT, podendo contribuir para uma melhor utilização de recursos como os de processamento, memória, energia e banda de comunicação dos dispositivos inteligentes, tendo em vista fatores inerentes do universo IoT como heterogeneidade, mobilidade, intermitência de comunicação, extensibilidade, escalabilidade e capacidade de autoconfiguração.

Quanto aos protocolos de comunicação na IoT este trabalho considerou protocolos de duas naturezas, que se destacaram na literatura. O UPnP enquanto uma abordagem que provê funcionalidades que minimizam os esforços de gerência dos dispositivos ou objetos inteligentes conectados, como a ativação, desativação e descoberta destes dispositivos e seus recursos por parte das aplicações. A outra abordagem considerou o CoAP e MQTT como alternativa de protocolos que objetivam o baixo consumo computacional e energético dos dispositivos introduzidos pelo cenário da IoT, que se caracterizam por possuírem capacidade restrita quanto a esses quesitos.

2.3.1 UPnP

As características de autoconfiguração e de descoberta automática de dispositivos inerentes ao UPnP (*Universal Plug and Play*), associado ao fato deste utilizar protocolos e padrões consolidados da Internet, tais como HTTP, SOAP e XML constituem um facilitador na gerência do crescente número de dispositivos conectados, o que traduz o fato deste ser bastante utilizado no cenário da IoT. Outro fator interessante é a quantidade significativa de dispositivos comerciais com a certificação DLNA (*Digital Living Network Alliance*) (DLNA, 2015) que tem como base o protocolo UPnP para comunicação.

O UPnP é uma arquitetura proposta inicialmente pela Microsoft que está sendo mantida atualmente por um fórum do qual participam centenas de fabricantes (UPNP, 2015). Este permite a interoperabilidade entre diferentes dispositivos computacionais, em uma rede baseada no modelo *Plug and Play*, que facilita a instalação e configuração de dispositivos heterogêneos numa rede IP, sem a necessidade de configurá-los para possibilitar os processos de obtenção de endereços IP e de descoberta e invocação de serviços de rede.

Sua arquitetura de software oferece uma conexão de rede denominada *Peer-to-Peer* (P2P), onde cada um dos pontos ou nós da rede funciona tanto como cliente quanto como servidor.

Estão definidos no padrão UPnP três componentes básicos.

- **Dispositivo UPnP**: contêm serviços UPnP e podem conter outros dispositivos UPnP aninhados. Por exemplo, uma impressora (dispositivo UPnP) pode consistir em um serviço de impressão e um dispositivo scanner aninhado, que por sua vez oferece um serviço de fotocópia.

- **Serviço UPnP**: expõe ações que podem ser aplicadas ao mesmo durante sua invocação por meio de um servidor de controle residente no dispositivo que hospeda o serviço, bem como uma tabela que armazena um conjunto de variáveis de estado do serviço (tabela de estados). A tabela de estados do serviço UPnP pode ser monitorada por um servidor de eventos (também residente no dispositivo que hospeda o serviço), que tem como função publicar a outras entidades interessadas a modificação de variáveis de estado desse serviço.

- **Ponto de Controle UPnP**: atua, em parte, como servidor de diretório, tendo como tarefa descobrir e controlar os dispositivos UPnP presentes na rede.

No caso de um dispositivo não possuir capacidade de portar o padrão UPnP, um gateway UPnP deve ser usado para mapear os protocolos definidos pelo UPnP nos protocolos entendidos nativamente pelo dispositivo.

O UPnP permite o controle de dispositivos e o processamento de serviços de forma descentralizada, e permite a interoperabilidade entre pontos de controle. Além disso, a tecnologia UPnP inclui o conceito de descoberta e de descrição de dispositivos e serviços, permitindo assim que um dispositivo possa ser dinamicamente encontrado e totalmente compreendido, em termos de funcionalidade, por descrições XML.

2.3.1.1 ***Pilha de Protocolos UPnP***

A pilha de protocolos UPnP é constituída de protocolos específicos localizados nas camadas de mais alto nível (três camadas superiores), e por protocolos padrões da Internet (vide Figura 3).

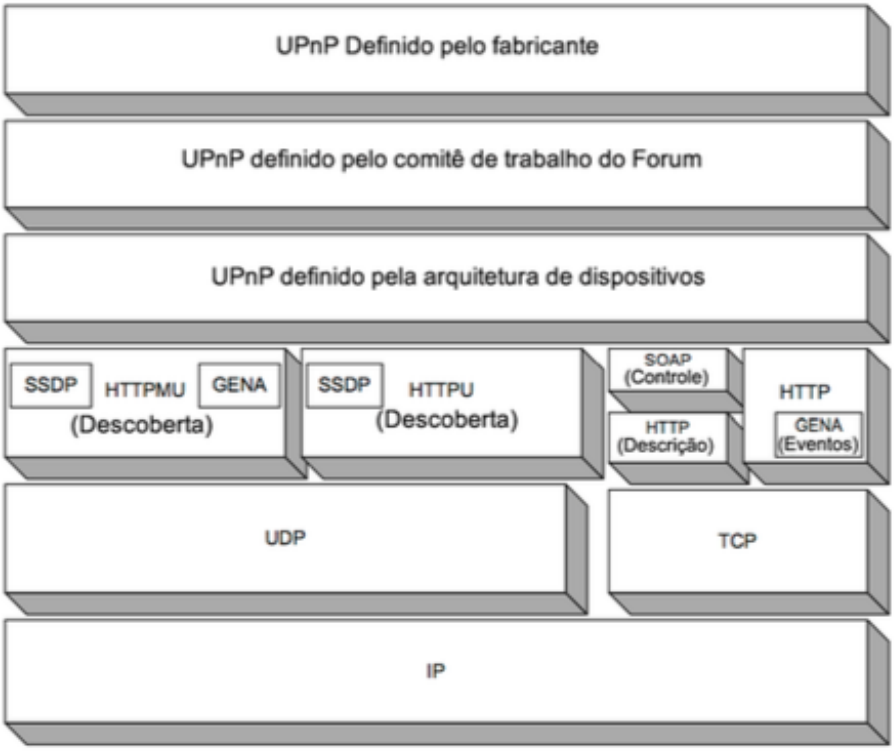

Figura 3: Pilha de Protocolos UPnP. Fonte: (FERREIRA, 2014)

Protocolos Específicos UPnP

Os fabricantes de dispositivos UPnP, os comitês de trabalho do fórum UPnP e o documento da arquitetura de dispositivo UPnP definem os protocolos das camadas superiores utilizados para implementar o UPnP.

A arquitetura de dispositivo UPnP, do inglês *UPnP Device Architecture* (UDA) define um esquema ou modelo para a criação de descrições de dispositivo e serviço. Baseado nesta arquitetura os comitês de trabalho definem especificações para tipos de dispositivo como videocassetes (VCRs), sistemas HVAC, lavadoras de pratos e outros aparelhos. Subsequentemente, os fabricantes de dispositivos UPnP adicionam dados específicos aos seus dispositivos como o nome do dispositivo, número do modelo, nome do fabricante e URL para a descrição do serviço.

TCP/IP

A pilha de protocolos de rede TCP/IP serve como base para a construção dos demais protocolos UPnP. Dispositivos UPnP podem usar diversos protocolos na pilha TCP/IP, incluindo TCP, UDP, IGMP, ARP e IP, além de serviços TCP/IP como DHCP e DNS.

HTTP, HTTPU, HTTPMU

O HTTP, que é o principal responsável pelo sucesso da Internet, é também parte central do UPnP. Todos os aspectos do UPnP são construídos sobre o HTTP ou suas variantes.

O HTTPU e o HTTPMU são variantes do HTTP definidas para entregar mensagens sobre UDP/IP em vez de TCP/IP. Esses protocolos são utilizados pelo SSDP (*Simple Service Discovery Protocol*). Os formatos básicos de mensagem usados por esses protocolos correspondem aos do HTTP, sendo que o HTTPMU é utilizado para difusão seletiva (multicast) e o HTTPU para entrega ponto a ponto (unicast).

SSDP - Simple Service Discovery Protocol

O Protocolo de Descoberta de Serviço Simples (SSDP) define como os serviços de rede podem ser descobertos na rede. Baseia-se em HTTPU e HTTPMU e define métodos para que um ponto de controle localize recursos de interesse na rede, e para que dispositivos anunciem sua disponibilidade na rede, diminuindo a sobrecarga na comunicação e agilizando o trabalho decorrente da existência de mais de um ponto de controle na rede.

Além da capacidade de descoberta, o protocolo SSDP possui a capacidade de Notificação de Despedida (desconexão da rede), liberando os dispositivos e serviços indesejados ou ociosos.

GENA - Generic Event Notification Architecture

A Arquitetura de Notificação de Evento Genérico (GENA) foi definida para fornecer a capacidade de envio e recebimento de notificações utilizando HTTP sobre TCP/IP e HTTPMU sobre UDP/IP.

SOAP - Simple Object Access Protocol

O Protocolo de Acesso a Objeto Simples (SOAP) define o uso da Linguagem de Marcação Extensível (XML) e do HTTP para executar chamadas de procedimento remoto.

