Capítulo 3

Plataformas para a Internet das Coisas

Paulo F. Pires, Flavia C. Delicato, Thais Batista, Thomaz Barros,
Everton Cavalcante e Marcelo Pitanga

Resumo

A Internet das Coisas (IoT) é um paradigma no qual objetos inteligentes colaboram de forma ativa com outros objetos físicos e virtuais disponíveis na Internet. Ambientes de IoT são caracterizados por um alto grau de heterogeneidade de dispositivos e protocolos de rede. Para tratar tal heterogeneidade, várias plataformas de middleware para IoT têm sido propostas visando abstrair as especificidades de tais dispositivos para aplicações e usuários finais, bem como promover interoperabilidade entre eles. Entretanto, a falta de padronização existente em IoT faz com que tais plataformas adotem diferentes modelos de programação e não abordem adequadamente uma série de requisitos importantes nesse contexto. Dessa forma, arquiteturas de referência (ARs) são capazes de definir um conjunto inicial de blocos de construção para ambientes de IoT e fornecer uma base sólida para alavancar sua ampla adoção. Tendo em vista a relevância de plataformas de middleware e arquiteturas de referência em IoT, bem como os desafios e oportunidades de pesquisa existentes, este capítulo tem por objetivos: (i) levantar e discutir requisitos de middleware para IoT; (ii) apresentar ARs e plataformas de middleware para IoT existentes; e (iii) demonstrar o uso prático de uma plataforma de IoT.

Abstract

The Internet of Things (IoT) is a paradigm in which smart objects actively collaborate with other physical and virtual objects available in the Internet. IoT environments are characterized by a high degree of heterogeneity, encompassing devices with different capabilities, functionalities, and network protocols. To address such heterogeneity, several middleware platforms have been proposed aiming at abstracting away the specificities of such devices from applications and/or end-users, as well as promoting interoperability among them. Nevertheless, the lack of standardization in IoT makes these platforms to adopt different programming models and to not properly address several important requirements in this context. Therefore, reference architectures are

able to define an initial set of building blocks for IoT environments and to provide a solid foundation for leveraging its wide adoption. Due to the relevance of middleware platforms and reference architectures in IoT, as well as the existing research challenges and opportunities, this chapter aims to: (i) elicit and discuss requirements for IoT platforms; (ii) present existing reference architectures and middleware for IoT; (iii); demonstrate the practical use of an IoT middleware platform.

3.1. Introdução

A Internet das Coisas (do inglês *Internet of Things – IoT*) [Atzori *et al*. 2010] é um paradigma que preconiza um mundo de objetos físicos embarcados com sensores e atuadores, conectados por redes sem fio e que se comunicam usando a Internet, moldando uma rede de objetos inteligentes capazes de realizar variados processamentos, capturar variáveis ambientais e reagir a estímulos externos. Esses objetos interconectam-se entre si e com outros recursos (físicos ou virtuais) e podem ser controlados através da Internet, permitindo o surgimento de uma miríade de aplicações que poderão se beneficiar dos novos tipos de dados, serviços e operações disponíveis. A IoT é uma das principais tecnologias emergentes que contribuem para concretizar novos domínios de aplicação das tecnologias de informação e comunicação (TICs), a exemplo do domínio de cidades inteligentes, no qual o uso de tecnologias avançadas de comunicação e sensoriamento visa prover serviços de valor agregado para os órgãos administrativos de tais cidades e para seus cidadãos [Zanella *et al*. 2014].

Vários avanços tecnológicos recentes possibilitaram o surgimento da IoT, tais como redes de sensores sem fio, comunicação móvel e computação ubíqua. No entanto, há ainda uma série de desafios a serem superados para alavancar a ampla disseminação desse paradigma, principalmente com relação ao desenvolvimento de aplicações e à alta heterogeneidade decorrente da inerente diversidade de tecnologias de *hardware* e *software* desse ambiente. O primeiro desafio diz respeito a heterogeneidade dos ambientes de IoT, a qual demanda soluções para permitir a interoperabilidade e integração dos diversos componentes que fazem parte desses ambientes. Nesse contexto, plataformas de *middleware* têm surgido como soluções promissoras para prover tal interoperabilidade e gerenciar a crescente variedade de dispositivos associados a aplicações, bem como o consumo de dados por parte dos usuários finais [Teixeira *et al*. 2011]. Tais plataformas 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 através de uma interface de alto nível [Bandyopadhyay *et al*. 2011]. A adoção de uma plataforma de *middleware* também pode contribuir para facilitar a construção de aplicações para IoT. Nesse contexto, 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 não precisarão lidar com funcionalidades de baixo nível para a manipulação de tais objetos [Delicato *et al*. 2013a]. Outros desafios concernem à enorme escalabilidade desses ambientes, em termos do número de dispositivos conectados, à necessidade de gerenciar tais dispositivos e ao grande volume de dados produzidos.

Existem várias propostas de *middleware* para IoT, cada uma atendendo um subconjunto dos requisitos necessários para viabilizar tais ambientes, principalmente no que tange a comunicação entre dispositivos heterogêneos. 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. Os trabalhos descritos por Pires *et al*. (2014), Qin *et al*. (2011) e Gao *et al*. (2011) são alguns exemplos de plataformas concebidas para endereçar alguns dos desafios anteriormente descritos, principalmente com relação à integração transparente de dispositivos heterogêneos e à provisão de mecanismos de alto nível para desenvolvimento de aplicações. Contudo, tais propostas ainda não atingiram um nível de maturidade, requerendo esforços de pesquisa adicionais, como: (i) a construção de infraestruturas robustas e tolerantes a falha para gerenciar e processar dados provenientes dos vários dispositivos integrados; (ii) o gerenciamento de incertezas e resolução de conflitos, e; (iii) suporte à adaptação de aplicações sob condições dinâmicas do ambiente.

No contexto de IoT, a falta de padronização existente na área faz com que tais plataformas de *middleware* adotem diferentes modelos de programação que, em geral, não são compatíveis entre si, gerando silos verticais que são ainda um obstáculo à plena interoperabilidade requerida por esse paradigma. Outras limitações das soluções existentes dizem respeito ao fato de elas não abordarem requisitos de escalabilidade de forma apropriada, fornecerem modelos inadequados de governança, e negligenciarem questões de privacidade e segurança na sua concepção. Finalmente, faz-se ainda necessária a inclusão de mecanismos para lidar com a massiva quantidade de dados (também potencialmente heterogêneos) produzidos pelos dispositivos interconectados. Tais dados devem ser manipulados de forma eficiente em termos do consumo de recursos dos dispositivos e, ao mesmo tempo, atender as demandas de aplicações, muitas delas em tempo real ou *quasi*-real. Nesse sentido, faz-se necessária a criação de *arquiteturas de referência* que definam um conjunto inicial de blocos de construção para ambientes de IoT, levando em conta todos os requisitos desses ambientes, e forneçam uma base sólida para alavancar sua ampla adoção. Uma arquitetura de referência pode ser entendida como uma arquitetura abstrata que envolve conhecimento e experiências em um domínio de aplicação específico, sendo capaz de facilitar e guiar o desenvolvimento, a padronização e a evolução de sistemas de software em tal domínio [Cloutier *et al*. 2010, Nakagawa *et al*. 2011].

Tendo em vista a relevância do papel desempenhado por plataformas de *middleware* e por arquiteturas de referência no contexto de IoT, bem como os desafios e oportunidades de pesquisa existentes, este capítulo tem por objetivos: (i) levantar e discutir requisitos de *middleware* para IoT; (ii) apresentar arquiteturas de referência para IoT existentes; (iii) descrever plataformas de *middleware* que implementem soluções para os requisitos levantados, e; (iv) demonstrar o uso prático de uma plataforma de *middleware* para IoT. Como resultado, será possível conhecer o estado da arte no desenvolvimento de plataformas de *middleware* e arquiteturas de referência para IoT, fornecer subsídios para avaliar tais soluções, e compreender as tecnologias necessárias para a concretização dos desafios apresentados.

O restante do capítulo está organizado como segue. A Seção 3.2 descreve alguns requisitos importantes de plataformas de *middleware* para IoT. A Seção 3.3 apresenta duas arquiteturas de referencia para IoT. A Seção 3.4 discute algumas plataformas de

middleware encontradas nos meios acadêmico e comercial. A Seção 3.5 descreve, através de cenários de uso típicos do domínio de IoT, como uma plataforma de *middleware* pode ser usada na construção de aplicações. A Seção 3.6 contém as considerações finais.

3.2. Requisitos de Middleware para IoT

No contexto de IoT, plataformas de *middleware* devem satisfazer a um conjunto de requisitos visando atender às necessidades de aplicações e usuários, bem como endereçar os desafios que surgem nesse cenário. Esta seção aborda alguns dos requisitos considerados fundamentais para plataformas de *middleware* em IoT [Nagy *et al*. 2009, Bandyopadhyay *et al*. 2011, Chaqfeh e Mohamed 2012] e que são frequentemente mencionados na literatura, a saber: (i) interoperabilidade; (ii) descoberta e gerenciamento de dispositivos; (iii) interfaces de alto nível; (iv) ciência de contexto; (v) escalabilidade; (vi) gerenciamento de grandes volumes de dados; (vii) segurança, e; (viii) adaptação dinâmica. É importante mencionar que alguns desses requisitos são inerentes a toda e qualquer plataforma de *middleware*, independentemente do domínio de aplicação, tais como interoperabilidade e escalabilidade. Outros, por sua vez, são compartilhados com plataformas de *middleware* em computação ubíqua, paradigma correlato à IoT, a exemplo da ciência de contexto. Por fim, outros requisitos, tais como adaptação dinâmica, apesar de previamente existentes em outros contextos, são de fundamental importância ou mesmo mandatórios nesse novo cenário.

Um primeiro requisito a ser imperativamente endereçado por uma plataforma de *middleware* em IoT diz respeito a prover **interoperabilidade** entre os diversos dispositivos e plataformas disponíveis no ambiente. Esse é um dos principais desafios para a concretização do paradigma de IoT devido ao grande número de dispositivos a serem integrados e sua heterogeneidade tanto em termos de *hardware* quanto de *software*, protocolos (muitos deles proprietários), formatos de dados, etc. Dessa forma, ao endereçar a questão da interoperabilidade, plataformas de *middleware* passam a desempenhar um papel de suma importância no cenário de IoT por permitirem que aplicações sejam criadas de maneira mais rápida e com maior valor agregado para os usuários, podendo fazer uso de diversos dispositivos e/ou mesmo outras aplicações. Delicato *et al*. (2013a) destacam 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 [Bandyopadhyay *et al*. 2011];
- 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.

Uma característica comum da infraestrutura de comunicação em ambientes de IoT diz respeito ao fato de que sua topologia é dinâmica e frequentemente desconhecida, visto que dispositivos podem ser integrados ao ambiente e utilizados de maneira oportunista e não previamente planejada [Delicato *et al*. 2013b]. Dessa forma, é igualmente 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 dispositivos.

Ciência de contexto é outro requisito importante para plataformas de *middleware* para IoT. De acordo com Dey (2001), *contexto* é qualquer informação que pode ser utilizada para caracterizar uma pessoa, lugar ou objeto considerado relevante ao ambiente em questão. Dessa forma, informações de contexto, tais como o estado do objeto, seus vizinhos e sua localização, por exemplo, necessitam ser coletadas e processadas com o objetivo de efetuar ações ou reagir a estímulos com base nos dados extraídos [Perera *et al*. 2014]. Plataformas de *middleware* em IoT devem então ser responsáveis pela coleta, gerenciamento e processamento de informações de contexto providas por múltiplas fontes, liberando as aplicações e usuários da tarefa de manipulá-las e tornando transparente tal manipulação.

Considerando o amplo potencial do paradigma de IoT, a consultoria americana Gartner, Inc. prevê que bilhões de dispositivos estarão aptos a serem utilizados por aplicações em curto prazo de tempo [Gartner 2014]. Dessa forma, uma plataforma de *middleware* para IoT deve dar suporte à **escalabilidade**, i.e., deve ser capaz de assimilar um número crescente de dispositivos e requisições e funcionar corretamente, mesmo em situações de uso intenso. Devido à sua facilidade de provisão e uso de recursos computacionais, que podem ser alocados e liberados sob demanda, o paradigma de computação em nuvem tem surgido como uma solução promissora para endereçar a questão da escalabilidade em ambientes de IoT, proporcionando o surgimento da chamada "nuvem de coisas" (*cloud-of-things*, em Inglês) [Soldatos *et al*. 2012].

Com o aumento do número de dispositivos presentes em um ambiente de IoT, cresce também o volume de dados providos e transmitidos através da rede. Nesse contexto, o **gerenciamento de grandes volumes de dados** é um requisito importante de uma plataforma de *middleware* para IoT, permitindo que ela possa acompanhar a demanda de coleta e análise de dados e, consequentemente, prover respostas, decisões e/ou atuações de maneira eficiente. Nesse contexto, 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, por exemplo. Recentemente, soluções baseadas em *Big Data* e também em computação em nuvem têm surgido como uma potencial resposta a alguns desses desafios, a fim de permitir lidar com um imenso volume de dados, diverso e não estruturado [Soldatos *et al*. 2012, Tracey e Sreenam 2013, Chen *et al*. 2014].

No contexto de IoT, muitas vezes o papel dos dispositivos integrados é de coletar dados privados que podem inclusive ser transportados através de redes sem segurança adequada. Por essa razão, é importante que uma plataforma de *middleware* em IoT forneça 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. Técnicas como a prevenção à modificação maliciosa de dados (*tamperproofing*) e ofuscação de código podem ser usadas para endereçar segurança com relação aos dispositivos, enquanto que é possível adotar estratégias para promover a segurança dos recursos, tais como bloqueio de portas abertas não usadas, e o uso de protocolos de segurança e de protocolos de autorização e/ou autenticação.

Por fim, considerando a alta dinamicidade dos ambientes de 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.), plataformas de *middleware* devem prover estratégias para **adaptação dinâmica**, garantindo assim a disponibilidade e qualidade das aplicações durante a sua execução. Esse é um requisito especialmente importante para aplicações em domínios críticos, a exemplo de aplicações de *health care* que monitoram pacientes, visto que falhas ou degradação de parâmetros de qualidade nesse tipo de aplicação podem ser uma ameaça à vida e à saúde das pessoas. Dessa forma, plataformas de *middleware* em IoT devem manter-se disponíveis e funcionando adequadamente nesse ambiente dinâmico, coletando, analisando e reagindo a mudanças no contexto em que elas e objetos a ela conectados estão inseridos.

3.3. Arquiteturas de Referência para IoT

Apesar das diferentes definições encontradas na literatura, uma **arquitetura de referência** pode ser entendida como um tipo de arquitetura abstrata que envolve conhecimento e experiências acerca de como projetar sistemas em um determinado domínio, sendo, portanto capaz de guiar o seu desenvolvimento e evolução. Além disso, arquiteturas de referência podem ser utilizadas como um artefato de padronização para permitir interoperabilidade entre sistemas ou componentes de sistemas [Muller 2008, Angelov *et al*. 2009, Nakagawa *et al*. 2011].

Não raramente, os termos **arquitetura de referência** e **modelo de referência** têm sido utilizados de maneira intercambiável ou mesmo como sinônimos; entretanto, tais termos possuem definições distintas. Um **modelo de referência** é um artefato abstrato que apresenta um conjunto de conceitos comuns e relacionamentos entre eles (podendo ser representados, por exemplo, através de modelos conceituais, taxonomias ou ontologias) com relação a um domínio específico, sendo, portanto independente de padrões, tecnologias, implementações ou outros detalhes mais concretos [Nakagawa *et al*. 2014]. Por sua vez, uma **arquitetura de referência** pode ser concebida com base em um ou mais modelos de referência para especificar, de maneira unificada e não ambígua, regras de negócio, estilos/padrões arquiteturais, decisões arquiteturais, boas práticas de desenvolvimento e elementos de *hardware* e/ou *software* necessários à construção de **arquiteturas concretas**, que dizem respeito aos sistemas propriamente ditos no domínio em questão. Ou seja, um modelo de referência pode ser utilizado para prover a base comum a ser adotada para o estabelecimento de uma arquitetura de referência que, por sua vez, fornece os elementos concretos e abstratos que devem ser considerados para conceber a arquitetura de um sistema, que é apenas uma **instância** da

arquitetura de referência. A Figura 3.1 ilustra esses relacionamentos entre modelos de referência, arquiteturas de referência e arquiteturas concretas. É importante destacar que, apesar de ser desejável que arquiteturas de referência sejam estabelecidas com base em modelos de referência devido ao vocabulário comum que estes últimos proveem, essa não é uma condição mandatória.

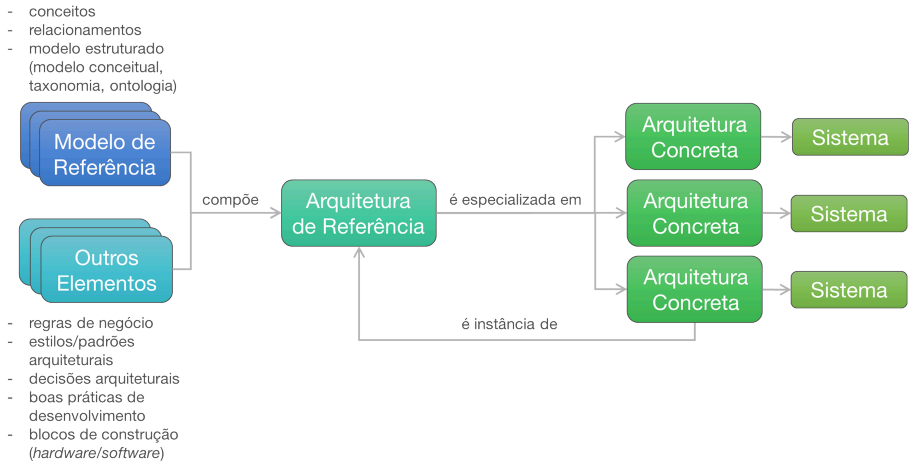

Figura 3.1. Relacionamentos entre modelos de referência, arquiteturas de referência e arquiteturas concretas.