O UPnP utiliza SOAP para entregar mensagens de controle aos dispositivos e retornar resultados ou erros aos pontos de controle. Cada solicitação de controle de UPnP é uma mensagem SOAP que contém a ação da solicitação juntamente com um conjunto de parâmetros, já a resposta é uma mensagem SOAP contendo o status, o valor de retorno e demais parâmetros de retorno.

2.3.1.2 ***Etapas envolvidas na rede UPnP***

Para que pontos de controle e dispositivos UPnP interajam entre si, a arquitetura de dispositivos UPnP (UDA) define as seguintes etapas:

1. **Endereçamento**: determina um endereço IP para uma entidade UPnP através de DHCP ou auto IP.

2. **Descoberta**: anuncia o dispositivo no UPnP, através do protocolo SSDP, podendo este ser localizado pelos pontos de controle que já estão em execução. A descoberta ocorre assim que os dispositivos forem conectados à rede e endereçados adequadamente. Quando um dispositivo é adicionado à rede, o SSDP permite que ele anuncie seus serviços aos pontos de controle na rede. Quando um ponto de controle é adicionado à rede, o SSDP permite que ele procure dispositivos de interesse na rede. A troca básica em ambos os casos é uma mensagem de descoberta contendo algumas especificações essenciais sobre o dispositivo ou um de seus serviços, por exemplo, seu tipo, identificador e ponteiro para o respectivo documento de descrição de dispositivo XML.

3. **Descrição**: os pontos de controle interessados podem solicitar arquivos XML contendo informações do dispositivo (Documento de Descrição do Dispositivo) e de seus serviços (Documentos de Descrição de Serviços), através de uma requisição HTTP. Após a descoberta, o ponto de controle ainda sabe muito pouco sobre o dispositivo. Para obter mais informações sobre o dispositivo e seus recursos, ou para interagir com ele, o ponto de controle deve recuperar a sua descrição na URL fornecida pelo dispositivo na mensagem de descoberta. A descrição do UPnP em XML para um dispositivo inclui informações de fabricante específicas do fornecedor incluindo o nome e número do modelo, número de série, nome do fabricante, URLs de sites específicos do fornecedor e assim por diante. A descrição também inclui uma lista de todos os dispositivos ou serviços incorporados, assim como URLs para controle, eventos e apresentação.

4. **Controle**: depois de recuperar uma descrição do dispositivo, o ponto de controle terá as informações básicas para o seu controle. Para saber mais sobre o serviço, o ponto de controle deve recuperar uma descrição detalhada do UPnP para cada serviço. A descrição para o serviço também é expressa em XML e inclui uma lista de comandos, ou ações, aos quais o serviço responde, além de parâmetros ou argumentos para cada ação. A descrição de um serviço inclui também uma lista de variáveis que modelam o estado do serviço na execução e são descritas em termos do tipo de dados, do intervalo e das características do evento. Para controlar um dispositivo, um ponto de controle envia uma solicitação de ação a um serviço de dispositivo. Para isso este envia uma mensagem XML de controle, através do protocolo SOAP, adequada a URL de controle do serviço (fornecido na descrição do dispositivo). Em resposta à mensagem de controle, o serviço retorna valores ou códigos de falha específicos da ação.

5. **Evento**: paralelo ao controle, a fase de Evento permite que pontos de controle se cadastrem nos dispositivos para receberem notificações de eventos, que ocorrem quando as variáveis de estado do dispositivo são modificadas. Uma descrição UPnP para o serviço inclui uma lista de ações às quais o serviço responde e uma lista de variáveis que modelam o estado do serviço na execução. O serviço publica atualizações quando essas variáveis mudam, e um ponto de controle pode inscrever-se para receber essas informações. Estas atualizações são enviadas através de mensagens de evento expressas em XML e formatadas usando o protocolo GENA, que contêm os nomes de uma ou mais variáveis de estado e o valor atual das variáveis.

6. **Apresentação**: se um dispositivo possuir uma URL de apresentação, então o ponto de controle pode recuperar uma página dessa URL, carregá-la em um navegador e, dependendo dos recursos da página, permitir que um usuário controle o dispositivo e/ou exiba seu status. O grau de realização dessas etapas depende dos recursos específicos da página de apresentação e do dispositivo.

2.3.2 CoAP

O CoAP (*Constrained Application Protocol*) é um protocolo que recentemente tournou-se um *Request for Comments* (RFC - 7252) (SHELBY; HARTKE; BORMANN, 2015), desenvolvido pelo grupo de trabalho CoRE (*Constrained RESTful Environment*) do IETF (*Internet Engineering Task Force*), com o objetivo de se ter um protocolo web genérico, alternativo ao HTTP, adequado aos requisitos especiais do ambiente introduzido pela IoT e caracterizado por dispositivos de baixa potência, considerando principalmente, a questão do consumo de energia.

O protocolo CoAP baseia-se em princípios arquiteturais REST (FIELDING, 2000). Um dos pontos chave deste estilo arquitetural tem a ver com a utilização de URIs (*Uniform Resource Identifier*) para identificação de recursos na web. Outra característica do estilo é que os serviços são abstraídos através de uma interface uniforme (HTTP e seus respectivos métodos) que fornece mecanismos para que as aplicações cliente possam escolher a melhor representação possível das respostas que recebem do serviço.

O diferencial do CoAP é que este permite a execução das funções HTTP em redes com recursos limitados, reduzindo o *overhead* original do HTTP ao comprimir o cabeçalho da camada de aplicação para apenas 4 bytes (excluindo os campos opcionais, que ocupam 4 bits cada um).

2.3.2.1 ***Características do CoAP***

- UDP nativo: utiliza o UDP (*User Datagram Protocol*) na camada de transporte em vez de TCP (*Transmission Control Protocol*). O UDP é um protocolo não orientado a conexão, que permite uma maior rapidez de *wake up* e transmissão de ciclos, bem como de pacotes com menos sobrecarga. Isso permite que dispositivos possam permanecer em estado de repouso por longo período de tempo conservando a energia da bateria.

- Suporte a *Multicast*: a rede é inerentemente *unicast* porém possui suporte *multicast*. Isso é devido ao CoAP ser construído em cima de IPv6, o qual permite o endereçamento *multicast* para os dispositivos, além de seus endereços IPv6 normais.

- Segurança: por utilizar o protocolo UDP, não é possibilitado o uso por parte do CoAP da tecnologia SSL/TLS (*Secure Sockets Layer/Transport Layer Security*), no entanto este utiliza o DTLS (*Datagram Transport Layer Security*), que permite a mesma segurança do TLS. Como TCP, UDP é criptografado, mas a segurança pode ser - e deve ser - ampliada com DTLS.

- Descoberta de Recurso/Serviço: os servidores CoAP fornecem uma listagem completa dos seus recursos e isto permite aos clientes saberem quais os recursos podem utilizar, bem como o seu tipo. Todos os recursos são descritos através de URIs, um conjunto de atributos e, se necessário, relações com outros recursos.

- Comunicação Assíncrona: a maioria das mensagens são enviadas e recebidas usando o modelo *request/response*. No entanto, o CoAP tem um mecanismo simplificado chamado "*observe*" semelhante ao *publish/subscribe*, no qual o cliente indica que pretende receber as atualizações do recurso sempre que o seu estado for alterado.

2.3.2.2 ***Mensagem CoAP***

CoAP é baseado na troca de mensagens compactas com características que intuem o aumento na confiabilidade da sua transmissão. As mensagens CoAP podem ser de quatro tipos, descritos a seguir (SHELBY; HARTKE; BORMANN, 2015).

- *Confirmable* (CON): este tipo de mensagem oferece confiabilidade sobre o protocolo UDP. Sempre que é enviada uma mensagem deste tipo ao servidor, ela segue com um *time out* associado. Quando o servidor recebe a mensagem, envia um ACK com o mesmo identificador da mensagem do pedido, confirmando a recepção do mesmo. Quando o cliente não recebe um ACK passado o tempo

definido no *time out*, este retransmite a mensagem e o *time out* enviado nesta retransmissão é duplicado para garantir que a entrega do pacote seja efetuada. Estas retransmissões são efetuadas até ser recebido um ACK com o mesmo identificador de mensagem por parte do servidor.

- *Non-confirmable* (NON): não necessita de confirmação de recebimento. O cliente não tem como saber se o pedido chegou ao servidor. Como alternativa, o cliente pode enviar múltiplos pedidos. Esta característica é útil no caso de uma aplicação que recebe leituras constantes de um sensor de temperatura em um espaço muito curto de tempo, onde a perda de uma ou outra mensagem não é motivo para preocupações.

- *Acknowledgment* (ACK): são mensagens que confirmam o recebimento de uma mensagem *Confirmable*. Caso o servidor necessite enviar alguma informação no *Payload* da mensagem para o cliente, esta pode ser enviada juntamente com o ACK. Se o ACK e a informação da resposta forem enviados em mensagens separadas, o ID destas mensagens será diferente, visto que, para cada retransmissão é necessário um novo identificador. Quando o cliente receber a informação, tem que enviar um ACK, indicando que recebeu a resposta. Neste caso, o pedido e a resposta são identificados devido à utilização da opção *Token*.

- *Reset* (RST): indica que outra mensagem (CON ou NON) foi recebida, mas por algum imprevisto não pôde ser devidamente processada. Ela pode ocorrer no caso de algum dispositivo ter reiniciado e a mensagem enviada não foi interrompida.

Os tipos descritos podem ser usados em mensagens de Requisição (*Request*), Resposta (*Response*) e Vazia (*Empty*). A mensagem *Empty* é caracterizada pelo código 0.00 e não apresenta nenhuma informação adicional, apenas o cabeçalho de 4 bytes. A relação de utilização do tipo de mensagem por parte destas mensagens pode ser observada na Tabela 2.

Os caracteres descritos na tabela possuem o seguinte significado: “X” representa que o tipo de mensagem pode ser utilizado e “-” que este tipo de mensagem não pode ser considerado. O caractere “*” significa que a combinação não é utilizada em condições normais de operação, mas apenas para provocar uma mensagem de *Reset*, constituindo-se assim uma forma de verificar se o dispositivo CoAP está ativo (*Ping* CoAP).

Estas mensagens são transmitidas via IEEE 802.15.4 e apresentam-se codificadas em um formato binário com um cabeçalho de 4 bytes, seguido por um *Token* de largura variável (de 0 a 8 bytes). Após o Token pode não haver nenhuma ou uma série

Tabela 2: Utilização dos Tipos de Mensagens CoAP

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

Fonte: (SHELBY; HARTKE; BORMANN, 2015)

de opções do CoAP no formato *Type-Length-Value* (TLV), que são opcionalmente seguidas por um *Payload*, ocupando o resto do datagrama. A Figura 4 mostra o formato da mensagem CoAP.

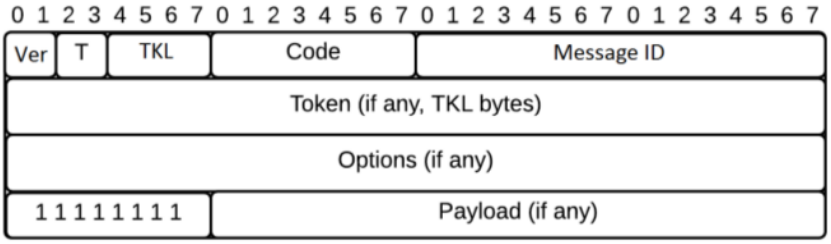

Figura 4: Formato da Mensagem CoAP. Fonte: (LIMA, 2014)

O *Token* é utilizado para relacionar requisições e respostas. Ele é gerado no cliente e transportado junto com a requisição. O servidor deve então ecoar este Token na resposta correspondente.

Já o *Payload* é um campo opcional. Quando a mensagem possui Payload ele deve ser indicado por um marcador de um byte (*Payload Marker*) que indica o fim das opções e o início da carga. A ausência do marcador significa um Payload de largura 0 bytes, enquanto a presença de um marcador seguido por um Payload de 0 bytes deve ser processado como um erro no formato da mensagem.

Os campos que formam o cabeçalho do datagrama são definidos pelo RFC 7252 - IETF (SHELBY; HARTKE; BORMANN, 2015) da seguinte maneira.

- Versão (Ver): inteiro não assinado de dois bits que indica o número correspondente à versão do CoAP.
- Tipo (T): inteiro não assinado de dois bits que indica o tipo da mensagem, que pode variar entre *Confirmable* (0), *Non-confirmable* (1), *Acknowlegment* (2) ou *Reset* (3).
- Largura do *Token* (TKL): inteiro não assinado de quatro bits usado para indicar a largura variável do Token (até 8 bytes).
- Código: inteiro não assinado de oito bits que caracteriza o tipo da mensagem,

podendo indicar um método de requisição (vide Tabela 3) ou um código de resposta (vide Tabela 4).

- ID da mensagem: inteiro não assinado de 16 bits usado para detectar duplicação de mensagens e também para comparação de mensagens do tipo ACK, por exemplo.

Tabela 3: Códigos dos Métodos de Requisição CoAP

Código	Nome
0.01	GET
0.02	POST
0.03	PUT
0.04	DELETE

Fonte: (SHELBY; HARTKE; BORMANN, 2015)

Tabela 4: Códigos de Resposta CoAP

Código	Nome
2.01	Criado
2.02	Deletado
2.03	Válido
2.04	Alterado
2.05	Conteúdo
4.00	*Bad Request*
4.01	Não autorizado
4.02	*Bad Option*
4.03	Proibido
4.04	Não encontrado
4.05	Método não permitido
4.06	Inaceitável
4.12	Pré-condição falhou
4.13	Entidade de requisição muito grande
4.15	*Content-format* não suportado
5.00	Erro interno no servidor
5.01	Não implementado
5.02	*Bad gateway*
5.03	Serviço indisponível
5.04	*Gateway time out*
5.05	*Proxying* não suportado

Fonte: (SHELBY; HARTKE; BORMANN, 2015)

2.3.3 MQTT

O MQTT (*Message Queuing Telemetry Transport*) foi desenvolvido por Andy Stanford-Clark, da IBM, e Arlen Nipper da Arcom (agora Eurotech), em 1999 (MQTT,

2015). Trata-se de um protocolo de mensagens baseado na arquitetura *publish/subscribe* voltado a atuar principalmente, mas não exclusivamente, em ambientes onde há redes instáveis com baixa largura de banda, bem como dispositivos embarcados com recursos limitados de memória e processamento (MARTINS; ZEM, 2015).

Em seu desenvolvimento foram considerados alguns princípios, como o de minimizar requerimentos de recursos do dispositivo, bem como o de largura de banda tentando assegurar confiabilidade e garantia de entrega, o que justifica a sua ampla utilização no cenário da IoT. O MQTT tornou-se um padrão aberto OASIS (*Organization for the Advancement of Structured Information Standards*) em 07 de novembro de 2014, em sua versão 3.1.1 (OASIS-MQTT, 2015).

O protocolo segue o modelo cliente/servidor. Os dispositivos sensores são clientes que se conectam a um servidor (chamado de *broker*) usando TCP. As mensagens a serem transmitidas são publicadas para um endereço (chamado de tópico), que inclusive, assemelha-se a uma estrutura de diretórios em um sistema de arquivos, por exemplo, casa/quarto2/temperatura. Clientes por sua vez podem se subscrever para vários tópicos, tornando-se assim capazes de receber as mensagens que outros clientes publicam neste tópico.

2.3.3.1 ***Características do MQTT***

- Arquitetura *Publish/Subscribe*: *brokers* e clientes publicam informações, como também podem se subscrever para receber informações de outros, de acordo com o conteúdo da mensagem, tipo ou assunto.

- Desacoplamento de Espaço: o cliente não necessita ter conhecimento do endereço IP de outros clientes para publicar ou receber informações destes, basta conhecer o IP do *broker*. O *broker* então fica responsável por intermediar a comunicação, gerenciando o fluxo de informações aos clientes.

- Desacoplamento de Tempo: o cliente pode publicar a sua informação, independentemente do estado que encontra-se o cliente alvo. As publicações são transmitidas ao *broker* que repassa ao cliente correspondente quando este estiver ativo. Isso permite que clientes permaneçam em estado de repouso, mesmo quando outros clientes estão publicando mensagens relevantes para eles.

- Segurança: *brokers* MQTT podem exigir nome de usuário e senha para autenticação de clientes, possibilitando a estes um controle de nível de acesso as informações pelos clientes. A privacidade é garantida pela transmissão dos dados criptografada via TCP, mediante o uso de SSL/TLS.

- Níveis de *Quality of Services* (QoS): possui três níveis de QoS (0, 1 e 2) que

representam a garantia de qualidade de entrega das mensagens (vide Tabela 6).

- Aviso de Desconexão: possui mecanismo que notifica as partes interessadas quando um cliente se desconecta da rede anormalmente.
- Tópicos de Subscrição Flexíveis: um cliente pode se subscrever a todas as mensagens dentro de uma determinada funcionalidade. Por exemplo em uma cozinha o dispositivo cliente forno pode receber todas as informações referentes ao tópico cozinha/forno/+, com o código "+" como coringa. Isto permite uma quantidade mínima de código (ou seja, memória e custo).
- MQTT-SN: possui uma variante do protocolo MQTT para *Sensors Networks* (Redes de Sensores). Define um mapeamento UDP de MQTT, voltado para redes não TCP/IP como *Zigbee*, além de adicionar suporte ao *broker* para indexação de nomes de tópicos. Isto é devido a necessidade da sequência destes nomes não poder ser longa em redes de sensores sem fio com baixa taxa de transmissão.