3.3.1. Por que precisamos de uma arquitetura de referência?

Entre os principais objetivos de uma arquitetura de referência, destacam-se [Muller 2008, Angelov *et al*. 2009, Nakagawa *et al*. 2014]:

- **Facilitar o desenvolvimento de sistemas, promovendo redução de tempo e custo.** Arquiteturas de referência são capazes de influenciar diretamente a produtividade e a qualidade do desenvolvimento de sistemas, principalmente pelo fato de elas proverem os elementos fundamentais para a construção das arquiteturas concretas desses sistemas. Nessa perspectiva, o desenvolvimento de um sistema cuja arquitetura é baseada numa arquitetura de referência demanda menor esforço por parte da equipe com relação à investigação e ponderação sobre decisões arquiteturais, por exemplo.
- **Padronizar arquiteturas de sistemas em um domínio.** Através do consenso estabelecido por uma arquitetura de referência em termos de elementos fundamentais a serem considerados e diretrizes a serem seguidas, é possível desenvolver arquiteturas concretas interoperáveis entre si, facilitando assim a integração e compatibilidade entre diferentes sistemas heterogêneos no domínio em questão.
- **Guiar a evolução de sistemas existentes.** Evolução é um processo natural e necessário aos sistemas atualmente em execução, seja ela motivada pela necessidade de adaptá-los a novos contextos, modificar suas funcionalidades a fim de atender novos requisitos, ou promover melhorias em sua qualidade. Dessa forma, o conhecimento estabelecido em uma arquitetura de referência é de fundamental importância para que tal processo seja conduzido de forma

sistemática a fim de endereçar as mudanças necessárias a serem feitas no sistema, evitando, ao mesmo tempo, a erosão de sua arquitetura.

Assim como em diversos outros domínios, o estabelecimento de arquiteturas de referência é uma questão importante em IoT, tendo em vista os objetivos desse tipo de arquitetura. Em primeiro lugar, os direcionamentos providos por uma arquitetura de referência são elementos essenciais para guiar e facilitar a construção de sistemas de IoT, considerando sua crescente escala e complexidade. Mais ainda, por proverem os blocos de construção fundamentais à construção das arquiteturas concretas de tais sistemas, arquiteturas de referência permitem construir sistemas capazes de atender aos requisitos existentes nesse domínio. Por fim, considerando a alta heterogeneidade e a falta de padronização em termos de dispositivos, protocolos e sistemas intrínsecas à IoT, desenvolver soluções que possam ser integradas entre si é um aspecto de suma importância nesse cenário, e tal interoperabilidade pode ser alcançada projetando-se arquiteturas de sistemas que sejam fundamentadas em uma arquitetura de referência.

Apesar de sua relevância, arquiteturas de referência em IoT são um elemento de pesquisa muito recente, com poucas iniciativas até o presente momento. Nas Seções 3.3.2 e 3.3.3, duas propostas de arquiteturas de referência para IoT são apresentadas e discutidas, a saber: (i) o *IoT Architectural Reference Model* [Bassi *et al*. 2013], desenvolvido no contexto do projeto europeu Internet of Things Architecture (IoT-A)[1], e (ii) a arquitetura de referência proposta pela empresa WSO2 [Fremantle 2014].

3.3.2. A arquitetura de referência do IoT-A

O projeto IoT-A propõe um **modelo arquitetural de referência (MAR)** que envolve uma arquitetura de referência base e a definição de um conjunto de características chave para sua construção. Essa arquitetura de referência é definida em um alto nível de abstração, fornecendo visões arquiteturais e perspectivas que são relevantes para a construção de várias (e potencialmente diferentes) arquiteturas para IoT. As visões do MAR fornecem descrições variadas que mostram a arquitetura sob diferentes ângulos e podem ser utilizadas durante as fases de projeto e implementação de uma arquitetura concreta. Uma visão é composta de pontos de vista que agregam vários conceitos arquiteturais. O MAR fornece as seguintes visões: (i) funcional; (ii) informação; (iii) operação, e; (iv) implantação.

Além das visões, as perspectivas definidas no MAR definem as decisões arquiteturais que tratam interesses comuns a mais de uma visão ou mesmo para todas. Esses interesses estão frequentemente relacionados a requisitos não funcionais ou atributos de qualidade. Uma perspectiva arquitetural é definida com uma coleção de atividades, táticas e diretrizes que são usadas para garantir que um sistema exiba um conjunto particular de atributos de qualidade relacionados, os quais influenciam um determinado número de visões arquiteturais de um sistema. O MAR fornece as seguintes perspectivas: (i) evolução e interoperabilidade; (ii) disponibilidade e resiliência; (iii) confiabilidade, segurança e privacidade, e; (iv) desempenho e escalabilidade. As próximas subseções detalham as visões e perspectivas da IoT-A.

[1] Internet of Things Architecture (IoT-A): http://www.iot-a.eu/

3.3.2.1 Visão funcional

Conforme mostrado na Figura 3.2, a **visão funcional** definida no MAR do IoT-A possui nove grupos de funcionalidades (GFs), a saber: (i) aplicação; (ii) gerenciamento; (iii) organização de serviço; (iv) gerenciamento de processo de IoT; (v) entidade virtual; (vi) serviço IoT; (vii) segurança; (viii) comunicação, e; (ix) dispositivo. Cada um desses GFs envolve um ou mais componentes funcionais (CFs), representados Figura 3.2 por retângulos internos aos GFs. Todavia, apesar de a visão funcional descrever os CFs, ela não especifica as interações que ocorrem entre esses elementos pelo fato de tais interações serem tipicamente dependentes de escolhas de projetos, sendo portanto realizadas durante o desenvolvimento da arquitetura concreta. É importante observar que os GF de aplicação e de dispositivo estão fora do escopo da arquitetura de referência do IoT-A, enquanto que os GF de gerenciamento e de segurança são transversais aos demais GFs. Cada um dos GFs e respectivos CFs são apresentados a seguir.

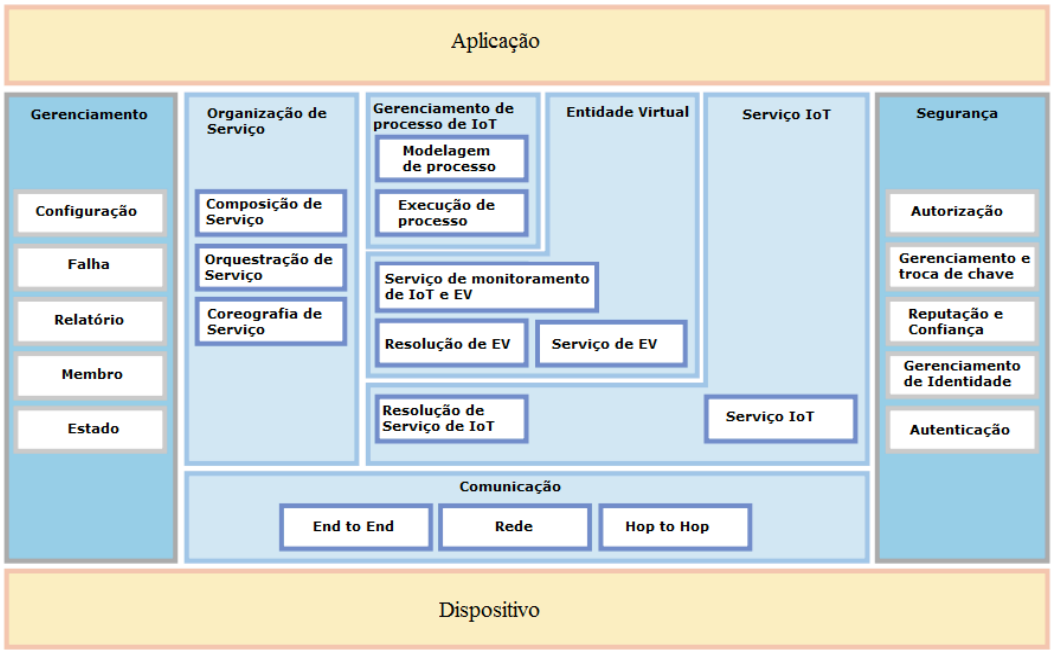

Figura 3.2. Ponto de vista de decomposição funcional da arquitetura de referência da IoT (adaptado de [Bassi *et al.* 2013, p. 168]).

3.3.2.1.1. Gerenciamento de processo de IoT

O GF Gerenciamento de Processo de IoT refere-se à integração do processo tradicional de gerenciamento de sistemas com o MAR IoT, tendo como objetivo global fornecer os conceitos funcionais e interfaces necessárias para ampliar processos tradicionais (de negócio) com os processos do mundo da IoT. Esse GF engloba dois CFs, a saber, (i) Modelagem de Processo e (ii) Execução de Processo.

O CF **Modelagem de Processo** fornece um ambiente para a modelagem de processos de negócios de IoT que serão serializados e executados no CF Execução de Processo e sua função principal é prover as ferramentas necessárias para modelar processos usando notações padronizadas, ou seja, usando novos conceitos de modelagem especificamente endereçando os comportamentos peculiares do ecossistema

de IoT. Por sua vez, o CF **Execução de Processo** executa os processos de IoT modelados no CF Modelagem de Processo utilizando os serviços de IoT orquestrados no GF Organização de Serviço. Esse CF é responsável por implantar os modelos de processos nos ambientes de execução, de modo que as atividades de um processo IoT são alocadas a ambientes apropriados de execução que realizam o processo de execução através da busca e invocação dos serviços de IoT apropriados. Para a execução adequada das aplicações, os requisitos de serviços de IoT devem ser resolvidos antes de os serviços de IoT específicos serem invocados. Para essa etapa, o CF Execução de Processo utiliza componentes do GF Organização de Serviço e, após selecionar os serviços de IoT adequados, os respectivos serviços são invocados. Assim, a próxima atividade do processo será executada com base no resultado da invocação do serviço.

3.3.2.1.2. Organização de Serviço

O GF Organização de Serviço atua como um ponto de comunicação central entre muitos outros GFs. Uma vez que o conceito primário de comunicação dentro da MAR IoT é a ideia de serviço, a Organização de Serviço é utilizada para compor e orquestrar serviços de diferentes níveis de abstração. Esse GF engloba três CFs: (i) Composição de Serviço, (ii) Orquestração de Serviço, e (iii) Coreografia de Serviço.

O CF **Composição de Serviço** determina os serviços que são compostos de serviços de IoT e de outros serviços a fim de criar serviços com funcionalidade estendida. Esse CF tem duas funções principais:

- *Suporte à composição de serviços flexíveis*. Para tal, o CF deve prover resolução dinâmica de serviços complexos, compostos por outros serviços. Esses serviços passíveis de serem combinados são escolhidos com base na sua disponibilidade e nos direitos de acesso do usuário solicitante.
- *Aumento da qualidade da informação*. A qualidade de informação pode ser aumentada combinando informação de múltiplas fontes. Por exemplo, um valor médio, com uma incerteza intrinsicamente menor, pode ser calculado com base na informação acessada através muitos recursos.

O CF **Orquestração de Serviço** é responsável por controlar e coordenar os serviços de IoT que são apropriados para atender requisições dos usuários ou do CF Execução de Processos (c.f. Seção 3.3.2.1.1). Se necessário, recursos temporários serão criados para armazenar resultados intermediários que alimentam o CF Composição de Serviço ou para o processamento de eventos complexos.

O CF **Coreografia de Serviço** provê um mediador (*broker*) que trata a comunicação publicação/subscrição entre serviços. Um serviço pode oferecer suas capacidades no CF e a função do mediador garante que um cliente interessado na oferta encontrará o serviço com as capacidades desejadas. Por sua vez, os consumidores de serviços podem registrar as suas requisições de serviços para o CF, mesmo que um serviço adequado não esteja disponível no momento em que a requisição foi emitida. Com isso, o consumidor do serviço será notificado tão logo o serviço que atenda a sua requisição estiver disponível.

3.3.2.1.3. Entidade Virtual

Entidades físicas são representadas no mundo digital por uma *entidade virtual* (EV). Há muitos tipos de representações digitais de entidades físicas, tais como modelos 3D, avatares, entradas de banco de dados e objetos (ou instâncias de uma classe em uma linguagem de programação orientada a objetos). No contexto de IoT, uma EV está associada a uma entidade física única, a qual ela representa. Embora haja geralmente apenas uma entidade física para cada EV, é possível que a mesma entidade física seja associada a várias EVs, e.g., uma representação diferente por domínio de aplicação. Cada EV deve ter um identificador unívoco e pode ser classificada como ativa ou passiva. Uma EV ativa refere-se a aplicações, agentes ou serviços que podem acessar outros serviços ou recursos, enquanto que uma EV passiva representa elementos de *software* passivos tais como entradas de banco de dados. Idealmente, EVs são representações sincronizadas de um determinado conjunto de aspectos (ou propriedades) da entidade física, ou seja, parâmetros digitais relevantes que representam as características da entidade física são atualizados imediatamente após qualquer alteração na entidade física. Da mesma forma, mudanças que afetam o mundo virtual também devem se manifestar na entidade física.

O GF Entidade Virtual consiste de três CFs: (i) Resolução de EV, (ii) Serviço de Monitoramento de IoT e EVs, e (iii) Serviço de EV. **Resolução de EV** é o CF que permite um usuário de IoT recuperar associações entre EVs e serviços de IoT, incluindo a descoberta de novas e dinâmicas associações entre EV e serviços associados. Caso não haja uma associação, ela pode ser criada. O usuário também pode continuamente inscrever-se ou cancelar uma inscrição para receber notificações sobre a descoberta de associação que se encaixam na especificação da EV ou do serviço. No caso de uma notificação, uma função de retorno será chamada. Similarmente, o usuário pode inscrever-se ou cancelar uma inscrição para notificações sobre procura de associações. Esse CF também permite procurar serviços de EV relacionados, como, por exemplo, pesquisar por serviços expondo serviços relacionados a uma EV. Finalmente, ele também permite gerenciar associações, tais como realizar inclusão, exclusão e atualização de associações entre a EV e os serviços de IoT que estão associados à EV.

O CF **Serviço de Monitoramento de IoT e EV** é responsável por encontrar automaticamente novas associações, que são então incluídas no CF Resolução de EV. Novas associações podem ser derivadas com base em associações existentes, descrições de serviços e informações sobre EVs. As funções desse CF são: (i) declarar associações estáticas, como, por exemplo, criar uma nova associação estática entre EVs e serviços descritos pela associação fornecida; (ii) descobrir associações dinâmicas, i.e., criar uma nova associação dinâmica ou monitorar associações entre EVs e serviços, e; (iii) atualizar e excluir associação do arcabouço de resolução de EVs.

Finalmente, o CF **Serviço de EV** manipula serviços de entidades. Um serviço de entidade representa um ponto de acesso global para uma entidade específica oferecendo meios para aprender e manipular o seu estado. O serviço fornece acesso a uma entidade via operações que habilitam leituras e/ou atualizações de valores dos seus atributos, operações essas que podem ser de somente leitura, somente escrita ou ambas. Além disso, um serviço específico de EV pode prover a funcionalidade de armazenamento de histórico para a publicação de informações integradas ao contexto, informação de estado de EVs e capacidades de EVs.

3.3.2.1.4. Serviço IoT

O GF Serviço IoT contém os serviços IoT e funcionalidades para descoberta, busca e resolução de nomes de serviços IoT. Ele consiste de dois CFs, (i) Serviço IoT e (ii) Resolução de Serviço de IoT. Um CF **Serviço IoT** expõe um recurso para torna-lo acessível a outras partes do sistema IoT. Tipicamente, serviços IoT podem ser usados para obter informações fornecidas por um recurso recuperado de um dispositivo sensor ou de um recurso de armazenamento conectado através de uma rede. Mais ainda, um serviço IoT pode ser usado para entregar informação para um recurso, controlar dispositivos atuadores ou mesmo configurar um recurso em termos de aspectos não funcionais tais como controle de acesso, disponibilidade e desempenho. Além disso, serviços IoT podem ser invocados de forma síncrona, para responder uma requisição do serviço, ou de forma assíncrona, para enviar notificações de acordo com subscrições previamente realizadas através do serviço. Um tipo particular de serviço IoT é o que armazena o histórico de recursos, que fornece capacidades de armazenamento para as medições geradas pelos recursos.

O CF **Resolução de Serviço IoT** fornece todas as funcionalidades necessárias para que um cliente seja capaz de procurar e invocar serviços IoT. Ele também fornece aos serviços a capacidade de gerenciar as suas descrições de serviços (tipicamente armazenadas como uma entrada no banco de dados). Assim, eles podem ser procurados e descobertos pelos clientes, que podem ser usuários humanos ou mesmo componentes de *software*. Descrições de serviços são identificadas por um identificador e contêm um localizador (*service locator*) que permite o acesso ao serviço. Tipicamente, essas descrições possuem informações adicionais, tais como a saída do serviço, o tipo de serviço ou a área geográfica para o qual o serviço é fornecido. O conteúdo exato, estrutura e representação dependem das escolhas de projeto realizadas, questão deixada em aberto no nível de arquitetura de referência. As funcionalidades oferecidas por esse CF são:

- *Descoberta*. Utilizado na procura de serviço IoT sem qualquer conhecimento prévio, tal como é realizado por um identificador de serviço. A procura é realizada através da execução de uma consulta (*query*) e a sua base é dependente da descrição do serviço, por exemplo, as saídas e o tipo do serviço e geolocalização.
- *Pesquisa*. Permite ao usuário acessar a descrição do serviço tendo conhecimento prévio sobre o identificador do serviço.
- *Resolução*. Determina os identificadores de serviço através dos quais o usuário pode contatar o serviço. A função de resolução reduz a quantidade de informações que devem ser comunicadas, especialmente se a descrição do serviço é grande e as informações contidas não são necessárias.
- *Gerenciamento de descrição de serviços*. Permite atualizar, incluir ou excluir descrição de serviços.

3.3.2.1.5. Comunicação

O GF de Comunicação é uma abstração que modela a variedade de esquemas de interação derivados das diferentes tecnologias pertencentes a sistemas IoT e fornece

uma interface comum para o GF Serviço IoT. Este GF consiste de três CFs, (i) Salto-a-Salto (Hop-to-Hop); (ii) Fim-a-Fim (End-to-End) e (iii) Rede.

O CF **Salto-a-Salto** fornece a primeira camada de abstração da tecnologia de comunicação dos dispositivos físicos, a qual permite o uso e a configuração de qualquer tecnologia da camada de enlace. A principal função desse CF é transmitir quadros (*frames*) do CF Rede e de dispositivos para o CF Salto-a-Salto. Os argumentos para a transmissão do quadro podem ser definidos e incluem, por exemplo, confiabilidade, integridade, criptografia e controle de acesso. Além disso, ele é responsável pelo roteamento do quadro. Finalmente, esse CF permite gerenciar a fila do quadro e definir o tamanho e prioridade das filas de entrada e saída de quadros. Essa função pode ser aproveitada para atender requisitos de Qualidade de Serviço (QoS).