2.3.3.2 ***Mensagem MQTT***

Cada uma das chamadas mensagens de comando MQTT possui um cabeçalho fixo composto de dois bytes, onde o primeiro byte contém o campo que identifica o tipo da mensagem e também campos marcadores (DUP, Nível QoS e *RETAIN*). A Figura 5 mostra o formato do cabeçalho fixo das mensagens.

bit	7	6	5	4	3	2	1	0
Byte 1	Tipo da Mensagem				Flag DUP	Nível QoS		RETAIN
Byte 2	Largura restante							

Figura 5: Cabeçalho Fixo de uma Mensagem MQTT. Fonte: (MARTINS; ZEM, 2015)

O tipo da mensagem ocupa os 4 bits mais significativos no primeiro byte do cabeçalho fixo e é representado por um valor não assinado. A lista de tipos definida na versão 3.1.1 do MQTT (BANKS; GUPTA, 2014) é descrita na Tabela 5.

Os quatro bits restantes do primeiro byte dividem-se em quatro campos que servem de marcadores para indicar preferências definidas antes do envio da mensagem. São eles:

- *Duplicate delivery* (DUP): acrônimo relativo à entrega duplicada, este marcador ocupa o bit 3 e é ativado quando o cliente ou o servidor tentam reenviar mensagens do tipo *PUBLISH, PUBREL, SUBSCRIBE ou UNSUBSCRIBE* (vide Tabela 5) que tenham *Quality of Service* (QoS) maior que 0 e requeiram *Acknowlegment* (ACK);

- *Quality of Service* (QoS): este marcador ocupa os dois penúltimos bits menos significativos e indica o nível de garantia da entrega de uma mensagem *PUBLISH*. Os níveis de QoS são mostrados na Tabela 6;

- *RETAIN*: quando um cliente envia uma mensagem *PUBLISH* ao servidor com este marcador ativo, ela deve ser retida no servidor mesmo depois de ser entregue aos assinantes. No evento de uma nova subscrição a um tópico, a última mensagem retida para este tópico deve ser enviada para o novo assinante caso este marcador esteja ativado.

Tabela 5: Tipos de Mensagem de Controle

Mnemônico	Enumeração	Descrição
Reservado	0	Reservado
CONNECT	1	Requisição de conexão do cliente ao servidor
CONNACK	2	ACK de conexão
PUBLISH	3	Mensagem de Publicação
PUBACK	4	ACK de publicação
PUBREC	5	Publicação recebida (garantia de entrega parte I)
PUBREL	6	Publicação liberada (garantia de entrega parte II)
PUBCOMP	7	Publicação completa (garantia de entrega parte III)
SUBSCRIBE	8	Requisição de subscrição do cliente
SUBACK	9	ACK de subscrição
UNSUBSCRIBE	10	Requisição de cancelamento de subscrição do cliente
UNSUBACK	11	ACK de cancelamento de subscrição
PINGREQ	12	Requisição PING
PINGRESP	13	Resposta PING
DISCONNECT	14	Cliente desconectando
Reservado	15	Reservado

Fonte: (OASIS-MQTT, 2015)

Tabela 6: Níveis de QoS

Valor QoS	Bit 2	Bit 1	Descrição		
0	0	0	Até uma vez	Disparar e esquecer	≤ 1
1	0	1	Ao menos uma vez	Entrega com ACK	≥ 1
2	1	0		Entrega garantida	1
3	1	1	Reservado		

Fonte: (OASIS-MQTT, 2015)

O segundo byte do cabeçalho fixo é usado para representar a quantidade de bytes remanescentes na mensagem. Incluindo dados do cabeçalho variável e do *payload*.

Cabeçalho variável é um componente presente em alguns tipos de mensagem MQTT e está localizado entre o cabeçalho fixo e o *payload*. Usado principalmente nas mensagens CONNECT, este cabeçalho possui dois campos para nome e versão

do protocolo e mais uma série de marcadores que definirão algumas diretivas para a conexão entre cliente e servidor.

O *payload* pode armazenar diferentes tipos de informação, tudo vai depender do tipo da mensagem transmitida: CONNECT (irá conter ID do cliente), SUBSCRIBE (contém tópicos que o cliente pode subscrever e/ou nível QoS) ou SUBACK (lista de níveis de QoS garantidos pelo servidor).

3 PLATAFORMAS DE MIDDLEWARE PARA A IOT

Neste capítulo serão apresentados quatro trabalhos que propuseram plataformas de middlewares para IoT. A descrição de cada trabalho buscou caracterizar a sua arquitetura de software, bem como as abordagens utilizadas para interoperação entre os dispositivos da IoT. Ao final foi realizada uma análise, levando em consideração os seguintes requisitos: (i)interoperabilidade, (ii) descoberta e gerenciamento de dispositivos, (iii) interfaces de alto nível, (iv) Ciência de Contexto e (v) escalabilidade.

3.1 EcoDiF - Ecossistema Web de Dispositivos Físicos

A plataforma EcoDiF (DELICATO et al., 2013) (PIRES et al., 2014) integra dispositivos físicos heterogêneos e os conecta à Internet, fornecendo funcionalidades de controle, visualização, processamento e armazenamento de dados em tempo real. Esta foi desenvolvida como uma proposta de solução a alguns dos desafios IoT, tais como: (i) a necessidade de uma camada de abstração sobre dispositivos e serviços a aplicações e usuários finais; (ii) o fornecimento de serviços de busca e descoberta desses elementos; (iii) a interconexão de objetos e serviços via rede; (iv) o monitoramento do estado e da localização dos objetos conectados, e; (v) o gerenciamento da interoperabilidade entre os objetos envolvidos.

Uma característica dessa plataforma refere-se ao fato dela se alinhar a um outro paradigma surgido a partir do termo Internet das Coisas, denominado Web das Coisas, do inglês *Web of Things* (WoT). Este paradigma se caracteriza pelo desenvolvimento de aplicações que utilizam protocolos e padrões amplamente aceitos e já em uso na Web tradicional, tais como HTTP (*Hypertext Transfer Protocol*) e URIs (*Uniform Resource Identifier*) (GUINARD, 2010).

No caso o HTTP não é utilizado apenas como um protocolo de comunicação, mas também como uma maneira de oferecer suporte a todas as interações com os dispositivos interconectados, eliminando barreiras no tocante à incompatibilidade entre diferentes fabricantes, protocolos proprietários e formatos de dados. Assim dispositivos físicos são integrados ao ambiente digital, de maneira que seus dados e serviços

possam ser acessados por diferentes aplicações como um recurso qualquer da Web.

A EcoDiF foi implementada utilizando a linguagem de programação Java e implantada em um servidor de aplicações JBoss, que permite o gerenciamento de componentes distribuídos, promove a confiabilidade de aplicações baseadas em um conjunto de serviços acessíveis via Web e em grandes volumes de dados, como é típico em ambientes de IoT. Os dados são recebidos por requisições HTTP PUT, de forma a prover uma interface RESTful para os clientes (sejam humanos ou aplicações).

A arquitetura de software da EcoDiF, ilustrada pela Figura 6, é composta por sete módulos descritos a seguir.

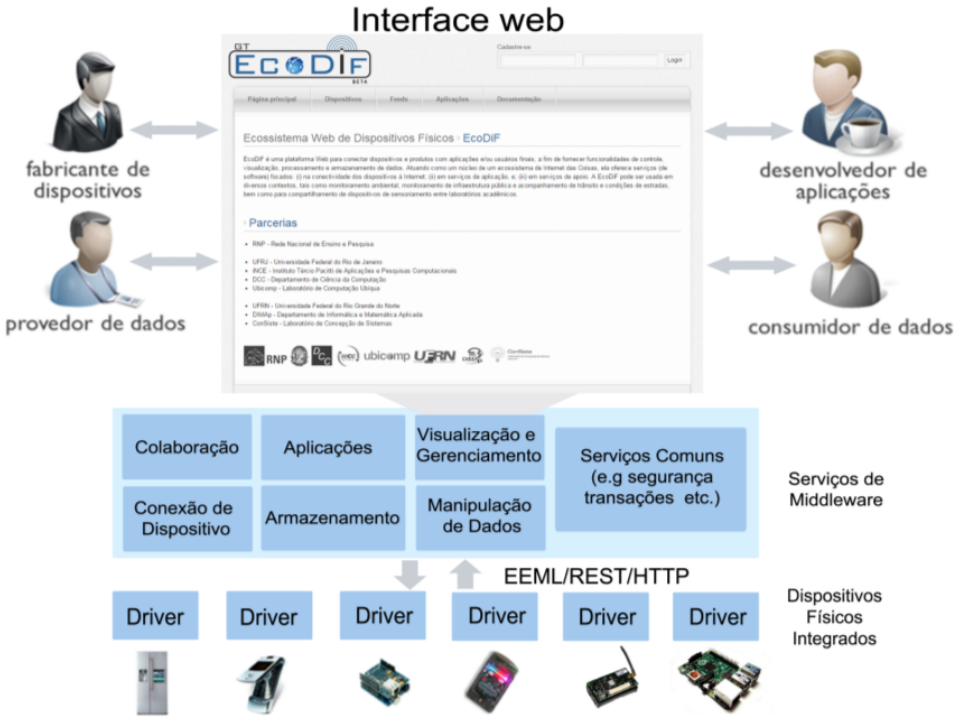

Figura 6: Arquitetura EcoDiF. Fonte: (PIRES et al., 2014)

- **Módulo de Colaboração**: visa facilitar a colaboração entre os usuários da EcoDiF, permitindo realizar buscas por dispositivos e aplicações a partir de seus respectivos metadados (tipo, usuário, localização, etc.), através da interface Web.
- **Módulo de Conexão de Dispositivos**: tem como objetivo viabilizar a conexão de dispositivos físicos à plataforma. A forma encontrada para a integração dos dispositivos foi por meio da utilização de *drivers* customizados, desenvolvidos para cada tipo de dispositivo. Dessa forma a heterogeneidade é abstraída de usuários e aplicações, podendo estes fazerem uso dos dados gerados por estes dispositivos. Os *drivers* são construídos com base em princípios REST e padrões e protocolo Web. Além disso os *drivers* também são responsáveis por

estruturar os dados utilizando a linguagem EEML, para após serem enviados à EcoDiF através de requisições HTTP PUT.

- **Módulo de Aplicações**: fornece um modelo e um ambiente para programação e execução de aplicações que fazem uso dos dados disponibilizados pela EcoDiF, chamados de *feeds*. Estes *feeds* podem gerar novas informações, ficando estas disponíveis para serem usadas por outros *feeds* e aplicações mais complexas. Essas aplicações são construídas possibilitando o uso de *mashups*, pois podem ser criadas através da composição de diferentes tipos de informação, providas por diversas fontes, tais como serviços Web e bases de dados relacionais, com a intenção de complementar e melhorar a oferta de determinado serviço. Aplicações *mashup* são implementadas como *scripts* escritos em EMML e executadas em um motor (*engine*) de execução que processa tais *scripts*.

- **Módulo de Armazenamento**: consiste de dois repositórios básicos, um para armazenamento de dados utilizando uma base de dados relacional, e outro para armazenamento de *scripts* de aplicações em um sistema de arquivos. Esses repositórios podem fazer uso de uma infraestrutura de computação em nuvem para armazenar dados e arquivos, de forma a aproveitar os atributos de qualidade inerentes a este ambiente como robustez, confiabilidade, disponibilidade e escalabilidade.

- **Módulo de Visualização e Gerenciamento**: responsável por prover uma interface Web que permita aos usuários gerenciarem os dispositivos conectados à EcoDiF. Dentre as funcionalidades da interface Web está a de monitorar o estado e localização dos dispositivos, bem como visualizar dados históricos armazenados na plataforma, além de possibilitar o envio de notificações baseada em eventos (valores atuais dos *feeds*) que especificam condições previamente definidas.

- **Módulo de Manipulação de Dados**: tem como função tratar os dados recebidos dos dispositivos estruturados no protocolo EEML para após realizar o registro efetivo em uma base de dados relacional *MySQL*, utilizando as especificações da *Java Persistence API* (JPA) implementadas no *framework* Hibernate.

- **Módulo de Serviços Comuns**: é responsável por serviços de infraestrutura da plataforma, tais como segurança (permite o controle de autenticidade, confidencialidade e integridade de usuários utilizando as especificações *Java Authentication and Authorization Service* (JAAS) implementadas no servidor de aplicações JBoss), gerenciamento do ciclo de vida de aplicações e transações.

3.2 S^{3}OIA - Smart Spaces and Smart Objects Interoperability Architecture

O trabalho S^{3}OIA (VEGA-BARBAS et al., 2012) tem como objetivo contribuir para a padronização e interoperabilidade da IoT, através do desenvolvimento de uma arquitetura altamente distribuída, orientada a serviços e que integra diferentes objetos físicos e virtuais, abstraindo diferentes tecnologias. A arquitetura é alinhada com padrões Web RestFul e utiliza Espaço de Tuplas para expressar semanticamente informações dos diferentes dispositivos integrados pela plataforma. Além disso a arquitetura possibilita a utilização subjacente de recursos heterogêneos como um substrato para a composição automática de aplicativos complexos, criados de forma dinâmica e adaptável, uma vez que são capazes de evoluir em função do contexto em que são executados.

Devido a capacidade limitada de alguns dispositivos IoT, a arquitetura provê a utilização de gateways, que podem realizar funções de controle e gestão de recursos por estes dispositivos, além de realizar a abstração destes e normalização dos dados, de forma a tratar a heterogeneidade .

Neste trabalho é vislumbrado três níveis de abstração para um cenário IoT, mostrado na Figura 7. O nível mais baixo de abstração diz respeito a camada onde estão dispostos os dispositivos físicos. O nível intermediário é o da abstração dos serviços disponibilizados pelos dispositivos, podendo estes serem descobertos. Por fim a camada de mais alto nível retrata os serviço agrupados em um espaço inteligente virtual que permite a cooperação entre serviços.

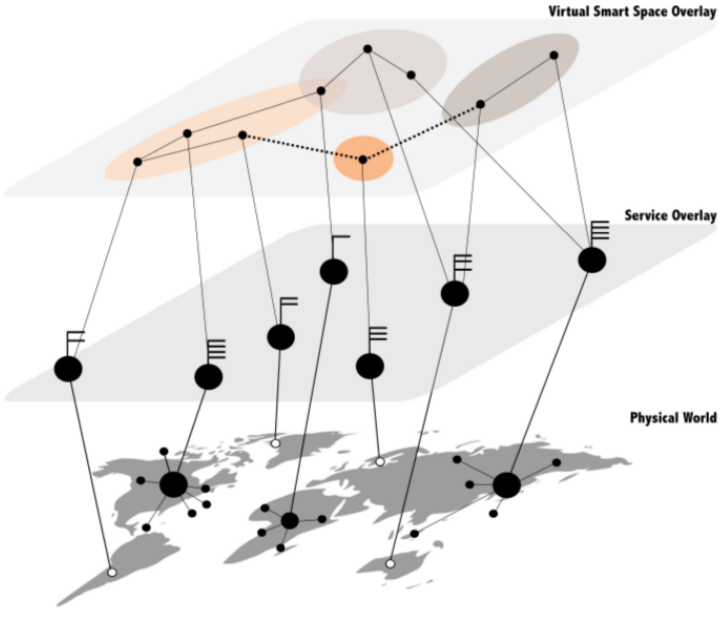

Figura 7: Níveis de Abstração S^{3}OIA. Fonte: (VEGA-BARBAS et al., 2012)

A arquitetura do S^{3}OIA, apresentada na Figura 8, é organizada em módulos e possui cinco grupos funcionais.

1. **Descoberta de Dispositivos e Serviços**: é composto pelos módulos responsáveis pela descoberta, integração e abstração de todos os dispositivos e seus serviços, não importando o protocolo de comunicação usado pelos mesmos. Este grupo contempla dois casos: (i) comunicação entre dispositivos capazes de implementar protocolos como DPWS e UPnP e (ii) comunicação entre estes e outros dispositivos que não possuem a capacidade de implementar estes protocolos, sendo esta situação possível devido a utilização de gateways que abstraem os dispositivos, fornecendo uma interface comum de comunicação e servindo de *proxy* para os protocolos envolvidos na troca de dados.

2. **Exposição de *Web Services* e Semântica *Triple Spaces***: este conjunto de módulos possibilita realizar uma computação distribuída baseada na semântica *Triple Spaces* para sistemas ubíquos, de forma que dispositivos heterogêneos possam compartilhar informações assincronamente e de forma orientada a recursos. A computação *Triple Spaces* executa uma comunicação baseada em espaço de tuplas usando triplas RDF, em que a unidade de informação tem três dimensões que expressam os dados semânticos: sujeito, predicado e objeto. A *Triple Spaces* oferece também autonomia de referência, tempo e espaço, que habilita expor uma API compatível Web que objetiva compartilhar conhecimento entre grupos de nós semelhantes e cientes de contexto.

3. **Repositório de Serviços e Resolução de Dependências**: gerencia os serviços disponíveis dentro de um espaço inteligente e resolve as dependências extraídas da composição de aplicação em um contexto local. Além disso, ele possui um módulo gerenciador de eventos que segue o paradigma *publish/subscribe*, que facilita a criação de aplicações e a composição de serviços. Dessa forma, quando um novo serviço é disponibilizado ou quando um serviço já estabelecido se torna indisponível, seu estado é comunicado para todos os outros módulos envolvidos. O repositório de serviços representa uma instância de armazenamento de informações de todo o espaço IoT distribuído.

4. **Interface de Interação**: este grupo funcional gerencia a interação entre os seres humanos e a plataforma, dentro dos limites do espaço inteligente. A ideia principal deste grupo de módulos é a de recuperar informações a partir da interação dos usuários, as intenções, que podem ser convertidas em um tipo de ordem com o objetivo de compor aplicações complexas, ou para adequar os objetos já implantados às necessidades dos usuários.

5. **Composição, Tolerância a Falhas e Dependências Distantes**: responsável por tratar da composição e orquestração de serviços, além de gerenciar o acesso aos recursos da plataforma quando dois ou mais objetos requisitam-nos simultaneamente. Devido a arquitetura ser descentralizada, as inconsistências e dependências que não são resolvidas localmente são disparadas para camadas superiores. Existem também módulos responsáveis em tratar inconsistências, desconexões ou falhas na rede distribuída.