O CF **Rede** permite a comunicação entre redes através de localizadores (endereçamento) e resolução de identificadores (IDs). Esse CF tem como função principal transmitir um pacote do CF Salto-a-Salto e do CF Fim-a-Fim para o próprio CF. Os argumentos para transmissão do pacote podem ser configurados e incluem, por exemplo, confiabilidade, integridade, criptografia, endereçamento *unicast/multicast* e controle de acesso. Mais ainda, esse CF inclui função de roteamento, que permite relacionar espaços de endereços de redes diferentes e diferentes tecnologias de rede, as quais podem ser convergidas através da tradução de protocolos de rede, por exemplo, de IPv4 para IPv6. Outra função é permitir a obtenção de um localizador a partir de um determinado ID, o que pode ser realizado internamente com base em uma tabela de pesquisa (*lookup*) ou de forma externa através de um *framework*. Finalmente, esse CF pode gerenciar filas de pacote e configurar o tamanho e prioridades das filas de entrada e saída de pacotes. Essa função pode ser aproveitada para atender requisitos de QoS.

O CF **Fim-a-Fim** é responsável por toda a abstração de comunicação fim-a-fim, envolvendo confiabilidade da transferência, transporte e funcionalidades de tradução, suporte a *proxies/gateways* e ajustes de parâmetros de configuração quando a comunicação cruza diferentes ambientes de redes. Esse componente é também responsável por transmitir uma mensagem do CF Rede e do Serviço IoT para o próprio CF. Os argumentos para mensagens podem ser configurados e incluem confiabilidade, integridade, criptografia, controle de acesso e multiplexação. Outras funções disponíveis são cache e *proxy* de mensagem, tradução de protocolos (que permite a tradução entre diferentes protocolos de comunicação fim-a-fim (e.g., HTTP/TCP para COAP/UDP). Uma última função é passar o contexto de protocolos de tradução entre *gateways*. O contexto pode estar relacionado com endereçamento, métodos específicos para um protocolo RESTful e credenciais de segurança.

3.3.2.1.6. Segurança

O GF Segurança é responsável por garantir a segurança e privacidade dos sistemas compatíveis com a IoT-A. Ele consiste de cinco CFs: (i) Autorização; (ii) Autenticação; (iii) Gerenciamento de Identidade; (iv) Gerenciamento de Troca de Chave, e; (v) Reputação e Confiança.

O CF **Autorização** é um *front-end* para gerenciar políticas de controle de acesso e executar decisões de controle de acesso baseadas nas políticas. Essa tomada de decisão pode ser invocada sempre que é requisitado o acesso a um recurso restrito, e.g., verificar se para um dado usuário é permitida a execução de uma pesquisa por um

recurso requisitado. Duas funcionalidades são oferecidas, a saber: (i) determinar se uma ação está autorizada ou não, com base na informação fornecida pela assertiva (informação que garante a ocorrência de uma autenticação de um cliente em um determinado momento utilizando um método particular de autenticação), na descrição do serviço e no tipo de ação, e (ii) gerenciar políticas, tal como adicionar, atualizar ou excluir uma política de acesso.

O CF **Autenticação** é responsável pela autenticação de serviços e usuários, verificação de credenciais fornecidas por um usuário e, caso estas sejam válidas, devolução de uma assertiva como resultado, requerida para usar o cliente do serviço de IoT. Além de verificar a exatidão das credenciais fornecidas por um novo nó que se junta ao sistema, ele estabelece contextos seguros entre este nó e as várias entidades em seu ambiente local. As duas funcionalidades fornecidas são (i) autenticar um usuário baseado na credencial fornecida e (ii) verificar se uma assertiva fornecida por um usuário é valida ou inválida.

O CF **Gerenciamento de Identidade** lida com questões de privacidade, emissão e gerenciamento de pseudônimos e informação extra para que componentes confiáveis possam operar (utilizar ou prover serviços) de forma anônima. Ele possui somente uma funcionalidade, que é criar uma identidade fictícia (identidade raiz, segunda identidade, pseudônimo ou identidade de grupo) junto com as credenciais de segurança relacionadas para usuários e serviços usarem durante o processo de autenticação. O CF **Gerenciamento** e **Troca de Chave** habilita comunicações seguras entre dois ou mais pares que inicialmente não se conhecem ou cuja interoperabilidade não esteja garantida, assegurando integridade e confidencialidade. Esse CF possui duas funcionalidades, (i) distribuir chaves de forma segura e (ii) registrar recursos de segurança.

Por fim, o CF **Reputação e Confiança** coleta a pontuação de reputação de usuários e calcula os níveis de confiança do serviço. Esse CF possui duas funcionalidades: (i) requisição de informação de reputação, que é uma função invocada por uma determinada entidade remota para requisitar informações de reputação sobre outra entidade, e; (ii) fornecimento de informação de reputação, que é uma função invocada por uma determinada entidade remota para prover informação de reputação (recomendações ou *feedback*) sobre outra entidade.

3.3.2.1.7. Gerenciamento

O GF Gerenciamento consiste de cinco CFs: (i) Configuração; (ii) Falha; (iii) Membro; (iv) Relatório; (v) Estado.

O CF **Configuração** é responsável por realizar funções de inicialização da configuração do sistema, tais como coletar e armazenar as configurações dos demais CFs e dispositivos. Esse CF também é responsável por rastrear as mudanças de configuração e realizar planejamento para futuras extensões do sistema.

O CF **Falha** tem como objetivo identificar, isolar, corrigir e registrar falhas que ocorrem no sistema de IoT. Para cada ocorrência de falha, uma notificação é enviada pelo CF correspondente para o CF de Falha, o qual reúne mais dados a fim de identificar a natureza e o grau do problema. Esse CF possui funções para tratar, monitorar e recuperar uma falha.

O CF **Membro** é responsável pelo gerenciamento de associações de membros do sistema IoT e informações importantes de qualquer entidade relevante (GF, CF, EV, serviço IoT, dispositivo, aplicações, usuário). Esse CF possui três funções principais: (i) monitoramento contínuo de membros; (ii) recuperação de membros, que permite recuperar membros do sistema obedecendo um determinado filtro e permite a subscrição para receber atualizações de entidades pertencentes a um determinado dono, e; (iii) atualização de membro, que permite atualizar metadados do membro no banco de dados de membros e registrar/cancelar o registro de metadados de membros no banco.

O CF **Relatório** permite refinar as informações fornecidas por outros CFs do GF, gerando relatórios ou recuperando relatórios de um histórico. Por exemplo, dentre muitos objetivos de um sistema de relatório, pode-se determinar a eficiência do sistema atual através da coleta e análise de dados de desempenho.

O CF **Estado** visa monitorar e fornecer os estados passado, presente e futuro do sistema IoT que são requeridos pelo CF Falha, possuindo como funções trocar ou aplicar um estado particular no sistema. Ele também permite verificar a consistência de comandos fornecidos para esta função, bem como verificar resultados previsíveis. Também monitora o estado, que é principalmente utilizada no modo de inscrição no qual se monitora o estado do sistema e se notifica os inscritos sobre mudanças de estados relevantes. Outras funções são prever o estado por um determinado tempo, recuperar o estado do sistema através de um histórico e realizar atualização de estado.

3.3.2.2. Visão da Informação

Objetos inteligentes conectados à IoT têm como propósito principal a troca de informações entre eles e também com sistemas externos. Entretanto, a forma como definir, estruturar, armazenar, processar, gerenciar e trocar as informações é muito importante. Dessa forma, a Visão da Informação permite visualizar estruturas de informações estáticas e fluxo de informações dinâmicas. Essa visão possui enfoque na descrição, tratamento e ciclo de vida da informação, bem como no fluxo de informações através do sistema e os componentes envolvidos. Sendo assim, nas próximas seções fornecemos um ponto de vista somente da modelagem de **Entidades Virtuais** (EV).

3.3.2.2.1 Descrição da informação

Descrição de EVs

EV é o conceito chave para qualquer sistema de IoT, no qual modelar a entidade física (objeto) é o ponto de interesse real. Uma EV tem um identificador, um tipo (*entityType*) e atributos que fornecem informação sobre a EV ou podem ser utilizados para trocar o seu estado, desencadeando um estímulo sobre a entidade física modelada. A modelagem do *entityType* é muito importante uma vez que este pode ser utilizado para determinar quais atributos uma instância de EV pode ter, definindo assim sua semântica. Um *entityType* pode ser modelado como um elemento simples, conforme pode-se observar na Figura 3.3, ou de forma hierárquica utilizando conceitos como herança de elementos, conforme apresentado na Figura 3.4.

Figura 3.3. Exemplo de um modelo simples de *entityType*.

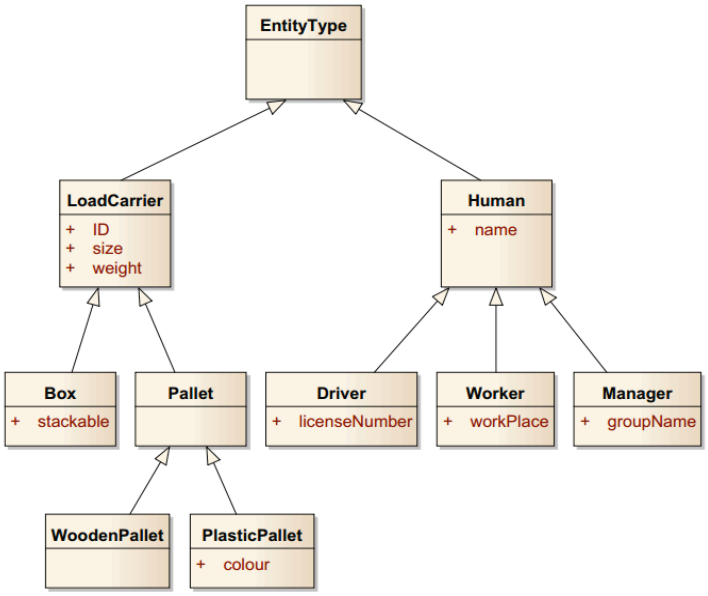

Figura 3.4. Exemplo de um modelo hierárquico de *entityType*.

EntityTypes são similares a classes na programação orientada a objetos. Sendo assim, diagramas de classes UML (*Unified Modeling Language*)[2], como o apresentado na Figura 3.3 e na Figura 3.4, podem ser utilizados para modelar *EntityTypes*. Em particular, a relação de generalização pode ser usada para modelar subclasses de *entityTypes*, conforme mostrado na Figura 3.4. Além da UML, linguagens de ontologias como a OWL (*Web Ontology Language*)[3] podem ser usadas de maneira alternativa, já que fornecem meios para modelar classes e subclasses.

Descrições de serviços

Serviços fornecem acesso a funções para recuperar informação ou executar tarefas de atuação nos dispositivos IoT e, portanto, precisam ser descritos apropriadamente. Essa tarefa é realizada através de **descrições de serviços** que contêm informações (tanto sintáticas quanto semânticas) sobre a interface do serviço. Por exemplo, informações de entrada, saídas, pré-condições necessárias e pós-condições. Além disso, a descrição do serviço pode incluir informação a respeito das, funcionalidades, ou sobre o dispositivo no qual o recurso está sendo executando (tais como sua plataforma de *hardware* ou sua localização geográfica, por exemplo). Apesar da existência de diferentes linguagens para realizar a descrição de um serviço, escolher uma linguagem para realizar essa tarefa não faz parte da definição da arquitetura de referência, e sim depende das escolhas realizadas durante o projeto.

[2] Unified Modeling Language (UML): http://www.omg.org/spec/UML/2.4.1/
[3] Web Ontology Language (OWL): http://www.w3.org/2001/sw/wiki/OWL

Associações entre EV e serviços

Serviços podem fornecer informações ou permitir atuação, mas porém eles podem não estar cientes de que tipos de informação uma EV pode fornecer ou qual o tipo de atuação ela permite, por exemplo. Esta informação pode ser então representada através de associações que relacionam uma EV a um serviço. Uma associação inclui o atributo da EV para que o serviço forneça a informação ou permita atuação como resultado de uma mudança de seu valor.

3.3.2.2.2. Manipulação da informação

Em um sistema de IoT, informações são manipuladas por serviços para fornecer acesso a recursos dos dispositivos como, por exemplo, recursos de sensores que tornam informações sobre o mundo físico acessíveis em tempo real aos sistemas. Outros serviços podem ainda processar e agregar informações fornecidas por outros serviços/recursos de IoT, derivando assim outras informações de maior valor agregado. Além disso, essas informações, sejam elas coletadas por serviços IoT ou adicionadas diretamente pelos usuários, podem ser armazenadas em uma classe especial de serviço IoT, o **Repositório de Histórico**.

Serviços IoT são registrados em sistemas IoT usando **descritores de serviços** que podem ser fornecidos por eles mesmos, pelos usuários ou por componentes especiais de gerenciamento que desejam fazer o serviço visível e detectável dentro de um sistema de IoT. O CF Resolução de Serviços IoT (c.f. Seção 3.3.2.1.4) é responsável por gerenciar e fornecer acesso a descrições de serviços por meio de uma interface de descoberta baseada nas especificações do serviço providas pelo solicitante, em termos de seu identificador ou localizador de serviço. Além disso, associações podem ser registradas através do CF Resolução de EV (c.f. Seção 3.3.2.1.3) por serviços que sabem para qual EV eles podem fornecer informação. O registro pode ser realizado pelos usuários ou por componentes especiais de gerenciamento.

3.3.2.2.3 Manipulação da informação por CFs

Esta subseção descreve como a informação é tratada e exposta por CFs em um sistema de IoT e apresenta o fluxo de informação entre eles (Figura 3.5). Nesse exemplo, a partir do dispositivo no nível atuador, a informação de temperatura é transferida para o serviço IoT e depois para o serviço EV. Finalmente, a partir do serviço de EV, o valor da temperatura é transferido para o aplicativo Android utilizando o padrão subscrição/notificação (*subscribe/notify*).

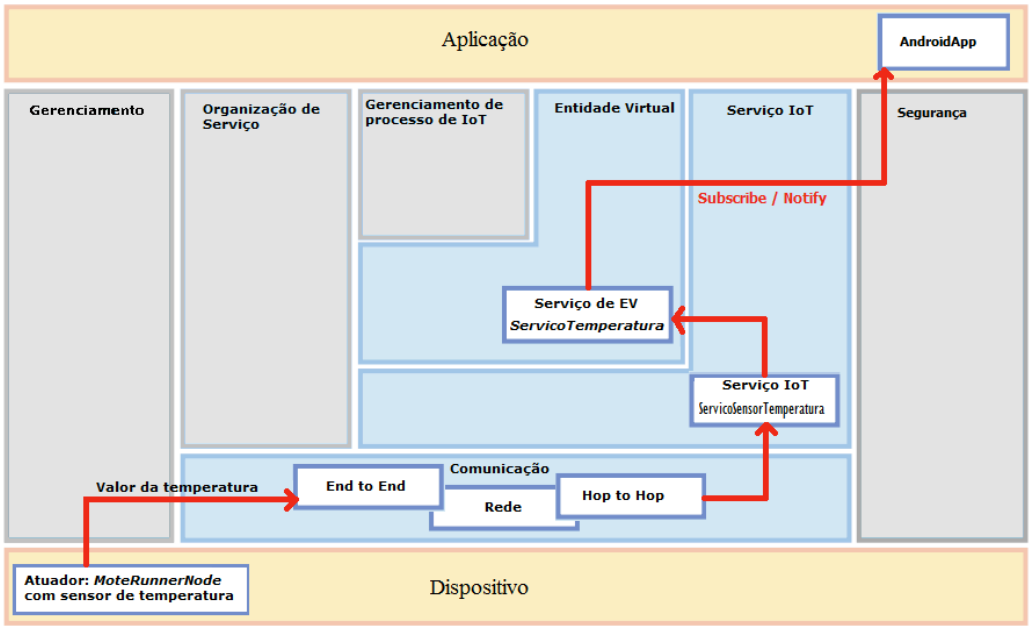

Figura 3.5. Exemplo de um fluxo de informação.

3.3.2.2.4 Ciclo de vida da informação

As informações fornecidas por recursos de um sensor são de natureza transiente e não podem ser medidas ou observadas sem uma requisição específica. A informação armazenada por um recurso de armazenamento pode ser permanentemente armazenada ou ter uma data de expiração na qual ela deve ser removida. Além disso, é possível adaptar a granularidade das informações armazenadas ao longo do tempo para que somente partes delas sejam mantidas e outras descartadas. A fim de evitar a manutenção de descrições para um serviço que não existe mais, um mecanismo de expiração de tempo (*timeout*) precisa ser desenvolvido na resolução de serviço de IoT. Após a expiração de tempo ser alcançada, a descrição do serviço deve automaticamente ser removida. Isso requer que os componentes que fornecem a descrição do serviço renovem o seu registro antes do tempo de expiração ser alcançado. O mesmo se aplica para associações armazenadas pelo CF Resolução de EV.

3.3.2.3. Visão de Operação e Implantação

Objetos inteligentes conectados na IoT podem ser realizados de muitas formas e podem se comunicar usando diferentes tecnologias. Além disso, diferentes sistemas podem precisar de comunicação compatível entre eles. Por isso, a Visão de Operação e Implantação é muito importante para tratar como sistemas atuais podem ser realizados pela seleção de tecnologias e fazê-los se comunicarem e operarem de uma forma abrangente, isto é, para alcançarem o maior número de sistemas.

O objetivo da Visão de Operação e Implantação é fornecer a usuários do MAR um conjunto de orientações para guia-los através de diferentes escolhas de projeto enquanto realizam as implementações reais de seus serviços. Todavia, uma completa análise de todas as possibilidades tecnológicas e suas combinações vai além do escopo desta visão, que identificará aquelas categorias que têm grande impacto na realização de sistemas de IoT. Em particular, iniciando a partir do Modelo de Domínio de IoT,

existem três grupos de elementos principais, mostrados na Figura 3.6: Dispositivos, Recursos, e Serviços. Cada um deles possui um problema diferente de implantação, que, por sua vez, se reflete nas capacidades operacionais do sistema.