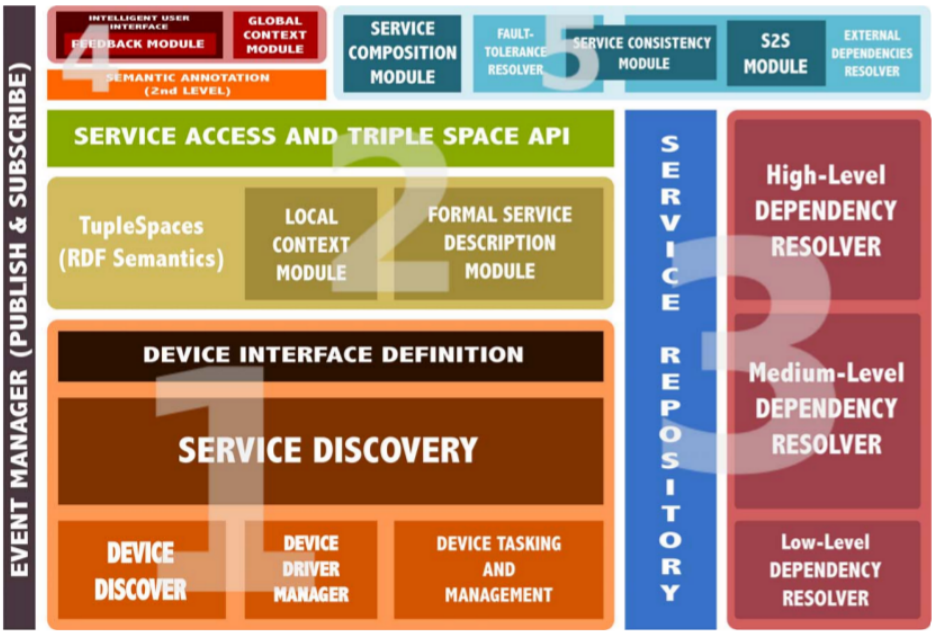

Figura 8: Arquitetura S^{3}OIA. Fonte: (VEGA-BARBAS et al., 2012)

3.3 LinkSmart

O projeto LinkSmart (anteriormente chamado Hydra) é uma plataforma de middleware baseado em uma Arquitetura Orientada a Serviços (SOA - *Service Oriented Architecture*) e desenvolvido na linguagem Java, que oferece suporte ao desenvolvimento de aplicações baseadas em informações fornecidas por dispositivos físicos heterogêneos, disponibilizando interfaces de serviços Web para controle destes dispositivos (WIKI LINKSMART, 2015).

Os dispositivos físicos integrados a plataforma são classificados como: dispositivos nativos (com capacidade computacional para hospedar a plataforma) ou dispositivos não nativos (dispositivos restritos, sem capacidade para hospedar a plataforma), no entanto podem ser integrados a ela através de gateways.

Para os dispositivos não nativos, é necessário uma máquina hospedeira (gateway) com no mínimo 1GB de memória RAM livre e sistema operacional com Java 1.6, OSGi,

.NET 3.5 (ou mono com suporte WCF- *Windows Communication Foundation*). Esta máquina deverá conter uma *proxy* para cada dispositivo ligado a ela, a fim de intermediar a comunicação da plataforma com estes dispositivos. Pela *proxy*, o gateway fica capacitado a trocar dados com o dispositivo, colocando-o na rede LinkSmart. Já os dispositivos nativos devem possuir no mínimo 256MB de RAM livre e sistema operacional com Java 1.6, OSGi, para hospedarem os códigos Java e bibliotecas LinkSmart e OSGi, necessários para integrá-los diretamente a plataforma.

Sua arquitetura possui três camadas principais: (i) camada de rede, responsável pela comunicação com os dispositivos; (ii) camada de serviço, responsável pelo gerenciamento de eventos, dispositivos, escalonamento de recursos, dentre outros, e; (iii) camada semântica.

A arquitetura LinkSmart é apresentada na Figura 9 e seus componentes caracterizados a seguir.

- **Gateway**: é representado por dispositivos computacionais com conexão à Internet contendo *proxies* que traduzem o tipo de tecnologias e formatos de dados empregados pelos dispositivos físicos ligados a ele. É responsável por integrar os dispositivos que não são nativos à rede LinkSmart. Possui suporte a conexão com estes dispositivos através de *USB, ZigBee e Bluetooth*. O gateway deve executar um gerenciador de rede que registre todos os serviços que pertencem aos dispositivos na rede LinkSmart.

- **Network Manager**: implementa serviços Web sobre a especificação JXTA para o modelo *Peer-to-Peer* com o intuito de realizar a comunicação entre os dispositivos.

- **Event Manager**: gerencia a troca de informações através do modelo de comunicação *Publish-Subscribe*, entre processos de todas as propriedades de dados não funcionais para serviços/componentes, dispositivos e rede.

- **Crypto and Trust**: realiza operações criptográficas, a avaliação de confiança e a execução de políticas de segurança de controle de acesso.

- **Device Application Catalogue (DAC)**: responsável por manter o controle de dispositivos disponíveis na rede LinkSmart e acesso destes pelas aplicações.

A LinkSmart também contempla uma descrição semântica dos dispositivos através do uso de ontologias de dispositivos, capazes de representar todas as meta-informações sobre os dispositivos. A ontologia utilizada é baseada na ontologia de dispositivos FIPA (*Foundation for Intelligent Physical Agents*) que permite a

parametrização semântica para incluir informações dos dispositivos, como seus recursos de segurança.

O módulo *Application Ontology Manager* é o responsável por prover uma interface para o uso da ontologia de dispositivos, tanto para buscar propriedades e funções dos dispositivos, quanto para descobertas e atualizações de dispositivos. Nesse contexto, a camada semântica da LinkSmart contém: (i) um módulo de inferências (*reasoner*), responsável por inferir sobre o status dos dispositivos e indicar qual tipo de dispositivo entrou na rede; (ii) um módulo de consulta (*query*) para recuperar informações sobre os dispositivos e suas capacidades; (iii) um módulo de atualização, que permite inclusão, remoção e mudanças na ontologia; (iv) um módulo de versionamento, que gerencia diferentes versões da ontologia, incluindo diferentes versões dos dispositivos e serviços, e; (v) um módulo de interpretação e anotação, responsável por automaticamente atualizar a ontologia com novos tipos de dispositivos e por realizar análise e anotação dos dispositivos existentes e descrições que são incluídas na ontologia.

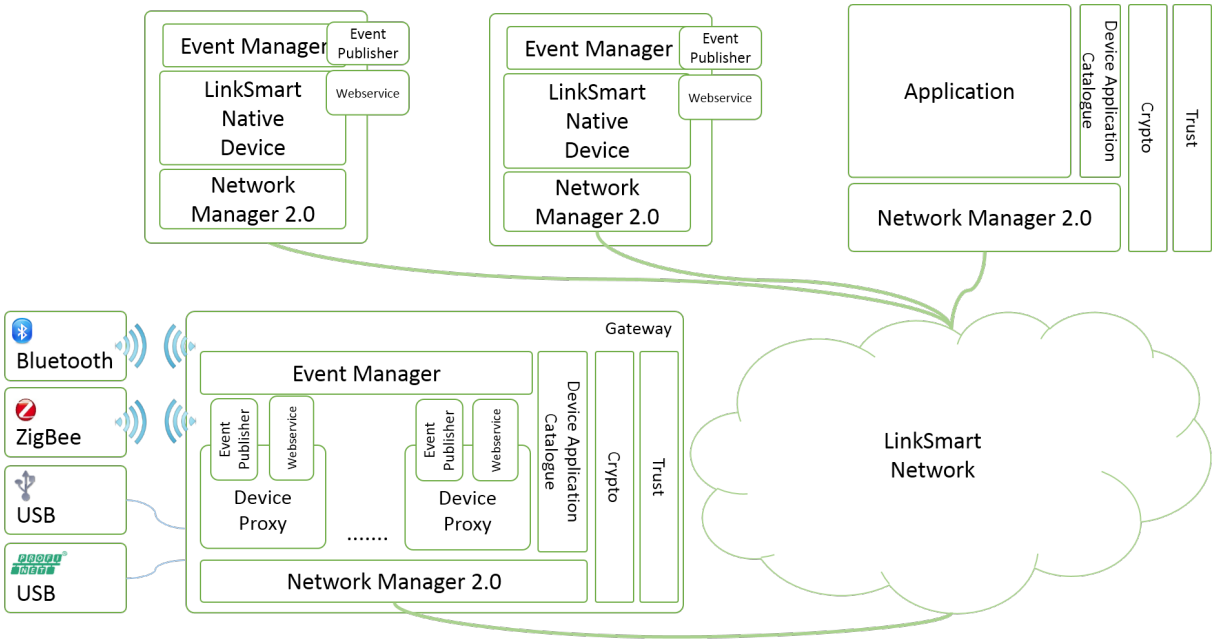

Figura 9: Arquitetura LinkSmart. Fonte: (WIKI LINKSMART, 2015)

3.4 Carriots

A Carriots é uma plataforma para aplicações em IoT que utiliza serviços de nuvem para gerenciar dados providos por qualquer tipo de dispositivo, além de conectar dispositivos a dispositivos e a outros sistemas, o que a faz se alinhar ao conceito de *Platform as a Service* (PaaS) da computação em nuvem (CARRIOTS, 2015). Portanto, se um sistema for conectado à plataforma, ele também pode ser modelado como um dispositivo. A partir de sua API RESTful, a Carriots tem por objetivo coletar e armaze-

nar qualquer dado originado dos mais diversos dispositivos, utilizá-lo em seu motor de aplicações e disponibilizá-lo a seus usuários não importando o volume de dispositivos conectados.