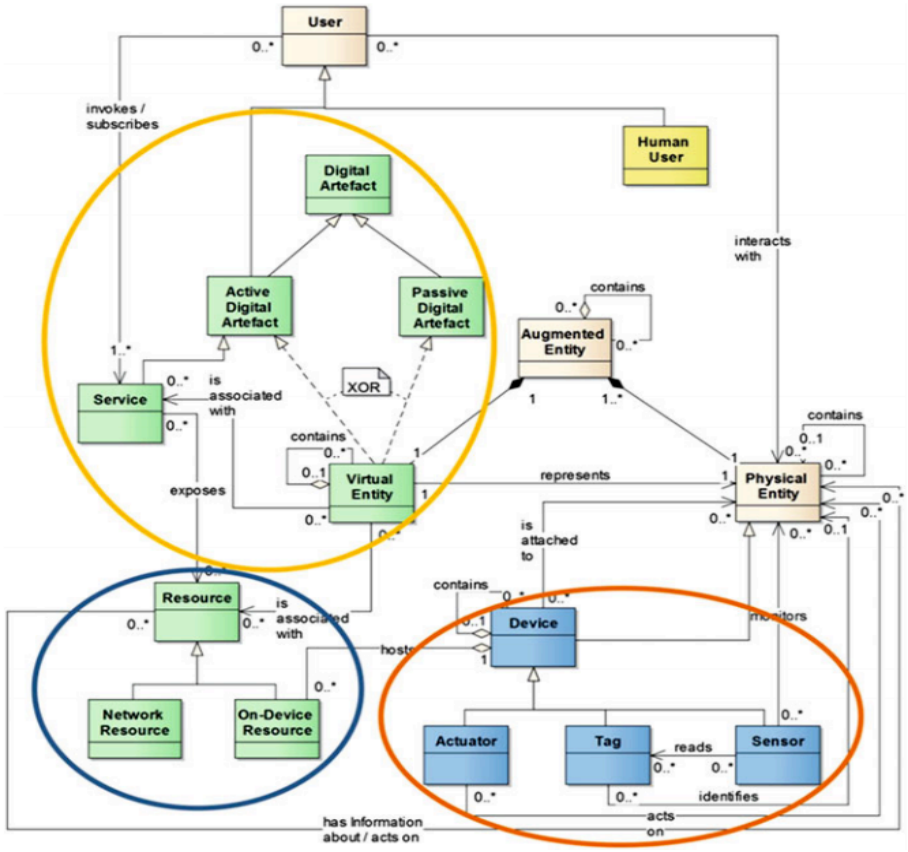

Figura 3.6. Elementos do modelo de domínio agrupados de acordo com seus aspectos comuns de implantação [Bassi *et al.* 2013, p. 17].

Os pontos de vista utilizados nesta visão são:

- **O Diagrama de Modelo de Domínio de IoT**, que é utilizado como uma diretriz para descrever uma aplicação específica de domínio. Dessa forma, diagramas da UML podem ser utilizados para detalhar ainda mais a interação entre os muitos elementos que compõem a aplicação alvo.
- **O Modelo Funcional**, que é utilizado como uma referência para a definição do sistema. Em particular, define-se grupos funcionais tais como serviços de IoT e grupos de conectividade, que são fundamentais para uma definição correta do sistema.
- **Diagramas de Conectividade de Rede**, que podem ser utilizados para planejar a topologia de conectividade para habilitar as capacidades de rede desejadas da aplicação alvo. No nível de implantação, o Digrama de Conectividade de Rede será utilizado para definir as hierarquias e o tipo de sub-redes compondo o sistema de rede completo.

- **Descrições de Dispositivos**, tais como planilhas e manuais de usuário, que podem ser utilizados para mapear o hardware real sobre os serviços e recursos requeridos do sistema alvo.

Dispositivos em sistemas de IoT incluem todo o espectro de tecnologias, variando da mais simples etiqueta de radiofrequência (RFIC) aos mais complexos servidores. As características unificadas são principalmente duas: por um lado, cada dispositivo é conectado com outro formando uma parte da IoT; por outro lado, cada dispositivo é "inteligente", mesmo com diferentes graus de complexidade. Essas duas características são assuntos da primeira escolha que o projetista do sistema tem de fazer. É importante notar que, para um dado dispositivo ser totalmente interoperável em um sistema em conformidade com a IoT-A, ele deve respeitar as definições funcionais do Modelo Funcional. Portanto, sistemas legados que não suportam totalmente tal Modelo Funcional podem implementar empacotadores (*wrappers*) e fazerem adaptações de *software* para tornarem-se compatíveis com o modelo.

Selecionar a complexidade computacional para um dado dispositivo é algo intrínseco para a aplicação alvo. Porém, escolher entre os diferentes tipos de conectividade não é tão simples, pois diferentes escolhas podem fornecer vantagens comparáveis. Pelo mesmo motivo, é possível perceber diferentes sistemas implementando a mesma ou diferentes aplicações a partir da Visão Funcional e que são extremamente diferentes da Visão de Operação e Implantação. Como uma consequência da coexistência de diferentes tecnologias de comunicação no mesmo sistema, a segunda escolha que o projetista do sistema deve considerar está relacionada aos protocolos de comunicação. Em particular, funcionalidades de conectividade para sistemas IoT são definidas na GF de Comunicação (c.f. Seção 3.3.2.1.5). Além disso, para melhor entender a aplicação, é importante descreve-la usando a Visão Funcional. Embora o MAR do IoT-A sugira um conjunto de protocolos de comunicação visando a interoperabilidade entre diferentes tecnologias utilizando o IP como denominador comum, o projetista do sistema pode ser forçado a fazer escolhas de baixa qualidade. Em particular, são identificadas as seguintes possibilidades:

- **Conjunto de protocolos padronizados para IoT.** Esta é a principal direção indicada pelo projeto IoT-A e fornece a melhor solução para interoperabilidade.
- **Soluções *ad-hoc* proprietárias**. Sempre que os requisitos de desempenho da aplicação forem mais importantes que a versatilidade do sistema, soluções *ad-hoc* podem ser a única direção a seguir.
- **Outros padrões**. Dependendo do domínio da aplicação alvo, podem existir regulações forçando o projetista do sistema a adotar padrões diferentes daqueles sugeridos pelo conjunto de protocolos de IoT por questões de compatibilidade e continuidade.

Depois de selecionados os dispositivos e seus métodos de comunicação, o projetista do sistema deve analisar os serviços e recursos, tal como definido na Seção 3.3.2.1.4 que descreve o GF Serviço de IoT. Tanto no caso de recursos e quanto para serviços, o ponto chave é escolher onde implantar o *software* relacionado com um dado dispositivo. As opções são as seguintes:

- **Objetos inteligentes**. Esta escolha aplica-se à definição de recursos e serviços leves tais como serviços Web que podem ser realizados em algumas dezenas ou centenas de *bytes*, por exemplo.
- ***Gateways***. Sempre que os dispositivos alvo não são poderosos o suficiente para eles mesmos executarem o *software*, *gateways* ou outros dispositivos mais poderosos devem ser implantados para auxiliar dispositivos com menores capacidades.
- **Nuvens computacionais.** Um sistema IoT pode também ser implantado em nuvens computacionais. Apesar de esta solução aumentar a disponibilidade dos serviços, ela pode diminuir o desempenho em termos de latência e taxa de transferência.

Deve-se notar que tal escolha tem de ser feita pelo tipo do recurso e serviço e depende do dispositivo relacionado. Por exemplo, um sensor de temperatura pode ser implantado em um dispositivo sem fio restrito, capaz de hospedar os recursos de temperatura com um serviço simples para fornecê-lo. Contudo, se um serviço mais complexo for necessário, o *software* tem que ser implantado em um dispositivo com maiores capacidades computacionais.

Na mesma linha, é importante selecionar onde armazenar a informação coletada pelo sistema. Em tal escolha, um projetista deve levar em consideração a sensibilidade (por exemplo, se o *software* é capaz ou não de executar a segurança do *framework*), a necessidade de disponibilidade dos dados e o grau de redundância necessária à resiliência de dados. As opções previstas são as seguintes:

- **Somente local**. O dado é armazenado somente no dispositivo que o produziu. Neste caso, a localidade dos dados é imposta e o sistema não requer um complexo banco de dados distribuído. Todavia, dependendo da localização de uma requisição, a resposta pode levar muito tempo para ser entregue e, no pior cenário, pode ser perdida.
- **Somente Web**. Nenhuma cópia local é mantida pelo dispositivo, de modo que tão logo os dados sejam enviados para o servidor, eles são armazenados em banco de dados.
- **Local com *cache* Web**. Uma estrutura hierárquica para armazenar dados é mantida a partir dos dispositivos até os servidores de banco de dados.

Finalmente, uma das características principais de sistemas de IoT é a resolução de serviços e entidades, o que é fornecido pelos CFs Resolução de Serviço e Entidade, respectivamente. Esta escolha tem somente duas opções, que são:

- **Implantação interna**. O modulo principal do CF é instalado nos servidores pertencentes ao sistema e é dedicado para a aplicação alvo ou compartilhado entre diferentes aplicações do mesmo provedor.
- **Uso externo**. O motor principal é fornecido por terceiros e o projetista deve construir APIs de acesso a esse serviço.

Diferentemente de outras escolhas, esta é dirigida pelo custo associado para a manutenção do módulo principal do sistema. Uma vez que ele é um componente crítico

do sistema, segurança, disponibilidade e robustez devem ser assegurados. Portanto, para pequenas empresas, a solução mais fácil é a de uso externo.

3.3.2.4. Perspectivas

O MAR do IoT-A propõe as seguintes perspectivas para sistemas de IoT: (i) evolução e interoperabilidade, (ii) disponibilidade e resiliência, (iii) confiança, segurança e privacidade, e (iv) desempenho e escalabilidade. Para descrever as perspectivas, segue-se uma estrutura sugerida por Rozanski e Wood (2011) apud Bassi *et al*. (2013), porém ajustada de acordo com as necessidades do ambiente IoT. Cada perspectiva contém as seguintes informações:

- **Qualidade desejada**. A propriedade de qualidade que a perspectiva está tratando, por exemplo, desempenho, segurança ou escalabilidade.
- **Requisitos**[4]. Os requisitos de IoT que a perspectiva trata.
- **Aplicabilidade**. Aplicabilidade da perspectiva, isto é, os tipos de sistemas aos quais a perspectiva é aplicada.
- **Atividades**. Um conjunto de atividades possíveis que são sugeridas para atingir a qualidade desejada.
- **Tática**. Lista de táticas arquiteturais que um arquiteto pode usar quando projeta um sistema. Uma tática arquitetural é uma decisão de projeto para realizar objetivos de qualidade no nível arquitetural.

3.3.3. WSO2

Como abordado nas seções anteriores, a grande variedade de requisitos e dispositivos a serem suportados na IoT tem como resultado uma arquitetura que não é simples de conceber e lidar. Entretanto, uma arquitetura modular e escalável que permita adicionar e remover capacidades, bem como dar suporte a uma vasta quantidade de requisitos, é útil e valiosa. Nesse contexto, esta seção aborda a arquitetura de referência de IoT proposta pela WSO2 [Fremantle 2014]. Essa arquitetura inclui os dispositivos bem as arquiteturas do lado servidor e arquiteturas de nuvem necessária para interagir com e gerenciar os dispositivos. O objetivo é fornecer aos arquitetos e desenvolvedores um ponto de partida eficaz que contemple a maior parte dos requisitos de sistemas e projetos envolvendo IoT. Contudo, não é o foco desta seção detalhar o funcionamento de uma arquitetura em particular de *hardware* cliente/servidor e/ou computação em nuvem, uma vez que essa proposta de arquitetura de referência é independente de fornecedor e não é específica para um conjunto de tecnologias.

A arquitetura de referência da WSO2, ilustrada na Figura 3.7, consiste de um conjunto de camadas no qual cada camada executa uma determinada função claramente definida. As camadas previstas são: (i) comunicações externas; (ii) processamento de eventos e análises; (iii) camada de agregação/barramento; (iv) comunicações entre dispositivos, e; (v) camada de dispositivos. Há também duas camadas transversais/verticais, que são (i) gerenciamento de dispositivo e (ii) gerenciamento de acesso e identidade. Cada uma dessas camadas pode ser instanciada utilizando

[4] Uma lista com tais requisitos pode ser encontrada em: http://www.iot-a.eu/public/public-documents/d6.2-updated-requirements-list.

tecnologias específicas. A seguir, são discutidas opções para a implementação de cada uma delas.

3.3.3.1. Camada de Dispositivos

A camada mais inferior da arquitetura é a **Camada de Dispositivos** (*Devices*). Os dispositivos podem ser de diversos tipos, mas para que sejam considerados dispositivos de IoT, eles devem ter alguma comunicação com a Internet, seja de forma direta ou indireta. Conexões diretas são aquelas em que os dispositivos possuem em seu *hardware* um componente para conexão nativa com a Internet, enquanto conexões indiretas requerem dispositivos auxiliares para estabelecerem uma conexão com a Internet. Como exemplos de dispositivos com conexões diretas, pode-se citar: (i) Arduino, com conexão Ethernet; (ii) Arduino Yun, com conexão WiFi; (iii) Raspberry Pi, que pode ser conectado via Ethernet ou WiFi, e; (iv) Intel Galileo, que também pode ser conectado via Ethernet ou WiFi. Como conexões indiretas, pode-se citar: (i) dispositivos ZigBee conectados através de um *gateway* ZigBee; (ii) dispositivos Bluetooth ou *Bluetooth* de baixa potência conectados via telefone celular, e; (iii) dispositivos de comunicação via rádio de baixa potência para um Raspberry Pi.

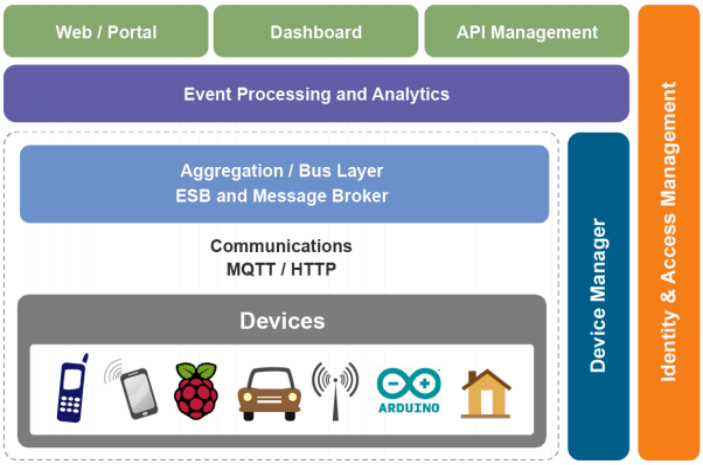

Figura 3.7. Arquitetura de referência para IoT da WSO2 [Fremantle 2014, p.9].

Além da capacidade de conexão, o dispositivo necessita de uma identidade única, que pode ser: (i) um identificador único (UUID) gravado permanentemente dentro do dispositivo; (ii) UUID provido por um subsistema rádio, por exemplo, identificador Bluetooth ou endereço MAC WiFi; (iii) *OAuth2*[5] *Refresh/Bearer Token*, que pode ser utilizado de maneira adicional a um dos dois anteriores, e; (iv) identificador armazenado em uma memória não-volátil tal como uma EEPROM. Para esta arquitetura de referência, recomenda-se que cada dispositivo possua um UUID, de preferência um ID fixo, não modificável e provido por *hardware*, bem como um *OAuth2 Refresh/Bearer Token* armazenado em memória EEPROM. O objetivo do *OAuth2 token* é prover um *token* de identidade segura separado do núcleo imutável de identidade de cada dispositivo. Por sua vez, o *Bearer Token* é utilizado inicialmente e passado para qualquer servidor ou serviço que precisa de identificação, porém ele possui um tempo de vida menor que o *Refresh Token*. Se o *Bearer Token* expirar, o

[5] OAuth: http://oauth.net/

Refresh Token é passado para a Camada de Identificação, e esta cria um *Bearer Token* atualizado. Apesar de especificação ser baseada sobre o protocolo HTTP (*Hypertext Transfer Protocol*)[6], a arquitetura de referência também provê suporte a esses fluxos utilizando o protocolo MQTT (*Message Queue Telemetry Transport*)[7].

3.3.3.2. Camada de Comunicações

A **Camada de Comunicações** (*Communications*) suporta a conectividade dos dispositivos. Há muitos protocolos potenciais para a comunicação entre os dispositivos e arquitetura de rede. Os três protocolos mais conhecidos são: (i) HTTP e seu uso sobre o estilo arquitetural REST [Fielding 2000]; (ii) MQTT, e; (iii) CoAP (*Constrained Application Protocol*)[8].

O protocolo de aplicação HTTP é bem conhecido e existem muitas bibliotecas que o implementam. É um protocolo baseado em texto e mesmo pequenos dispositivos, como os controladores de 8 *bits*, podem suporta-lo parcialmente, por exemplo, implementando código suficiente para realizar operações POST ou GET sobre um recurso. Os dispositivos baseados em 32 bits podem utilizar bibliotecas clientes com suporte total ao HTTP que implementam apropriadamente o protocolo inteiro.

Há muitos protocolos otimizados para uso em IoT, sendo que os dois mais conhecidos são o MQTT e CoAP. O MQTT é um sistema de publicação-subscrição de mensagens baseado em um modelo de *broker* e projetado para fluxos sobre TCP. O protocolo tem uma baixa sobrecarga (*overhead* em torno de 2 *bytes* por mensagem) e foi projetado para suportar redes com perdas e intermitentemente conectadas. Além disso, há uma especificação projetada para uso em redes de sensores estilo Zigbee chamada MQTT-SN. Por sua vez, o CoAP é um protocolo de camada de aplicação projetado para prover uma solução RESTful (i.e., baseada no estilo arquitetural REST – *REpresentational State Transfer*) [Fielding 2000] e, portanto, modelado seguindo a semântica HTTP, porém bem mais enxuto e com uma abordagem binária em vez de baseada em texto. O CoAP adota uma abordagem tradicional estilo cliente-servidor em vez de uma abordagem de *brokers* e foi projetado para ser utilizado sobre UDP (*User Datagram Protocol*)[9].

Para a arquitetura de referência da WSO2, optou-se por selecionar o MQTT como protocolo preferencialmente usado na Camada de Comunicação e o HTTP como uma opção alternativa. As razões para selecionar o MQTT em detrimento do CoAP são: (i) maior adoção; (ii) biblioteca de suporte mais ampla; (iii) uma ponte simplificada para a coleta de eventos existentes e sistemas de processamento de eventos, e; (iv) conexão simples sobre *firewalls* e redes NAT. Embora ambos os protocolos tenham suas vantagens e desvantagens específicas, haverá situações nas quais o CoAP pode ser preferível e poderá ser utilizado. Entretanto, para ter suporte ao protocolo MQTT, é necessário ter um *broker* MQTT na arquitetura, bem como bibliotecas de dispositivos.

Um importante aspecto com dispositivos IoT não está relacionado somente ao o fato de o dispositivo enviar dados para o servidor/nuvem, mas também com a comunicação na direção oposta. Isso é um dos benefícios da especificação MQTT:

---

[6] Hypertext Transfer Protocol – HTTP/1.1: http://www.w3.org/Protocols/rfc2616/rfc2616.html
[7] Message Queue Telemetry Transport: http://mqtt.org/
[8] Constrained Application Protocol: http://tools.ietf.org/html/draft-ietf-core-coap-18
[9] User Datagram Protocol – UDP: http://tools.ietf.org/html/rfc768

como este é um modelo de *broker*, clientes conectam com uma conexão de saída para o *broker,* quer o dispositivo esteja ou não atuando como um publicador (*Publisher*) ou como assinante (*subscriber*). Isso usualmente evita problemas com *firewall* porque esta abordagem funciona até mesmo atrás de *firewalls* ou via NAT. No caso de a comunicação principal ser baseada em HTTP, a abordagem tradicional de envio de dados para o dispositivo seria usar *HTTP Polling*, porém, tal abordagem é muito ineficiente e custosa, tanto em termos de tráfego de rede como em requisitos de energia. O substituto moderno para isso é o protocolo *WebSocket*[10], que permite uma conexão HTTP ser atualizada em uma conexão totalmente bidirecional. Tal conexão atua então como um *socket channel* (similar a um canal TCP puro) entre o servidor e o cliente. Uma vez que tenha sido estabelecida, cabe ao sistema escolher um protocolo em andamento para o túnel de conexão. Para a arquitetura de referência da WSO2, novamente recomenda-se o uso de MQTT como um protocolo com *WebSockets*. É ainda importante observar que apesar de existir algum suporte para *WebSocket* sobre pequenos controladores (a exemplo do Arduino[11]), a combinação de código de rede, HTTP e *WebSockets* utilizaria a maior parte do espaço de código disponível sobre um típico dispositivo Arduino 8 *bits*. Portanto, o uso de *WebSockets* é recomendado somente sobre dispositivos de 32 *bits* ou maiores.

3.3.3.3. A Camada de Agregação/Barramento

Uma importante camada da arquitetura de referência é a **Camada de Agregação/Barramento** (*Aggregation/Bus Layer*), que agrega e gerencia as conexões. Essa camada é importante por três motivos principais: (i) capacidade para suportar um servidor HTTP e/ou um *broker* MQTT para comunicação com os dispositivos; (ii) capacidade para agregar e combinar comunicações de diferentes dispositivos e rotear as comunicações para um dispositivo específico, possivelmente via um *gateway*, e; (iii) capacidade para realizar mapeamentos e conversões entre diferentes protocolos, e.g., para oferecer APIs baseadas em HTTP mediadas em uma mensagem MQTT indo para o dispositivo. Finalmente, esta camada precisa executar dois papéis chaves de segurança: ela deve ser capaz de atuar com um servidor de recursos OAuth2 (validando *Bearer Tokens* e escopos de acesso a recursos associados), bem como ser capaz de atuar como um ponto de aplicação de políticas (PEP) para acessos baseados em políticas. A camada de gerenciamento de acessos e identidade atua como um ponto de decisões políticas (PDP) neste processo. A camada de *barramento* implementa então os resultados dessas chamadas para o PDP a fim de permitir ou negar acesso a recursos.

3.3.3.4. Camada de Processamento de Eventos e Análise

A **Camada de Processamento de Eventos e Análise** (*Event Processing and Analytics*) obtém os eventos do barramento e provê capacidades para processar e agir sobre estes eventos. Uma capacidade principal dessa camada consiste em armazenar o dado em de um banco de dados, o que pode ocorrer de três formas. O modelo tradicional seria escrever uma aplicação do lado servidor, a exemplo de uma aplicação RESTful apoiada por um banco de dados. Entretanto, existem abordagens mais ágeis. A primeira é usar uma plataforma analítica de Big Data [Chen *et al*. 2014] que consiste em uma

[10] The WebSocket Protocol: http://tools.ietf.org/html/rfc6455
[11] Arduino: http://arduino.cc/

plataforma de nuvem escalável com suporte a tecnologias como Apache Hadoop[12] para prover análises baseadas em *map-reduce* sobre os dados provenientes dos dispositivos. A segunda abordagem é suportar processamento de eventos complexos para executar atividades de forma próxima a tempo real e ações baseadas nos dados dos dispositivos e do resto do sistema. A recomendação seria então fazer uso das seguintes abordagens: (i) armazenamento de dados altamente escalável para armazenar eventos; (ii) *map-reduce* para execução em *batch* (de longa duração) de processamento de dados, ou; (iii) processamento de eventos complexos para rápido processamento em memória e próxima a tempo real e ações autonômicas baseadas nos dados e atividades de dispositivos e outros sistemas. Além disso, a camada pode suportar plataformas de processamento de aplicações tradicionais tais como JavaBeans[13], *beans* dirigidos a mensagem, ou alternativas como node.js, PHP, Ruby ou Python.

3.3.3.5. Camada de Comunicações Externas

A arquitetura de referência precisa prover um modo que possibilite o sistema de IoT comunicar-se com o meio externo e vice-versa. Isto pode incluir três abordagens principais: (i) capacidade de criar páginas e portais Web (*Web portals*) que interajam com dispositivos e com a camada de processamento de eventos; (ii) capacidade de criar painéis de monitoramento (*Dashboards*) que ofereçam visões para análise e processamento de eventos, e; (iii) capacidade de interagir com sistemas fora dessa rede usando comunicações máquina-a-máquina (APIs). A abordagem recomendada para construir um *front-end* Web é utilizar uma arquitetura modular tal como um portal que permite compor interfaces de usuário de forma rápida e simples. Certamente a arquitetura também suporta tecnologias Web existentes do lado servidor como *Java Servelets/JSP*, *Php*, *Python*, *Ruby,* etc. A abordagem recomendada é baseada no arcabouço Java e no mais popular servidor web baseado em Java, o *Apache Tomcat*. O *dashboard* é um sistema reutilizável focado na criação de gráficos e outras visualizações de dados vindos dos dispositivos e da camada de processamento de eventos.

A Camada de Gerenciamento de API (*API Management*) provê três funções principais: (i) um portal focado no desenvolvedor (de forma antagônica ao portal focado no usuário) através do qual desenvolvedores podem encontrar, explorar e utilizar APIs do sistema; (ii) um *gateway* que gerencia o acesso às APIs, executando a verificação do controle de acesso (para requisições externas) e otimizando o uso baseado nas políticas definidas, além de executar funções de roteamento e balanceamento de carga, e; (iii) um *gateway* que publique os dados na camada de análise, onde são armazenados e processados para fornecer visões sobre como as APIs são utilizadas.

3.3.3.6. Gerenciamento de Dispositivos

O gerenciamento de dispositivos é endereçado por dois componentes, (i) **Gerente de Dispositivo** (*GD*) e, (ii) Agente de Gerenciamento de Dispositivo (AGD). O GD é um sistema do lado servidor que comunica-se com os dispositivos através de vários protocolos e provê controle tanto individual como em massa de dispositivos. Ele também gerencia remotamente o *software* e a implantação de aplicações no dispositivo e pode bloquear e/ou limpar o dispositivo se necessário. O AGD é um conjunto de

[12] Apache Hadoop: http://hadoop.apache.org/
[13] JavaBeans: http://en.wikipedia.org/wiki/JavaBeans

componentes genérico que fornece gerenciamento de dispositivos e utilitários como: (i) adaptadores de comunicação para HTTP e MQTT, (ii) inscrição de dispositivos, (iii) gerenciamento de token e, (iv) tipo de plataforma para gerenciar.

O GD trabalha em conjunto com o AGD e há muitos diferentes agentes para diferentes tipos de plataformas e dispositivos. O GD também precisa manter a lista de identidade dos dispositivos e mapeá-los para os respectivos proprietários. Ele também deve trabalhar com a Camada de Gerenciamento de Acesso e Identidade para gerenciar o controle de acesso sobre os dispositivos.

Há três níveis de dispositivos, a saber, (i) não-gerenciados, (ii) semigerenciados, e (iii) totalmente gerenciados. Os dispositivos totalmente gerenciados são aqueles que executam o agente completo de Gerenciamento de Dispositivos (GD). O agente GD oferece suporte a gerenciar o *software* instalado, habilitar/desabilitar funções dos dispositivos, gerenciar controles de segurança e identificadores, monitorar a disponibilidade do dispositivo, manter um registro da localização do dispositivo, se disponível, bloquear ou limpar o dispositivo remotamente se ele estiver comprometido, entre outros. Dispositivos não-gerenciados podem se comunicar com o resto da rede, mas não há agente envolvido. Dispositivos semigerenciados são aqueles que implementam algumas partes do GD, por exemplo, tais como o controle de características, mas não o gerenciamento de *software*.

3.3.3.7. Gerenciamento de Identidade e Acesso

A última camada é a de **Gerenciamento de Identidade e Acesso** (*Identity and Access Management*). Ela precisa fornecer os seguintes serviços: (i) emissão e validação de *token* OAuth2; (ii) outros serviços de identidade incluindo suporte a SAML2 SSO[14] e OpenID Connect[15] para identificar requisições de entrada da camada Web; (iii) XACML PDP[16]; (iv) diretório de usuários, e; (v) gerenciamento de políticas para controle de acesso. Esta camada pode ainda ter outros requisitos específicos para cada gerenciador de acesso e identidade para uma dada instanciação da arquitetura de referência.

3.4. Plataformas de *Middleware* para IoT

Como citado anteriormente, o paradigma de IoT ainda possui diversos desafios em aberto que requerem soluções em nível de *hardware* e de *software*. Em particular, faz-se necessária uma camada de *software* que forneça abstrações para dispositivos e aplicações, diversos níveis de transparência e interoperabilidade, bem como múltiplos serviços para usuários finais e aplicações. Essa camada de *software*, denominada *middleware*, oculta dos desenvolvedores de aplicações as complexidades e heterogeneidades referentes ao *hardware* subjacente, às camadas de protocolos de rede, às plataformas e dependências do sistema operacional, além de facilitar o gerenciamento de recursos do sistema e aumentar a previsibilidade da execução de aplicações [Bernstein 1996].

---

[14] Security Assertion Markup Language (SAML2): https://saml2.codeplex.com/
[15] OpenID: http://openid.net/
[16] Extensible Access Control Markup Language (XACML): https://www.oasis-open.org/committees/xacml/

No contexto de IoT, uma plataforma de *middleware* representa um artefato de *software* residindo entre a camada de aplicação e a infraestrutura de suporte (comunicação, processamento, sensoriamento), fornecendo acesso padronizado aos dados e serviços providos pelos objetos inteligentes através de interfaces de alto nível, além de promover o reuso de serviços genéricos, que podem ser compostos e configurados para facilitar o desenvolvimento de aplicações de forma mais eficiente para o ambiente de IoT. O desenvolvimento de plataformas de *middleware* para IoT tem atraído cada vez mais a atenção da comunidade acadêmica e da indústria. Por conta disso, diversas pesquisas já foram reportadas na literatura visando a concepção e a implementação de plataformas de *middleware* abordando os requisitos mencionados anteriormente. As subseções a seguir apresentam algumas propostas de plataformas de *middleware* para IoT e como tais plataformas procuram atender aos requisitos levantados na Seção 3.2.

3.4.1 EcoDiF

A EcoDiF (*Ecossistema Web de Dispositivos Físicos*) é uma proposta de solução a alguns dos desafios para a materialização do paradigma de 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 [Delicato *et al*. 2013b]. Dessa forma, a EcoDiF integra dispositivos físicos heterogêneos e os conecta à Internet, fornecendo funcionalidades de controle, visualização, processamento e armazenamento de dados em tempo real. A plataforma alinha-se com o paradigma de Web das Coisas (*Web of Things – WoT*, em Inglês) ao utilizar tecnologias e protocolos da Web, como o protocolo HTTP e URIs (*Uniform Resource Identifier*) [Guinard e Trifa 2009]. Dessa forma, a EcoDiF possibilita a inclusão de dispositivos físicos no meio digital de modo que seus dados e serviços sejam utilizados em diferentes aplicações como qualquer recurso da Web, alavancando assim a concretização da visão da IoT [Delicato *et al*. 2013a].

A EcoDiF foi projetada tendo como base princípios REST [Fielding 2000], utilizando assim o HTTP como mecanismo padrão de suporte a todas as interações entre os objetos endereçáveis através de suas principais operações (GET, PUT, POST, DELETE). Dessa forma, a plataforma provê uma interface bem definida para expor as funcionalidades dos objetos na Web, provendo a padronização e a simplificação do processo de desenvolvimento de aplicações, além de minimizar barreiras no tocante à incompatibilidade entre diferentes fabricantes, protocolos proprietários e formatos de dados [Pautasso *et al*. 2008]. Desse modo, a plataforma soluciona parte das questões de interoperabilidade de dispositivos, requisito essencial para plataformas de *middleware* no contexto do paradigma de IoT, e consolida a integração de dispositivos físicos heterogêneos. Assim, a EcoDiF é capaz de oferecer um acesso unificado a dados e serviços providos por dispositivos integrados através de interfaces de alto nível, bem como contribuir para tornar mais fácil o desenvolvimento de aplicações em ambientes de IoT altamente distribuídos e heterogêneos. A arquitetura lógica da plataforma é composta de diversos módulos, ilustrados na Figura 3.8 e apresentados a seguir.

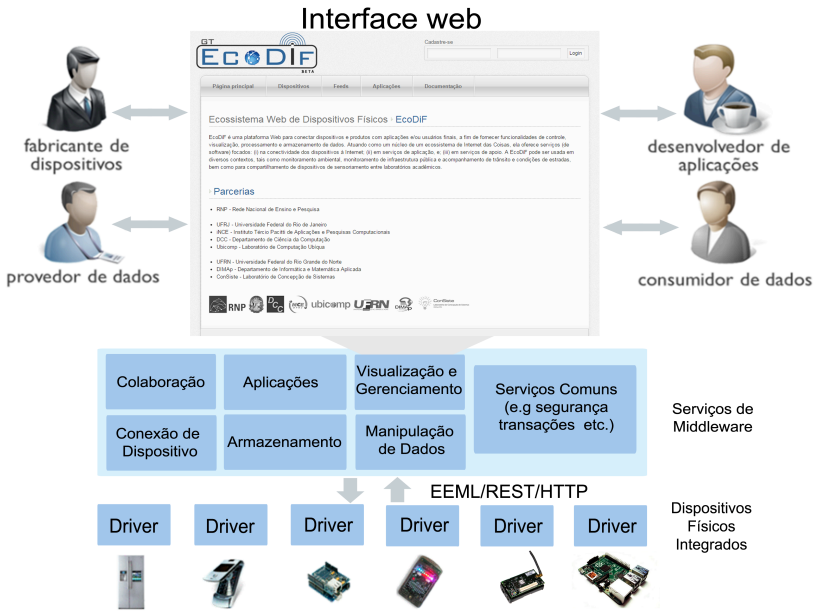

Figura 3.8. Arquitetura da EcoDiF.

O **Módulo de Conexão de Dispositivos** visa facilitar a conexão de dispositivos físicos à IoT, de acordo com a API da EcoDiF e fazendo uso de *drivers customizados*, desenvolvidos para cada tipo específico de plataforma de dispositivo. Esses *drivers* desempenham um papel de suma importância em relação à integração de dispositivos com a EcoDiF visto que a heterogeneidade de tais dispositivos é abstraída dos usuários e aplicações que fazem uso dos dados providos por eles. Os *drivers* da EcoDiF são construídos com base em princípios REST e padrões e protocolos da Web. Com isso, o HTTP não é utilizado apenas como protocolo de comunicação, mas também para oferecer suporte a todas as interações com os dispositivos interconectados, que são visualizados como recursos Web. Além disso, o uso do HTTP elimina barreiras no tocante à incompatibilidade entre diferentes fabricantes, protocolos proprietários e formatos de dados [Pautasso *et al*. 2008]. Na EcoDiF existem dois tipos de *drivers*, *ativos* e *passivos*. *Drivers* ativos obtêm os dados coletados por dispositivos fazendo requisições periódicas à API do dispositivo mesmo se os valores dos dados permanecem inalterados. Por sua vez, *drivers* passivos (ou baseados em eventos) aguardam por notificações da API do dispositivo, notificações essas disparadas sempre que há mudanças nos valores dos dados. Após obter dados dos dispositivos (os chamados *feeds*), os *drivers* estruturam tais dados utilizando a linguagem EEML (*Extended Environments Markup Language*)[17], usada para descrever dados obtidos por dispositivos em um dado contexto (ambiente). EEML é baseada na linguagem XML (*eXtended Markup Language*), padrão para representação de informações na Web, permitindo que dados obtidos por dispositivos sejam representados de modo simples e estruturada. Após a sua representação em EEML, tais dados são enviados à EcoDiF através de requisições HTTP PUT a fim de serem registrados na plataforma pelo **Módulo de Manipulação de Dados**.

O **Módulo de Visualização e Gerenciamento** provê uma interface Web para permitir que usuários gerenciem os dispositivos conectados à EcoDiF. Através dessa

[17] Extended Environments Markup Language (EEML): http://www.eeml.org/

interface, usuários podem monitorar o estado e localização de seus dispositivos, bem como visualizar dados históricos armazenados na plataforma. Além disso, os usuários podem criar *triggers*, que são mecanismos baseados em eventos para enviar notificações com base em condições predefinidas com relação aos valores atuais dos *feeds*. Por sua vez, o **Módulo de Colaboração** visa facilitar a colaboração entre usuários da EcoDiF, permitindo realizar buscas por dispositivos e aplicações a partir de seus respectivos metadados (tipo, usuário, localização, etc.) através da interface Web.

O **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. É importante destacar que esses repositórios podem fazer uso de uma infraestrutura de computação em nuvem para armazenar dados e arquivos, provendo assim atributos de qualidade como robustez, confiabilidade, disponibilidade e escalabilidade [Gao *et al*. 2011, Soldatos *et al*. 2012]. Já o **Módulo de Serviços Comuns** envolve serviços de infraestrutura providos pela plataforma, tais como segurança (autenticidade, confidencialidade, integridade), gerenciamento de ciclo de vida de aplicações, transações, etc.

O **Módulo de Aplicações** provê um modelo e um ambiente para programação e execução de aplicações que fazem uso dos dados (*feeds*) disponíveis na EcoDiF e geram novas informações que também podem ser disponibilizadas pela plataforma para serem usadas por outros *feeds* e aplicações mais complexas. Essas aplicações são construídas como *mashups* [Guinard *et al*. 2009], aplicações *ad-hoc* criadas a partir da composição de diferentes tipos de informação providas por diferentes fontes, tais como serviços Web e bases de dados relacionais. Por exemplo, considere-se um sensor que monitora a temperatura de uma dada localidade, e um usuário deseja combinar essa informação com um mapa que informa a localização das medidas coletadas. Dessa forma, uma única aplicação *mashup* pode compor tais informações de temperatura e localização. O modelo de programação e execução adotado pela EcoDiF é baseado no uso da linguagem EMML (*Enterprise Mashup Markup Language*)[18], uma linguagem declarativa aberta que tem sido usada como padrão para o desenvolvimento de *mashups*. Aplicações *mashup* são implementadas como *scripts* escritos em EMML e executadas em um motor (*engine*) de execução que processa tais *scripts*.