A plataforma Carriots foi projetada tendo como base dois blocos principais: (i) organização lógica de entidades e (ii) arquitetura de componentes de software.

No bloco de organização lógica de entidades, as entidades são definidas de forma hierárquica como demonstra o exemplo da Figura 10. Nela se verifica que os dispositivos são organizados primeiramente em grupos, de acordo com a sua localização geográfica. Por sua vez estes grupos são associados a um serviço que pertence a um projeto específico desenvolvido pelo usuário. Dessa forma, a primeira atividade de um usuário ao interagir com a plataforma é a criação de um projeto. Por conta dessa hierarquia, se um projeto for desabilitado, isso impactará na desativação de todas as entidades (serviços, grupos e dispositivos) vinculadas a ele.

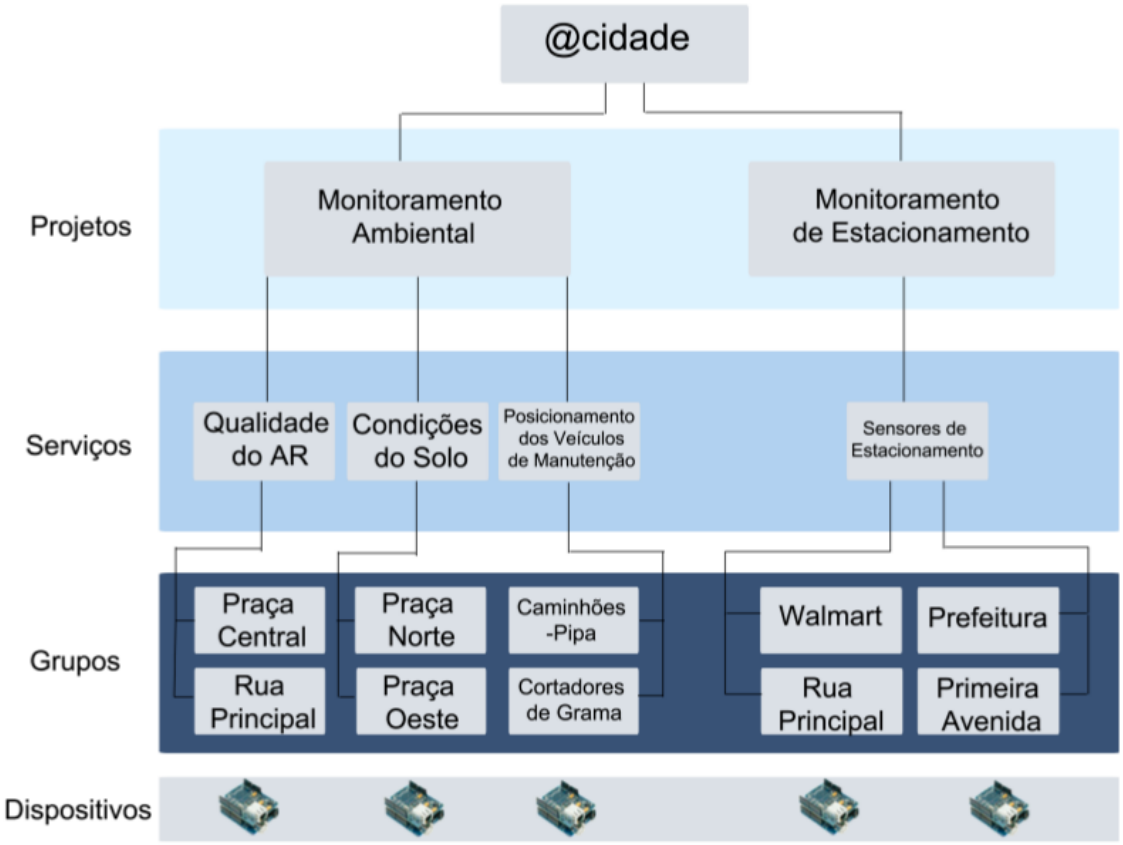

Figura 10: Exemplo de Hierarquia das Entidades na Plataforma Carriots. Fonte: (PIRES et al., 2015)

Outro componente que pode ser inserido nesta hierarquia de entidades é o *Listener*, responsável por realizar o monitoramento de eventos, como o recebimento ou persistência de dados, e caso um evento monitorado atenda alguma condição pré-configurada executar a análise e processamento dos dados relacionados a este

evento.

A arquitetura de software do Carriots (vide Figura 11) é formada pelos seguintes módulos:

- **API REST**: seguindo padrões da Internet esta API REST é implementada sobre HTTPS, sendo responsável pela forma como os dispositivos e outros sistemas interagem com a plataforma. A API REST utiliza os formatos JSON ou XML (em um formato particular da plataforma) para representação dos dados. Possui plena capacidade de interação entre a plataforma e os painéis de controle personalizados, *dashboards*, etc.

- ***Big Data***: provê as aplicações a flexibilidade de gerenciar os dados dos mais diversos dispositivos, de modo que a massa de dados seja armazenada em uma arquitetura de *Big Data* em formato *schemaless*, permitindo assim armazenar dados sem que seja necessário normalizá-los.

- **Gerenciamento de Projetos e Dispositivos**: possui como funcionalidade armazenar e gerenciar os projetos criados pelos usuários (dos mais simples aos mais complexos), podendo estes serem organizados para atenderem a quaisquer requisitos. Este módulo é também responsável por realizar a configuração dos dispositivos, como por exemplo o ajuste na taxa de amostragem de um sensor de estacionamento e atualizar os seus softwares embarcados, possuindo a capacidade de realizar o gerenciamento remoto dos dispositivos .

- **Processamento de Eventos e Regras de Negócio**: estes módulos são responsáveis por armazenar e executar eventos, segundo uma lógica de negócios, em forma de *scripts* criados utilizando a linguagem de programação *Groovy* e fazendo uso de regras do tipo *if-then-else*.

- **Segurança**: neste módulo questões de segurança podem ser tratadas de quatro formas: (i) através do uso de chaves (*APIkeys*) pré-compartilhadas para a definição de privilégios de acesso; (ii) através da utilização do protocolo HTTPS para a criptografia das requisições e respostas à API REST; (iii) pela utilização de Hash HMAC (*Keyed-Hash Message Authentication Code*) e senha para autenticação e verificação de conteúdo, e; (iv) por criptografia customizada a partir de *scripts* criados pelo usuário, permitindo a adição de soluções de segurança adicionais.

- ***Logs e Debug***: fornece um console de *debug* que facilita o desenvolvimento dos projetos hospedados na plataforma, através de ferramentas para depuração de erros e registro de mensagens, tornando tais dados (mensagens de *log*) acessíveis a partir do painel de controle.

- **Painel de Controle**: permite o gerenciamento pelo usuário de todos os outros módulos e recursos da plataforma a partir de uma interface Web.
- **Módulo de Comunicação Externa**: complementa os recursos de interação da plataforma ao permitir o envio de *e-mails*, SMS e a interação com outros sistemas (como a plataforma de compartilhamento de arquivos Dropbox ou a rede social Twitter).

Figura 11: Arquitetura Carriots. Fonte: (PIRES et al., 2015)

3.5 Análise das Plataformas de Middleware

Uma análise das plataformas de middleware discutidas anteriormente em relação à satisfação dos requisitos IoT considerados neste trabalho é apresentada na Tabela 7, onde o símbolo “X” denota que o requisito é completamente atendido, o símbolo “O” indica que o requisito é parcialmente atendido, e o símbolo “-” indica que o requisito não é atendido.

Tabela 7: Satisfação dos Requisitos pelas Plataformas de Middleware

Plataformas de middleware	Interoperabilidade	Descoberta e Gerenciamento de Dispositivos	Interfaces de Alto Nível	Ciência de Contexto	Escalabilidade
EcoDiF	X	O	X	O	X
S^{3}OIA	X	X	-	X	-
LinkSmart	X	O	-	X	-
Carriots	X	O	X	O	X

Diante do exposto na Tabela 7, constata-se que nenhuma plataforma de middleware de IoT foi capaz de atender a todos os requisitos levantados. Tais plataformas tratam apenas subconjuntos dos requisitos, de acordo com sua aplicação e objetivos. A Interoperabilidade é o único requisito atendido por todas as plataformas, pois este é um requisito primordial e deve ser imperativamente endereçado por qualquer plataforma de middleware IoT.