A EcoDiF foi implementada utilizando a linguagem de programação Java e implantada em um servidor de aplicações JBoss[19], 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. Usuários podem acessar as funcionalidades da EcoDiF através de uma interface Web provida pelo *Módulo de Gerenciamento e Visualização* e implementada com as tecnologias abertas JavaServer Faces (JSF)[20] e Primefaces[21]. Uma vez que a conexão entre a EcoDiF e os dispositivos integrados é habilitada pelos respectivos *drivers* (via o *Módulo de Conexão de Dispositivos*), os dados obtidos dos

[18] Enterprise Mashup Markup Language (EMML): http://en.wikipedia.org/wiki/Enterprise_Mashup_Markup_Language
[19] JBoss: http://www.jboss.org
[20] JavaServer Faces Technology (JSF): http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[21] Primefaces: http://www.primefaces.org

dispositivos são enviados à plataforma através de requisições HTTP PUT, provendo assim uma interface RESTful para os clientes (sejam humanos ou aplicações). Para dar suporte a tal abordagem RESTful, foi adotada a implementação RESTEasy[22] para o estilo arquitetural REST e a Java API for RESTful Web Services (JAX-RS)[23]. Os dados (*feeds*) produzidos pelos dispositivos são estruturados utilizando o protocolo EEML, baseado na linguagem XML, e são tratados pelo *Módulo de Manipulação de Dados* para serem efetivamente registrados em uma base de dados relacional MySQL utilizando as especificações da Java Persistence API (JPA)[24] implementadas no *framework* Hibernate[25]. Desenvolvedores de aplicações podem construir aplicações *mashup* fazendo uso dos dados providos pelos dispositivos (*feeds*) que são integrados à EcoDiF através do *Módulo de Aplicações*. Na EcoDiF, aplicações são implementadas usando a linguagem EMML, de modo que quando um *feed* é vinculado a uma aplicação, o *Módulo de Aplicações* automaticamente o inclui como uma variável de entrada no *script* EMML para a aplicação, e os usuários podem então codificar a lógica de programação. Por outro lado, usuários que possuem conhecimento acerca da linguagem EMML podem escrever seus próprios *scripts* (fora do ambiente da EcoDiF, em um editor de sua escolha) e então importa-los para a plataforma. As aplicações registradas na EcoDiF são executadas utilizando o motor de execução EMML também implantado no servidor de aplicações JBoss e seus respectivos resultados são exibidos para os usuários através interface Web da EcoDiF. Por fim, o *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)[26] implementadas no servidor de aplicações JBoss), gerenciamento do ciclo de vida de aplicações, transações, etc.

3.4.2 Xively

A plataforma Xively[27] utiliza serviços de nuvem para gerenciar dados providos por dispositivos. A plataforma fornece uma API para envio de dados a partir dos sensores, permitindo assim a visualização de dados históricos e provendo mecanismos para disparar eventos com base nos dados gerados pelos sensores (os chamados *triggers*). Assim como a EcoDiF, a Xively foi projetada tendo como base princípios REST e padrões da Web, como HTTP e URIs. Dessa forma, a plataforma fornece interfaces bem definidas e padronizadas, minimizando os problemas de incompatibilidade entre os diferentes dispositivos. Na Xively, os dados são organizados em *feeds, datapoints* e *datastreams*. Um *feed* representa os dados de um ambiente (uma sala, por exemplo) com seus respectivos *datastreams*, que representam dados enviados por um determinado sensor nesse ambiente (por exemplo, temperaturas do ambiente monitorado), enquanto um *datapoint* representa um único valor de um *datastream* em um determinado instante de tempo.

---

[22] RESTEasy: http://www.jboss.org/resteasy
[23] Java API for RESTful Services (JAX-RS): https://jax-rs-spec.java.net/
[24] Java Persistence API: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
[25] Hibernate: http://www.hibernate.org
[26] Java Authentication and Authorization Service (JAAS): http://www.oracle.com/technetwork/java/javase/jaas/index.html
[27] Xively – Business solutions for the Internet of Things: http://xively.com

Por se tratar de uma solução comercial e de código fechado, não há detalhes quanto à arquitetura dessa plataforma além do exposto na Figura 3.9. Contudo, sabe-se que há três formas de se recuperar dados aferidos pelos dispositivos conectados à Xively: (i) sensores enviam dados para a plataforma nos formatos JSON, XML (especificamente EEML) ou CSV usando a API REST; (ii) através de *sockets*, e; (iii) através do protocolo MQTT. No primeiro caso, através dos métodos implementados pelo HTTP, o método GET é utilizado por um programa cliente para recuperar dados de um *feed* ou *datastreams*, enquanto o método PUT é utilizado nos dispositivos conectados para o envio de dados. No segundo caso, um *socket* pode ser criado de modo a evitar o *overhead* de abrir e fechar comunicações HTTP em condições nas quais muitos dados são trocados. Além disso, os dois primeiros casos permitem a utilização do protocolo SSL/TLS a fim de prover autenticação e criptografia dos dados. Por fim, o terceiro caso utiliza os recursos de *publish* e *subscribe* do protocolo MQTT para o envio e a recuperação de dados a partir dos dispositivos.

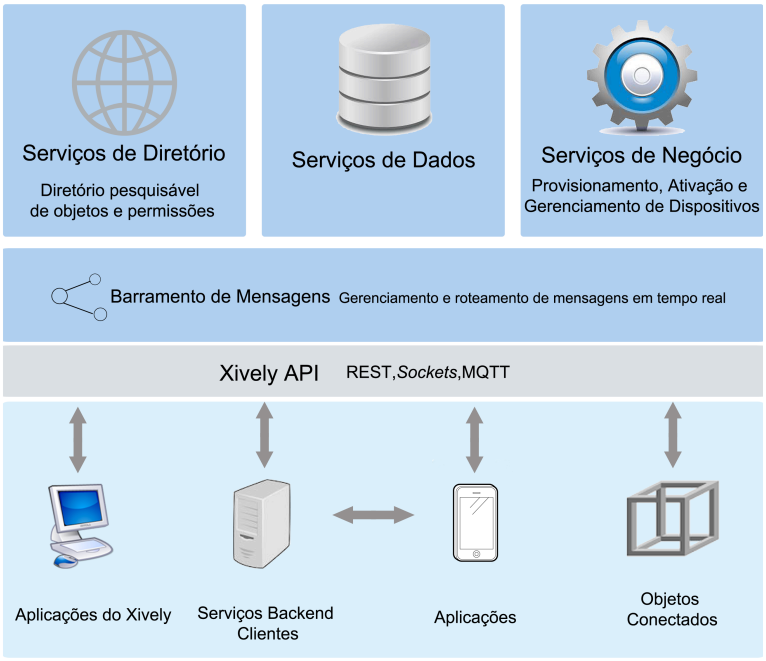

Figura 3.9. Arquitetura da Xively.

3.4.3 Carriots

A Carriots[28] também é uma plataforma de *middleware* para IoT que utiliza serviços de nuvem para gerenciar dados providos por dispositivos, além de conectar dispositivos a dispositivos e a outros sistemas. 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 armazenar 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.

Como ilustrado pela Figura 3.10, as entidades da plataforma possuem uma hierarquia bem definida. Dessa forma, todos os dispositivos conectados ao *middleware*

[28] Carriots: http://www.carriots.com

são associados a serviços (e.g., dispositivos físicos ou algum outro recurso) e todos os serviços pertencem a um projeto. Por exemplo, sensores de estacionamento, independente do seu contexto de utilização (como localização geográfica distinta), pertencem ao mesmo serviço dentro de um projeto. 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 vinculadas à ele. Além disso, *triggers* nessa plataforma são associados à exportação dos dados dos serviços (ao contrário da EcoDiF e da Xively, nas quais eles são associados a aplicações e a *datastreams*). Do ponto de vista dos dispositivos, eventos tais como o recebimento ou a persistência de dados podem ser monitorados por *listeners*, entidades responsáveis por executarem análise e processamento em caso de alguma condição pré-configurada ter sido atendida.

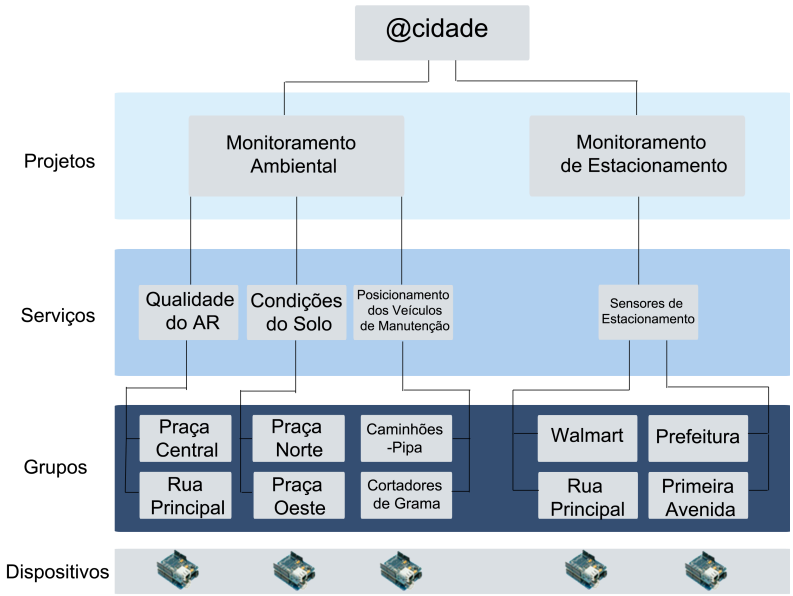

Figura 3.10. Exemplo de hierarquia das entidades da plataforma Carriots.

Como mostra a Figura 3.11, a arquitetura lógica da Carriots é formada pelos seguintes módulos: (i) a API REST; (ii) *Big Data*; (iii) Gerenciamento de Projetos e Dispositivos; (iv) Regras de Negócio e Processamento de Eventos; (v) Segurança; (vi) *Logs* e *Debug*; (vii) Painel de Controle, e; (viii) Módulo de Comunicação Externa.

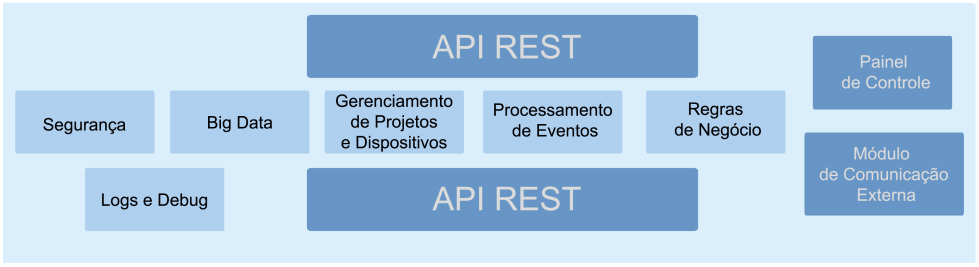

Figura 3.11. Arquitetura da plataforma Carriots.

Os dados trocados entre os dispositivos, os sistemas conectados e a plataforma podem ser representados de duas formas distintas: (i) sensores enviam dados nos formatos JSON ou XML (em um formato particular da plataforma) usando a **API**

REST, ou (ii) através do protocolo de mensagens **MQTT**. O módulo de **Big Data** provê às aplicações a flexibilidade de gerenciar os dados dos 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.

O módulo de **Gerenciamento de Projetos e Dispositivos** contém os projetos criados pelos usuários e é responsável por atualizações do *software* embarcado nos dispositivos e por suas configurações. O armazenamento e a execução de eventos em forma de *scripts* criados utilizando a linguagem de programação Groovy[29] e fazendo uso de regras do tipo *if-then-else*, são responsabilidade do módulo de **Regras de Negócio e Processamento de Eventos**.

A **segurança** é tratada de quatro formas, a saber: (i) através do uso de chaves pré-compartilhadas para a definição de privilégios de acesso; (ii) através da utilização do protocolo HTTPS para a encriptação dos dados enviados pela rede; (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.

O módulo ***Logs* e *Debug*** fornece um console para depuração de erros e registro de mensagens, tornando tais dados acessíveis a partir do painel de controle. Por sua vez, o módulo **Painel de Controle** permite o gerenciamento pelo usuário de todos os outros módulos e recursos da plataforma. Por fim, o 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).

3.4.4 LinkSmart

A LinkSmart (anteriormente chamado Hydra)[30] é uma plataforma de *middleware* baseada em Arquitetura Orientada a Serviços (SOA – *Service-Oriented Architecture*, em Inglês) para IoT que oferece suporte ao desenvolvimento de aplicações formadas por dispositivos físicos heterogêneos que operam com recursos limitados em termos de poder computacional, energia e memória. Ela oferece interfaces de serviços Web para controle de qualquer tipo de dispositivo físico e permite que desenvolvedores incorporem dispositivos físicos heterogêneos em suas aplicações. 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 descrição semântica dos dispositivos através do uso de ontologias de dispositivos é uma das características mais importantes dessa plataforma. Responsável por representar todas as meta-informações sobre os dispositivos, a ontologia usada é baseada na ontologia de dispositivos FIPA (*Foundation for Intelligent Physical Agents*)[31] e permite a parametrização semântica para incluir informações dos dispositivos, como seus recursos de segurança. A LinkSmart contém ainda o módulo *Application Ontology Manager*, que provê uma interface para uso da ontologia de

[29] The Groovy Programming Language: http://groovy-lang.org/
[30] LinkSmart: https://linksmart.eu
[31] Foundation for Intelligent Physical Agents (FIPA): http://www.fipa.org/specs/fipa00091/index.html

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 dispositivo 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. Por fim, a plataforma distingue dispositivos com recursos restritos (*non LinkSmart-enabled device*), que não são capazes de hospedá-la (mas são capazes de se comunicar através de *gateways*), e dispositivos mais poderosos (*LinkSmart-enabled devices*) que são capazes de hospedá-la.

3.4.5 OpenIoT

A plataforma de *middleware* OpenIoT [Soldatos *et al*. 2012] tem por objetivo ser uma camada de suporte para aplicações em IoT usando um modelo baseado em infraestrutura de nuvem. Os recursos IoT podem ser acessados por serviços sob demanda que seguem o modelo de computação em nuvem, de modo que, por exemplo, sensoriamento pode ser um serviço disponibilizado na nuvem (*Sensing as a Service*), e através do uso de serviços de nuvem para IoT, usuários podem configurar, implementar e usar a IoT. O projeto foca na convergência entre IoT e computação em nuvem, visando assim fornecer uma "nuvem de coisas" (*cloud of things*, em Inglês). Para tal, *middleware* tem como proposta permitir a customização de ambientes de nuvens auto-organizáveis para aplicações IoT, habilitando a implantação por provedores de serviços de infraestruturas de nuvem que forneçam serviços IoT a usuários.

A OpenIoT conecta sensores com o ambiente de nuvem, de forma que os recursos da nuvem poderão ser usados para processamento e gerenciamento de dados, funções especialmente úteis para processamento intensivo de sinais que, em geral, não podem ser realizadas em infraestrutura de IoT devido aos recursos limitados dos dispositivos. Para permitir interoperabilidade entre os vários objetos, a OpenIoT usa a ontologia SSN (*Semantic Sensor Network*)[32], que serve como base para especificação de solicitações dos serviços combinando sensores, fluxos de dados (*data streams*) e suas propriedades.

A arquitetura da OpenIoT (Figura 3.12) é composta por três planos lógicos distintos: (i) Utilidade/Aplicação; (ii) Virtualizado; e (iii) Físico. Tais planos, por sua vez, são compostos por sete módulos principais, apresentados a seguir.

[32] Semantic Sensor Network Ontology (SSN): http://www.w3.org/2005/Incubator/ssn/ssnx/ssn

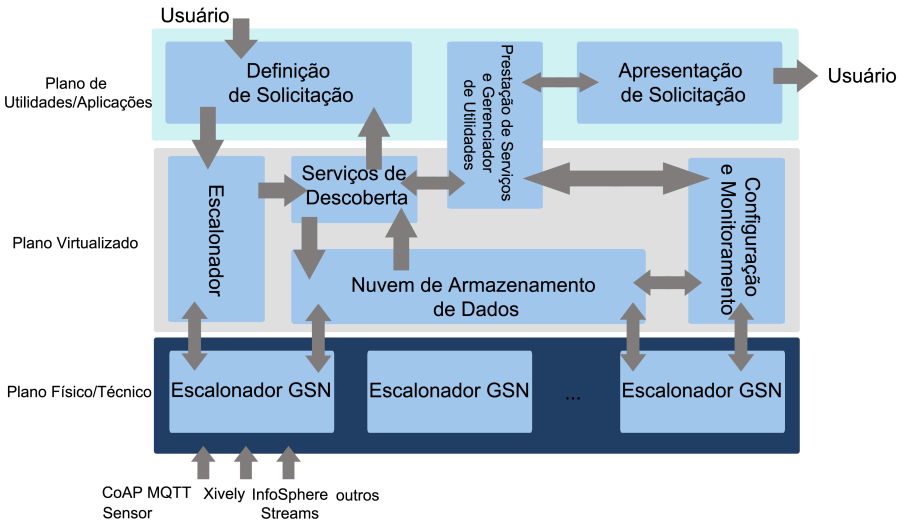

Figura 3.12. Arquitetura da plataforma de *middleware* OpenIoT.

O plano **Utilidades/Aplicações** é composto por três módulos: (i) Definição de Solicitação; (ii) Apresentação de Solicitação; e (iii) Configuração e Monitoramento. Através de uma interface Web, o primeiro módulo permite definir solicitações de serviços para a plataforma em tempo de execução, solicitações essas que são enviadas ao Escalonador. O módulo **Apresentação de Solicitação** permite, também via interface Web, que o usuário selecione *mashups* de um repositório. O módulo de **Configuração e Monitoramento** permite o gerenciamento e a configuração dos dispositivos conectados e dos serviços em execução na plataforma.

O plano **Virtualizado** também é composto por três módulos: (i) Escalonador; (ii) Nuvem de Armazenamento de Dados, e; (iii) Prestação de Serviços e Gerenciador de Recursos. O **Escalonador** processa todas as requisições para serviços originadas no módulo **Definição de Solicitação** e garante acesso aos recursos necessários. A **Nuvem de Armazenamento de Dados** tem como objetivo armazenar os fluxos de dados originados dos dispositivos conectados bem como os metadados necessários para o funcionamento da plataforma. Para tal, uma versão adaptada do *middleware* Linked Stream [Le-Phuoc *et al*. 2012] é utilizada. O módulo de **Prestação de Serviços & Gerenciador de Recursos** tem o papel de combinar fluxos de dados para serem entregues aos serviços solicitantes e realizar a contabilidade e o faturamento dos recursos da plataforma. Por fim, o plano **Físico** é composto apenas pelo módulo **Sensor Middleware**, responsável por coletar, filtrar, combinar e anotar semanticamente fluxos de dados originários tanto de dispositivos físicos quanto virtuais. Para tal, ele faz uso de uma versão adaptada do *middleware* GSN (*Global Sensor Network*)[33], que fornece uma API RESTful para a interoperação com os dispositivos, bem como alguns recursos de segurança (e.g., autenticação e permissão de acesso através da definição de papeis).

3.4.6 RestThing

Assim como a EcoDiF, RestThing [Qin *et al*. 2011] é uma infraestrutura de serviços Web baseada em REST cujo objetivo é ocultar a heterogeneidade de dispositivos e prover um modo de integrar dispositivos em aplicações Web. A plataforma visa permitir

[33] Global Sensor Network (GSN): https://github.com/LSIR/gsn/wiki

que desenvolvedores criem aplicações usando princípios REST, combinando recursos físicos e Web, de modo que dispositivos e informações Web são ambos representados como recursos e manipulados por uma interface uniforme no estilo REST.

Como ilustra a Figura 3.13, os elementos do RestThing são: (i) aplicações; (ii) API RESTful; (iii) provedor de serviço; (iv) camada de adaptação; (v) dispositivos embutidos, e; (vi) recursos Web. A API RESTful permite transmitir dados entre os sensores que usam IP, os gateways, o servidor Web e aplicações Web. O RestThing trabalha com três tipos de formatos de dados, a saber JSON, XML e CSV. Para compartilhamento de dados é usada a linguagem EEML, que descreve a estrutura dos dados. Para acesso aos objetos RESTful, são utilizadas as operações do protocolo HTTP: o método GET é utilizado para recuperar o estado corrente do objeto, o método PUT é utilizado para modificar o estado corrente do objeto, o método POST é utilizado para criar um novo objeto; DELETE para remover um objeto e, adicionalmente, o método LIST, que permite obter todos os objetos conectados à plataforma.

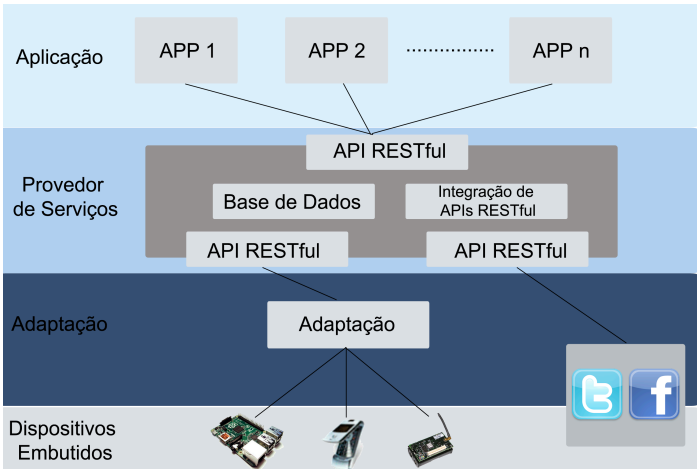

Figura 3.13. Infraestrutura do RestThing.

3.4.7 WoT Enabler

A WoT Enabler [Gao *et al*. 2011], ilustrada na Figura 3.14, é uma plataforma baseada em REST para a integração de sensores e compartilhamento de seus dados. Implementada utilizando a linguagem de programação Ruby, seu servidor se comunica com uma base de dados conectada à arquitetura, responsável pelo armazenamento dos dados os quais são enviados diretamente pelos dispositivos conectados (para dispositivos dotados de conexão direta à Internet) ou por *gateways* (para dispositivos que não possuem tal capacidade). Assim, tais dados podem ser acessados por aplicações e usuários através de requisições HTTP RESTful ao servidor WoT Enabler, que pode retornar representações de dados nos formatos XML, JSON ou CSV. De modo similar às plataformas EcoDiF, Xively e RestThing, a arquitetura também usa EEML para a estruturação dos dados provenientes dos sensores, diferindo apenas na hierarquia de dados utilizada: (i) um *system* representa uma coleção de dados referentes a um determinado ambiente; (ii) um *sensor* representa um sensor individual no contexto de um *system* e; (iii) um *data* é um par <*chave, valor*> que se refere ao valor de uma medida aferida por um sensor em um dado instante de tempo.

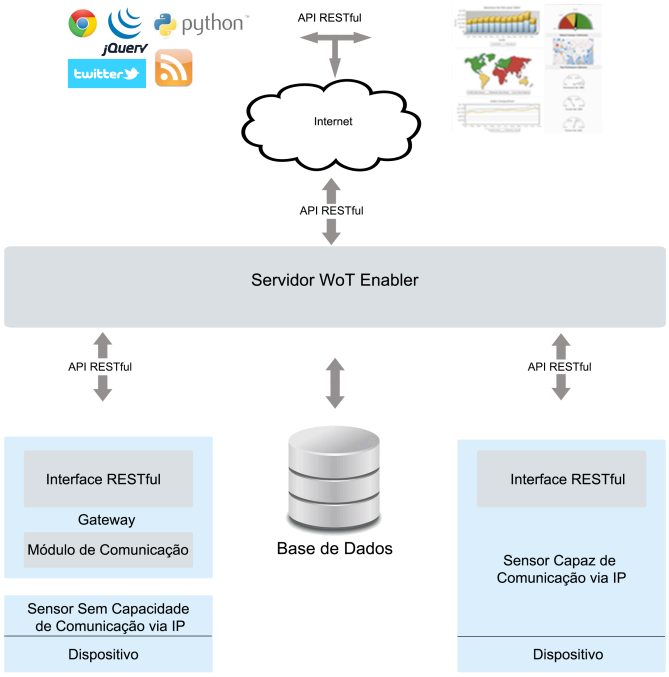

Figura 3.14. Arquitetura da plataforma WoT Enabler [Gao *et al.* 2011].

3.4.8 S^{3}OIA

S^{3}OIA (*Smart Spaces and Smart Objects Interoperability Architecture*) [Vega-Barbas *et al*. 2012] é uma arquitetura orientada a serviço para integração de diferentes tipos de objetos, físicos ou virtuais usando *tuplespace* para expressar semanticamente informações dos dispositivos integrados pela plataforma e possibilitar a representação destes. Seus módulos são divididos em cinco grupos de módulos funcionais: (i) Descoberta de Serviços e Dispositivos, (ii) Exposição de *Web Services* e *Triple Spaces*, (iii) Repositório de Serviços e Resolução de Dependências, (iv) Interface de Interação, e (v) Composição, Tolerância a Falhas e Dependências Distantes.

O grupo **Descoberta de Serviços e Dispositivos** é composto pelos módulos responsáveis pela integração e pela abstração de todos os dispositivos, não importando o protocolo de comunicação usado pelos mesmos. Por exemplo, através desse grupo, é possível que um Arduino se comunique com qualquer dispositivo que suporte o protocolo UPnP (*Universal Plug and Play*). Para tal, os *gateways* de comunicação da plataforma abstraem qualquer heterogeneidade ao oferecer uma interface comum de comunicação, servindo de *proxies* para os protocolos envolvidos na troca de dados.

O grupo **Exposição de *Web Services* e *Triple Spaces*** traz a computação distribuída baseada em *tuplespaces* para sistemas ubíquos, de forma que dispositivos heterogêneos possam compartilhar informações assincronamente. A computação Triple Space realiza comunicação baseada em espaços de tuplas usando triplas RDF.

O grupo **Repositório de Serviços e Resolução de Dependências** gerencia os serviços disponíveis dentro de um *Smart Space* e resolve as dependências extraídas da aplicação. Além disso, ele possui um módulo gerenciador de eventos que segue o paradigma *publish/subscribe*. 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 grupo **Interface de Interação** gerencia a interação entre os seres humanos e a plataforma. O principal objetivo desse grupo é recuperar informações sobre as interações entre dois atores (chamadas de *intents*) e adequar os objetos já implantados às necessidades dos usuários. Por fim, o grupo de **Composição, Tolerância a Falhas e Dependências Distantes** é responsável por tratar da composição e orquestração de serviços. Além disso, esses módulos gerenciam o acesso aos recursos da plataforma quando dois ou mais objetos requisitam-nos simultaneamente.

3.4.9 Ubiware

A Ubiware [Nagy *et al*. 2009] é uma plataforma de *middleware* baseada em três requisitos: automação, integração e interoperabilidade. A solução proposta incorpora princípios de sistemas multi-agentes, entidades computacionais com comportamento autônomo que facilitam o desenvolvimento de sistemas complexos. A integração se apresenta como um segundo desafio a ser tratado. Para tal, a Ubiware se utiliza dos princípios de Web Semântica para permitir que tanto humanos quanto máquinas possam entender e processar uma informação. No que tange à interoperabilidade, a proposta se baseia em suportar o máximo número de protocolos e dispositivos possível, por não acreditar no estabelecimento de novos padrões.

Para atingir os objetivos estabelecidos, a proposta foca no Ubicore, o núcleo da plataforma de *middleware* proposta. Esse componente provê a todos os objetos conectados a possibilidade de serem inteligentes ao conectá-los a um agente de *software*. Dessa forma, tais objetos ganham recursos de comunicação, autocontrole e auto-monitoramento através da utilização do conhecimento e das funcionalidades previamente adquiridos a partir de eventos tanto próprios quanto externos. Já com relação à ciência de contexto, o fato de que cada objeto tem um agente vinculado a ele implica que tal agente tem pleno conhecimento acerca de seu estado. Dessa forma, tal conhecimento pode ser trocado com outros agentes e utilizado para melhorar a execução de outros objetos vinculados à plataforma. Essa característica também influencia em como os dados são armazenados pela plataforma. Ao invés de um repositório de dados centralizados, a plataforma adota uma armazenagem de informações distribuída, na qual os agentes trocam informações entre si e mineram os dados recebidos a fim de encontrar as informações necessárias para sua utilização.

3.4.10 WSO2

Considerada uma instância da arquitetura de referência descrita na Seção 3.3.3, a plataforma WSO2 [Fremantle 2014] pode ser considerada uma arquitetura modular e versátil, permitindo o seu uso em diferentes domínios de aplicações a partir de uma única instalação. Por se tratar de uma solução voltada à iniciativa privada, a plataforma oferece suporte a diferentes ambientes de implantação, tais como servidores tradicionais (com sistemas operacionais Linux ou Windows, por exemplo), nuvens públicas (e.g., Amazon EC2, Microsoft Azure e Google Compute Engine), ou ainda nuvens privadas ou híbridas (tais como OpenStack, Suse Cloud, Eucalyptus, Amazon Virtual Private Cloud e Apache Stratos). A WSO2 é baseada na plataforma WSO2 Carbon que, por sua

vez, é uma implementação do OSGi (*Open Service Gateway Initiative*)[34], um conjunto de especificações que define um sistema dinâmico de componentes para Java.

Os módulos da plataforma mapeiam as camadas definidas pela arquitetura de referência WSO2 (c.f. Seção 3.3.3). A **Camada de Agregação/Barramento** é mapeada em dois módulos: o WSO2 Message Broker e o WSO2 Enterprise Service Bus. O primeiro módulo tem como papel básico lidar com a troca de mensagens dos protocolos do tipo *publish-subscribe* (AMQP e MQTT). Já o segundo módulo complementa o suporte aos protocolos *publish/subscribe*, lida com o protocolo HTTP, e lida com controle de acesso (OAuth2 e XACML). A **Camada de Análise e Processamento de Eventos** é, por sua vez, mapeada em dois módulos: WSO2 Business Activity Monitor e WSO2 Complex Event Processor. O primeiro oferece as funcionalidades: (i) coleta dos dados das camadas inferiores; (ii) armazenamento desses dados através do sistema de banco de dados distribuído Apache Cassandra[35]; (iii) plataforma de *map-reduce* baseada no Apache Hadoop; (iv) análise em *batch* fornecida pelo Apache Hive[36], e; (v) painel para a visualização dos dados recebidos. Já o segundo módulo é capaz de realizar análises de alto desempenho em tempo real e operações de provisionamento de recursos em caso de aumento da carga de trabalho. Por fim, a **Camada de Identidade de Gerenciamento de Acesso** é mapeada no módulo WSO2 *Identity Server* e fornece as seguintes funcionalidades de gerenciamento de acesso:

- provedor de identidade OAuth2 para emissão, revogação e gerenciamento de *tokens*.
- suporte a SSO (*Single Sign-On*) incluindo suporte a SAML2 SSO e OpenID.
- suporte para outros protocolos de identidade incluindo OpenID 2.0 e Kerberos[37].
- suporte completo para XACML (incluindo versões 2.0 e 3.0).
- capacidade de integração entre diferentes fornecedores de identidade e fornecedores de serviços incluindo intermediação de identidade.
- suporte para fornecimento de identidade incluindo SPML (*Service Provisioning Markup Language*)[38] e SCIM (*System for Cross-domain Identity Management*)[39].

3.4.11 *IoT Middleware* @ INRIA ARLES [Teixeira *et al.* 2011]

Com o intuito de propor soluções relativas aos requisitos de escalabilidade, adaptabilidade e heterogeneidade, a plataforma de *middleware* proposta por Teixeira *et al*. (2011) utiliza uma abordagem orientada a serviços baseada em semântica para descrever objetos físicos ou virtuais conectados como serviços de *software*. A partir dessa base, é introduzido o suporte a modelos matemáticos de estimativa de dados e resolução de conflitos de modo a se obter uma arquitetura com dispositivos e serviços

[34] OSGi Alliance: http://www.osgi.org/
[35] The Apache Cassandra Project: http://cassandra.apache.org/
[36] Apache Hive TM: https://hive.apache.org/
[37] MIT Kerberos Consortium: http://www.kerberos.org
[38] Service Provisioning Markup Language (SPML): https://www.oasis-open.org/committees/provision/
[39] System for Cross-Domain Identity Management (SCIM): http://www.simplecloud.info/

fracamente acoplados. A arquitetura projetada consiste basicamente de: (i) um módulo de descoberta; (ii) um módulo de estimativas, e; (iii) uma base de conhecimento.

O processo realizado pelo **módulo de descoberta** de serviços consiste de duas partes, uma de registro, que é o processo no qual cada dispositivo/objeto se conecta a um servidor para armazenar suas informações, e outra de descoberta, que é o processo de descoberta de serviços que satisfaçam o conjunto de atributos desejados (localização geográfica, modalidades de detecção, etc.). A fim de endereçar escalabilidade, é introduzido o conceito de descoberta probabilística, através do qual provê-se um conjunto de serviços que melhor se aproximam do resultado procurado. Em uma rede extremamente grande com uma base de conhecimento também grande, a busca de uma composição de serviços ótima se torna impraticável. Dessa forma, a proposta adota a abordagem de uma composição aproximadamente ótima fazendo uso de dois conceitos, a saber, expansão inteligente e mapeamento Probabilístico. No primeiro conceito, a fim de evitar cálculos exaustivos de todos os possíveis conjuntos de fluxos de dados equivalentes (dos quais, eventualmente, somente um será selecionado), utiliza-se uma abordagem mais inteligente, onde se produz um conjunto de "bons" fluxos de dados candidatos. Já no contexto do mapeamento probabilístico, tendo como entrada o conjunto selecionado na fase anterior, o mapeamento irá aleatoriamente escolher um pequeno subconjunto de todos os mapeamentos implementáveis através de consultas atômicas ao módulo de descobertas probabilísticas.

O **módulo de estimativas** é responsável por processar as requisições de sensoriamento e atuação, gerando um grafo de fluxo de dados que conecta os serviços disponíveis de modo a produzir a saída desejada com base nos parâmetros de entrada e a saída esperada. A fim de se enfrentar os desafios de localização desconhecida, os autores utilizam a estimativa automatizada que, por sua vez, utiliza-se da base de conhecimento da plataforma de *middleware* sobre sensoriamento, atuação, dispositivos, etc. Tal abordagem só é possível porque modelos físicos e estatísticos são previamente fornecidos e estão disponíveis na base de conhecimento. Dessa forma, quando uma informação ausente é detectada ou uma precisão maior é requisitada, a plataforma aplica os modelos mencionados a fim de estimar o valor com maior probabilidade de ser verdadeiro. Adicionalmente, é provável que o mesmo arcabouço usado na estimativa automatizada possa ser estendido para oferecer suporte a técnicas de resolução de conflito e de auto-calibração, uma vez que estas dependem de modelos armazenados na base de conhecimento, enfrentando-se, assim, os desafios de metadados incompletos ou inexatos e de resolução de conflitos. Por fim, a **base de conhecimento** armazena informações sobre sensoriamento, atuação, dispositivos, modelos de dados, etc. descritas na forma de ontologias e que são úteis para o módulo de estimativas. Tais ontologias contêm informações sobre como conceitos físicos diferentes se relacionam (ontologia de domínio), informações relacionadas a dispositivos físicos que podem existir na rede (ontologia de dispositivos), e informações acerca dos diferentes modelos de estimação (ontologia de estimativas).

Pelo exposto, essa plataforma de *middleware* adota uma arquitetura orientada a serviços a fim de abstrair sensores e atuadores como serviços a fim de ocultar suas heterogeneidades, e depende fortemente de uma base de conhecimento que contenha informações sobre sensores, atuadores, fabricantes, conceitos físicos, unidades físicas, modelos de dados, modelos de erros, entre outros. Para tratar os desafios decorrentes da grande escala e da profunda heterogeneidade inerente à IoT, a proposta concentra sua

contribuição em três núcleos: descoberta probabilística, composição aproximadamente ótima e estimação automatizada. Juntas, essas três características permitem à plataforma responder as requisições recebidas enquanto gerencia complexas relações entre precisão e tempo, memória, processamento e restrições energéticas dos dispositivos.

3.4.12. Plataformas de *middleware versus* requisitos

Esta subseção analisa as plataformas de *middleware* discutidas anteriormente em relação à satisfação dos requisitos vistos na Seção 3.2, a saber: (i) interoperabilidade; (ii) descoberta e gerenciamento de dispositivos; (iii) provisão de interfaces de alto nível; (iv) ciência de contexto; (v) escalabilidade; (vi) gerenciamento de grandes volumes de dados; (vii) segurança, e; (viii) adaptação dinâmica. Na Tabela 3.1 e na Tabela 3.2, o símbolo ✓ denota que o requisito é completamente atendido, o símbolo ○ indica que o requisito é parcialmente atendido, e o símbolo ✗ indica que o requisito não é atendido.

Tabela 3.1. Tabela de requisitos (parte 1).

Plataformas de *middleware*	Interoperabilidade	Descoberta e Gerenciamento de Dispositivos	Interfaces de Alto Nível	Ciência de Contexto
EcoDiF	✓	○	✓	○
Xively	✓	✓	✓	○
Carriots	✓	✓	✓	○
LinkSmart	✓	✓	✗	✓
OpenIoT	✓	✗	✓	✗
RestThing	✓	✗	✓	○
WoT Enabler	✓	✗	✓	○
S^3OIA	✓	✓	✗	✓
Ubiware	✓	✗	✗	✓
WSO2	✓	✓	✓	✗
INRIA ARLES	✓	✗	✗	✓

Tabela 3.2. Tabela de requisitos (parte 2).