Na análise dos trabalhos algumas características foram sistematizadas, como a tendência de middlewares IoT em usar princípios arquiteturais REST como meio para

disponibilização dos dispositivos e seus recursos, devido ao seu estilo simples e de fácil interpretação, e por este utilizar padrões abertos e protocolos amplamente utilizados na Internet, como o HTTP, URI, XML.

Outra característica é a adoção de *gateways* como forma de abstrair e integrar dispositivos de sensoriamento e/ou atuação com recursos computacionais limitados, bem como tratar a heterogeneidade e normalizar os dados produzidos.

Dentre os trabalhos revistos o único que apresentou uma estratégia de descoberta de dispositivos automática, *Plug and Play* e de auto configuração para o gerenciamento de sensores e/ou atuadores, foi o S^3OIA, através do uso do protocolo de comunicação UPnP.

4 CONSIDERAÇÕES FINAIS

A Internet das Coisas vem ganhando destaque como uma forma de materializar as premissas da Computação Ubíqua. No entanto, há uma série de desafios a serem superados para a sua plena implantação.

Um dos principais desafios está na concepção de infraestruturas capazes de integrar um crescente número de dispositivos IoT. Dispositivos estes com capacidade restrita e distribuídos dinamicamente, possuindo as mais diferentes tecnologias de hardware e software, bem como padrões de comunicação.

Para tal, se faz necessária que as infraestruturas concebidas possuam mecanismos capazes de tratar, de forma oportuna, os aspectos decorrentes de uma heterogeneidade e/ou escalabilidade elevadas.

A utilização de middlewares para tratar desafios da IoT vem sendo abordado como uma solução promissora, devido a estes se tornarem responsáveis por prover um meio padronizado para o acesso aos dados e serviços fornecidos pelos objetos inteligentes, através de uma interface de alto nível, facilitando a construção de aplicações para a IoT.

Desta forma este trabalho apresentou o estudo realizado sobre os desafios inerentes na concepção de infraestruturas IoT e as principais abordagens apontadas pela literatura como mais oportunas para endereçar tais desafios.

Protocolos de comunicação na IoT de duas naturezas foram selecionados, primeiro levando em consideração aspectos de autoconfiguração e identificação automática que facilitam a gerência dos dispositivos da IoT, com este intuito o UPnP foi apresentado e caracterizado. A outra abordagem considerou os protocolos CoAP e MQTT como protocolos mais usuais, que objetivam minimizar o consumo computacional e energético dos dispositivos restritos da IoT.

Também foram revisados e analisados trabalhos que constituem plataformas de middlewares para IoT, o que culminou em uma sistematização das principais estratégias utilizadas na concepção de infraestruturas, de forma a tratar os desafios inerentes ao cenário da IoT.

REFERÊNCIAS

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

BANDYOPADHYAY, S.; SENGUPTA, M.; MAITI, S.; DUTTA, S. Role of middleware for internet of things: A study. **International Journal of Computer Science & Engineering Survey (IJCSES)**, USA, v.2, n.3, p.94–105, 2011.

BANKS, A.; GUPTA, R. MQTT Version 3.1.1. **OASIS Standard**, [S.l.], 2014. Disponível em: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf >. Acesso em Julho de 2015.

CARRIOTS. **Carriots IoT Application Platform**. Disponível em :<https://www.carriots.com/>. Acesso em julho de 2015.

CHAQFEH, M.; MOHAMED, N. et al. Challenges in middleware solutions for the internet of things. **Collaboration Technologies and Systems (CTS), International Conference on**, USA, p.21–26, 2012.

DAL MORO, T.; DORNELES, C.; REBONATTO, M. T. Web services WS-* versus Web Services REST. **Revista de Iniciação Científica**, [S.l.], v.11, n.1, 2009.

DELICATO, F. C.; PIRES, P. F.; BATISTA, T.; CAVALCANTE, E.; COSTA, B.; BARROS, T. Towards an IoT ecosystem. **Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems - ACM**, USA, p.25–28, 2013.

DEY, A. K. Understanding and Using Context. **Personal and Ubiquitous Computing**, [S.l.], v.5, p.4–7, 2001.

DLNA. **Digital Living Network Alliance**. Disponível em: <http://www.dlna.org/ >. Acesso em Julho de 2015.

EVANS, D. A Internet das Coisas - Como a próxima evolução da Internet está mudando tudo. **Cisco Internet Business Solutions Group**, IBSG, 2011.

FERREIRA, H. G. C. **Arquitetura de Middleware para Internet das Coisas**. 2014. Dissertação de Mestrado — Programa de Pós-Graduação em Engenharia Elétrica - Universidade de Brasília.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Tese de Doutorado em Informação e Ciência da Computação — Universidade da Califórnia, Califórnia-USA.

FORSSTRÖM, S.; KANTER, T. Enabling ubiquitous sensor-assisted applications on the internet-of-things. **Personal and ubiquitous computing**, [S.l.], v.18, n.4, p.977–986, 2014.

GOUVEIA, P. R. N. T. **Convergência de Redes Sem Fios para Comunicações M2M e Internet das Coisas em Ambientes Inteligentes**. 2013. Dissertação de Mestrado — Universidade da Beira Interior, Covilhã.

GUILLEMIN, P.; FRIESS, P. Internet of things strategic research roadmap. **The Cluster of European Research Projects**, Europe, September 2009.

GUINARD, D. Towards Opportunistic Applications in a Web of Things. **IEEE International Conference on Pervasive Computing and Communications Workshops**, USA, 2010.

IOT-A. **Internet of Things Architecture of the European Lighthouse Integrated Project**. Disponível em :<http://http://www.iot-a.eu>. Acesso em dezembro de 2014.

LEE, G. M.; KIM, J. Y. The Internet of Things?A problem statement. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGY CONVERGENCE (ICTC), 2010., 2010. **Anais...** [S.l.: s.n.], 2010.

LIMA, A. T. **Aplicação de Internet of Things em casas inteligentes - Serviço Aplicacional**. 2014. Dissertação de Mestrado — Instituto Politécnico do Porto. Instituto Superior de Engenharia do Porto.

LOPES, J.; SOUZA, R.; GEYER, C.; COSTA, C.; BARBOSA, J.; PERNAS, A.; YAMIN, A. A Middleware Architecture for Dynamic Adaptation in Ubiquitous Computing. **J.UCS**, [S.l.], v.20, n.9, p.1327–1351, sep 2014.

MAIA, P.; BAFFA, A.; CAVALCANTE, E.; DELICATO, F. C.; BATISTA, T.; PIRES, P. F. Uma Plataforma de Middleware para Integração de Dispositivos e Desenvolvimento de Aplicações em e-health. **Anais do XXXIII SBRC**, Vitória - ES, p.361–374, 2015.

MARTINS, I. R.; ZEM, J. L. Estudo dos Protocolos de Comunicação MQTT e COAP para Aplicações Machine-to-Machine e Internet das Coisas. **Americana**, [S.l.], v.3, n.1, p.64–87, 2015.

MQTT. **Message Queue Telemetry Transport**. Disponível em: <http://mqtt.org/ >. Acesso em Julho de 2015.

NAGY, M.; KATASONOV, A.; SZYDLOWSKI, M.; KHRIYENKO, O.; NIKITIN, S.; TERZIYAN, V. **Challenges of middleware for the internet of things**. Croatia: INTECH Open Access Publisher, 2009.

OASIS-MQTT. **OASIS Message Queuing Telemetry Transport (MQTT) TC**. Disponível em: <https://www.oasis-open.org/committees/mqtt >. Acesso em Julho de 2015.

PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. Context aware computing for the internet of things: A survey. **Communications Surveys & Tutorials, IEEE**, USA, v.16, n.1, p.414–454, 2013.

PIRES, P. F.; CAVALCANTE, E.; BARROS, T.; DELICATO, F. C.; BATISTA, T.; COSTA, B. A platform for integrating physical devices in the Internet of Things. **Proceedings of the 12th IEEE International Conference on Embedded and Ubiquitous Computing**, USA, p.234–241, 2014.

PIRES, P. F.; DELICATO, F. C.; BATISTA, T.; BARROS, T.; CAVALCANTE, E.; PITANGA, M. Plataformas para a Internet das Coisas. **Livro Texto de Minicursos - SBRC 2015**, Vitória - ES, 2015.

SHELBY, Z.; HARTKE, K.; BORMANN, C. **Constrained Application Protocol (CoAP), IETF Standards Track RFC Proposed Standard, CORE Working Group**. Disponível em: <http://datatracker.ietf.org/doc/draft-ietf-core-coap/ >. Acesso em Julho de 2015.

UPNP, F. **Fórum UPnP**. Disponível em :<http://upnp.org//>. Acesso em Julho de 2015.

VEGA-BARBAS, M.; CASADO-MANSILLA, D.; VALERO, M.; IPINA, D. López-de; BRAVO, J.; FLÓREZ, F. et al. Smart spaces and smart objects interoperability architecture (S3OiA). **Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on**, USA, p.725–730, 2012.

WIKI LINKSMART, . **Middleware for Networked Embedded Systems**. Disponível em :<https://linksmart.eu/redmine/projects/linksmart-opensource/wiki>. Acesso em julho de 2015.