Plataformas de *middleware*	Escalabilidade	Gerenciamento de Grandes Volumes de Dados	Segurança	Adaptação Dinâmica
EcoDiF	✓	✓	✓	✗
Xively	✓	✓	✓	✗
Carriots	✓	✓	✓	✗
LinkSmart	✗	✓	✓	✗
OpenIoT	✓	✓	✓	✗
RestThing	✗	✗	✗	✗
WoT Enabler	✗	✗	✗	✗
S^3OIA	✗	✗	✗	✓
Ubiware	✗	✗	✗	✓
WSO2	✓	✓	✓	✓
INRIA ARLES	✗	✓	✗	✗

Diante do exposto nas Tabelas 3.1 e 3.2, 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, abordando-os de formas diversas. A interoperabilidade é um exemplo. Apesar de ser atendida por todos, EcoDiF e OpenIoT consideram que o uso de protocolos e tecnologias Web amplamente utilizados são suficientes para mitigar os problemas da heterogeneidade entre os dispositivos, enquanto plataformas como Carriots, Xively e WSO2 acreditam que o suporte a outros protocolos é importante. Já requisitos como ciência de contexto e adaptação dinâmica são pouco abordados. A ciência de contexto é abordada pela maioria das plataformas levantadas ao caracterizar os dados coletados através da inclusão de dados semânticos, como localização, horário de coleta e afins, em conjunto com os dados coletados. Por sua vez, a adaptação dinâmica possui abordagens distintas. Enquanto a WSO2 a atende ao utilizar um módulo capaz de redimensionar a utilização dos recursos da plataforma a fim de manter a disponibilidade dos serviços, a plataforma Ubiware se utiliza de compartilhamento de dados entre agentes de *software* para melhorar o funcionamento dos objetos, tanto físico quanto virtuais, envolvidos.

Por conta disso, é possível concluir que o estado da arte para plataformas de *middleware* para IoT ainda se encontra em um estágio inicial, onde requisitos importantes não foram exaustivamente explorados. Há várias questões de pesquisa e desenvolvimento em aberto, as tecnologias e abordagens ainda divergem, além de não haver uma solução capaz de englobar todos os requisitos necessários para a concretização do paradigma.

3.5. Estudos de Caso

Esta seção apresenta um conjunto de aplicações desenvolvidas como provas de conceito para a EcoDiF, plataforma desenvolvida pelos grupos de pesquisa dos quais participam os autores deste minicurso e validada em diferentes cenários reais [Pires *et al*. 2014]. Tais provas de conceito foram propostas com o objetivo de demonstrar, de maneira prática, o potencial da EcoDiF para a integração de dispositivos heterogêneos e possibilitar o controle, visualização e processamento de dados por eles providos em tempo real. Conforme apresentado nas subseções a seguir, os diferentes domínios de aplicação nos quais essas aplicações estão inseridas são: (i) monitoramento de Centros de Processamento de Dados (CPDs); (ii) sensoriamento participativo; (iii) monitoramento de redes de dutos para transporte de água e gás, e; (iv) monitoramento remoto de pacientes. Em particular, a aplicação para monitoramento de CPDs é usada nesta seção para ilustrar, de forma completa, as atividades necessárias para a criação de aplicações utilizando a EcoDiF, desde a criação de *drivers* de integração entre dispositivos e a plataforma, à execução da aplicação propriamente dita fazendo uso de *feeds* nela cadastrados.

3.5.1. Monitoramento de Centros de Processamento de Dados (CPDs)

Em CPDs, é necessário monitorar constantemente equipamentos (ativos) tais como servidores, dispositivos de rede, dispositivos de armazenamento, etc., e notificar gerentes em caso de possíveis anormalidades em sua operação. De maneira adicional aos equipamentos e à infraestrutura lógica do CPD, é necessário monitorar variáveis

relacionadas à sua infraestrutura física, como temperatura, umidade relativa do ar, consumo de energia elétrica, dentre outras.

Em CPDs, as soluções usadas para desempenhar tarefas de monitoramento não são sistemáticas, algumas vezes não automatizadas, e também não são integradas. Consequentemente, gerentes e supervisores de CPDs necessitam, em geral, utilizar *software* de gerenciamento de propósito geral ou *software* proprietário de cada dispositivo, acessando-o explicitamente. Além disso, pela falta de integração entre as soluções de monitoramento existentes, eles não conseguem ter uma visão sistêmica da situação real do CPD e sua estrutura física, nem dos equipamentos que o compõem. Nesse contexto, a EcoDiF possibilita a integração desses diversos dispositivos e os dados por eles providos podem ser disponibilizados via Web para os usuários, sendo possível que os gerentes e supervisores do CPD tenham controle sobre o que está acontecendo no ambiente e possam inclusive ser notificados (por *e-mail* ou por mensagens recebidas em seus dispositivos móveis, por exemplo) acerca dos possíveis eventos que ocorram, conforme ilustra a Figura 3.15.

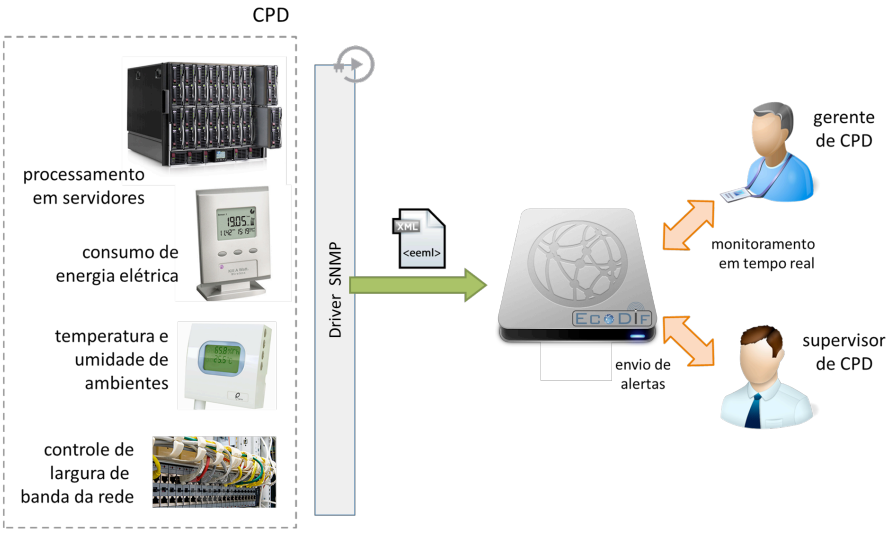

Figura 3.15. Ilustração de um cenário de monitoramento de CPDs com vistas à integração de dispositivos e disponibilização de dados através da EcoDiF.

A fim de disponibilizar dados coletados por dispositivos e permitir o seu posterior uso por aplicações, usuários da EcoDiF necessitam realizar um conjunto de atividades que envolvem principalmente: (i) o registro dos *feeds* que irão receber os dados enviados pelos *drivers*, no formato EEML; (ii) o desenvolvimento dos *drivers* de integração, que permitem os dispositivos comunicarem-se com a plataforma, enviando dados, e; (iii) o desenvolvimento das aplicações propriamente ditas, como *mashups* especificados utilizando a linguagem EMML. Uma vez criadas, as aplicações podem ser executadas sobre a EcoDiF fazendo uso dos *feeds* nela cadastrados. Essas atividades são brevemente descritas a seguir.

1) Criação de feeds na EcoDiF

Antes que dispositivos (através de seus respectivos *drivers* de integração) possam enviar dados à EcoDiF, é necessário criar os *feeds* que irão receber tais dados. A criação desses *feeds* é feita através da interface Web da plataforma, que disponibiliza um formulário a

ser preenchido pelo usuário. A Figura 3.16 mostra uma captura de tela referente ao formulário de cadastro de *feeds* na plataforma. No formulário, o usuário informa o dispositivo que enviará os dados e provê informações adicionais referentes ao *feed* que está sendo cadastrado, tais como descrição, unidade de medida utilizada e localização. Uma vez criado, o *feed* recebe um endereço (URI) único para o qual dados obtidos pelo dispositivo em questão deverão ser enviados via requisições HTTP PUT direcionadas a tal endereço.

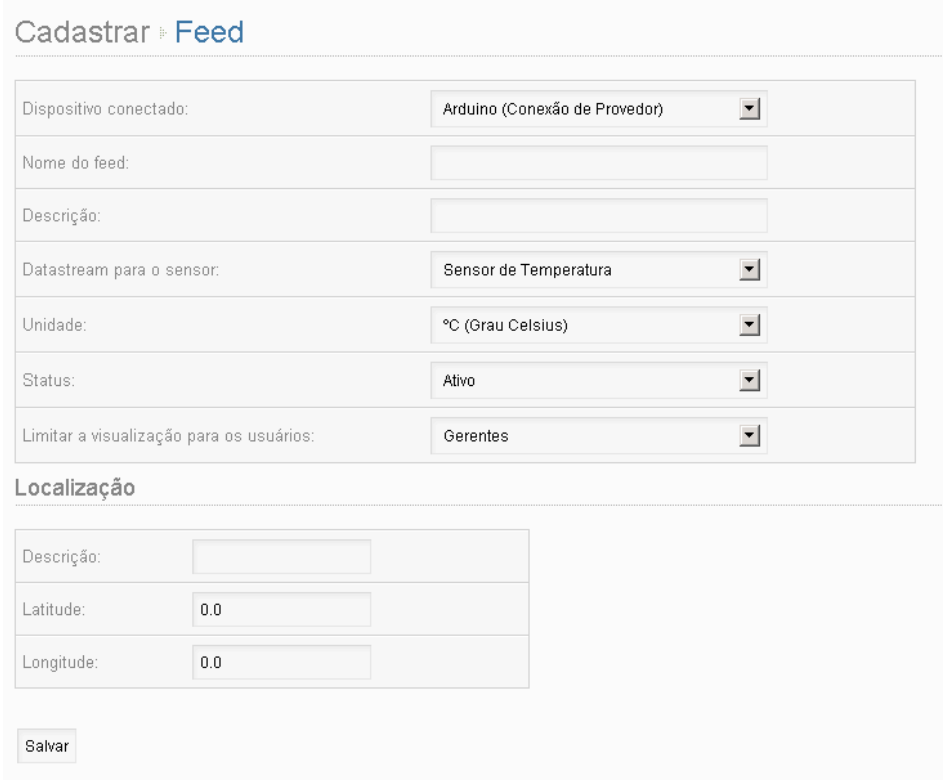

Figura 3.16. Interface Web para registro de *feeds* na EcoDiF.

2) Desenvolvimento de drivers de integração

Os equipamentos usados em CPDs, tipicamente, podem ser gerenciados usando o protocolo SNMP (*Simple Network Management Protocol*) [Case *et al*. 1990], protocolo da camada de aplicação para gerência de dispositivos em uma rede. O SNMP permite coletar diversas informações acerca de tais dispositivos usando os respectivos OIDs (*object identifiers*)[40] definidos na MIB (*Management Information Base*)[41] do dispositivo. Dessa forma, para obter uma dada informação acerca de um dispositivo, são necessários (i) o respectivo OID referente a essa informação, e (ii) o endereço do dispositivo em questão. A obtenção de tal informação dá-se por meio do protocolo SNMP, na camada de aplicação, sobre o protocolo UDP, na camada de transporte. Portanto, para permitir a integração desses dispositivos à EcoDiF e a disponibilização dos dados por eles providos na plataforma, com vistas à aplicação de monitoramento de CPDs, é necessário desenvolver um *driver* para o protocolo SNMP que, basicamente: (i) realize a captura das informações providas pelos dispositivos utilizando esse protocolo; (ii)

[40] Object identifier (OID): http://en.wikipedia.org/wiki/Object_identifier
[41] Management information base (MIB): http://en.wikipedia.org/wiki/Management_information_base

realize a estruturação desses dados capturados no formato EEML, adotado pela EcoDiF, e; (iii) envie as informações para a EcoDiF por meio de uma requisição HTTP PUT.

Para a aplicação em questão, foi desenvolvido um driver ativo para o protocolo SNMP como uma aplicação implementada utilizando a linguagem de programação Java e fazendo uso da SNMP4J[42], uma implementação de código aberto para o SNMP destinada a essa linguagem de programação. Para que o driver faça a coleta de dados referentes a um determinado equipamento compatível com o SNMP e os envie para a EcoDiF, quatro argumentos devem ser fornecidos como entrada: (i) o endereço Web do feed na EcoDiF no qual as informações deverão ser registradas; (ii) o endereço do equipamento a ser monitorado; (iii) o respectivo OID referente à informação a ser coletada a partir do equipamento, e; (iv) o intervalo de envio periódico dos dados, em segundos. Por exemplo, para obter a temperatura de um dispositivo no endereço `192.168.0.1`, o `driver` poderia ser executado utilizando o seguinte comando:

```
java -jar driver.jar
http://www.ecodif.com.br/EcodifAPI/api/feeds/1/datastreams/1
.1.3.6.1.4.1.25506.2.6.1.1.1.1.12.1
udp:192.168.0.1/161
3
```

o `http://www.ecodif.com.br/EcodifAPI/api/feeds/1/datastreams/1` é o endereço do *feed* no qual os dados referentes à temperatura serão armazenados na EcoDiF e `.1.3.6.1.4.1.25506.2.6.1.1.1.1.12.1` é o OID referente à temperatura do dispositivo em questão, conforme definido na sua respectiva MIB. Além disso, conforme especificado no último argumento, os dados de temperatura do equipamento serão coletados periodicamente a cada três segundos.

A Figura 3.17 mostra um trecho de código do *driver* SNMP referente ao método `startSending`, que instancia o serviço SNMP definido na SNMP4J para acessar o equipamento cujo endereço foi informado pelo usuário (variável *serverSNMP),* conforme descrito nas linhas 9 e 10. A informação de interesse é retornada pelo SNMP chamando-se o método `getAsString`, também definido na SNMP4J, que recebe como parâmetro o OID (variável `givenOID`) informado pelo usuário referente à informação em questão (linha 12). Uma vez obtida a informação de interesse, o método `sendValue` (linha 15) é chamado para enviar tal informação via requisição HTTP PUT ao endereço do *feed* especificado inicialmente pelo usuário como argumento (`addressPut`).

[42] SNMP4J – Free Open Source SNMP API for Java: http://www.snmp4j.org/

```
1   public static void startSending() {
2      int delay = 0;
3      int interval = Integer.parseInt(seconds) * 1000;
4
5      Timer timer = new Timer();
6
7      timer.scheduleAtFixedRate(new TimerTask() {
8         public void run() {
9            SnmpService client = new SnmpService(serverSNMP);
10           client.start();
11
12           String data = client.getAsString(new OID(givenOID));
13
14           DriverService driverService = new DriverService();
15           driverService.sendValue(data, addressPut);
16        }
17     }, delay, interval);
18  }
```

Figura 3.17. Trecho de código do *driver* referente ao método `startSending`, que instancia o serviço SNMP para coleta de dados do equipamento.

Após coletar o respectivo dado de interesse através do método `startSending` (Figura 3.17), o *driver* estrutura tal informação no formato EEML, adotado pela EcoDiF, e a envia para a EcoDiF executando o método `sendValue`, conforme mostrado na Figura 3.18. O método `sendValue` recebe como parâmetros o dado coletado pelo dispositivo usando o SNMP e o endereço para o qual ele deve ser enviado (i.e., o endereço do respectivo *feed* na EcoDiF). O dado recebido como parâmetro (armazenado na variável `value`) é estruturado no formato EEML, entre as linhas 7 e 19, fazendo uso da classe `Eeml_Contract`, implementada na EcoDiF e que define como dados devem ser estruturados nesse protocolo. Feito isso, os dados estruturados são efetivamente enviados para a EcoDiF via uma requisição HTTP PUT, através da chamada ao método `httpPutContract` (linha 21), que recebe como parâmetros o endereço do *feed*, informado pelo usuário como argumento de entrada, e o objeto representando o dado em EEML.

```
1   public void sendValue(String value, String address) {
2      Eeml_Contract contract = new Eeml_Contract();
3      Environment environment = new Environment();
4      Data data = new Data();
5      CurrentValue currentValue = new CurrentValue();
6
7      currentValue.setValue(value);
8      data.setCurrentValue(currentValue);
9      environment.getData().add(data);
10     contract.getEnvironment().add(environment);
11
12     JAXBContext jaxbContext = JAXBContext
13        .newInstance(Eeml_Contract.class);
14     Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
15     jaxbMarshaller
16        .setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
17
18     StringWriter strWriter = new StringWriter();
19     jaxbMarshaller.marshal(contract, strWriter);
20
21     String ret = httpPutContract(address, strWriter.toString());
22  }
```

Figura 3.18. Trecho de código do *driver* referente ao método `sendValue`, que estrutura o dado coletado no formato EEML e o envia para a EcoDiF.

A Figura 3.19 mostra um trecho de representação EEML referente ao *feed* associado à temperatura de um *switch* H3C® em um CPD da Universidade Federal do Rio Grande do Norte (UFRN), em Natal, representação essa gerada automaticamente pelo *driver* SNMP após coletar a respectiva informação acerca da temperatura atual do equipamento. Na linguagem EEML, *feeds* são descritos através do elemento `environment` (linha 2), que representa os dados do ambiente que está sendo monitorado. Esse elemento estrutura informações que descrevem o *feed* (e.g., nome, descrição, localização, estado atual, conforme descrevem as linhas 3 a 12), bem como as informações aferidas pelo sensor de temperatura do *switch*. As medições individuais são organizadas na forma de *datapoints* (linhas 15 a 21), informando o valor aferido em um determinado instante de tempo. A representação EEML exibida na Figura 3.19 informa ainda que o valor de temperatura do *switch* aferido mais recentemente (elemento *`current_value`*, linha 14) foi 40º C, inferior a valores coletados em medições anteriores (elemento `value`, linhas 16 a 20).

```
1   <eeml xmlns="http://www.eeml.org/xsd/0.5.1" version="0.5.1">
2      <environment creator="sinfoufrn">
3         <title>Temperatura_SINFO-5500G</title>
4         <status>Ativo</status>
5         <private>N</private>
6         <description>Temperatura de switch H3C S5500 na SINFO-UFRN</description>
7         <website>/EcodifAPI/api/feeds/1/datastreams/1</website>
8         <location domain="physical">
9            <name>SINFO-UFRN</name>
10           <lat>-5.84102</lat>
11           <lon>-35.198232</lon>
12        </location>
13        <data id="1">
14           <current_value at="2015-02-01T20:15:35.000-03:00">40</current_value>
15           <datapoints>
16              <value at="2015-02-01T07:42:33.000-03:00">41</value>
17              <value at="2015-02-01T07:42:33.000-03:00">41</value>
18              <value at="2015-02-01T07:42:33.000-03:00">41</value>
19              <value at="2015-02-01T07:42:33.000-03:00">41</value>
20              <value at="2015-02-01T07:42:34.000-03:00">41</value>
21           </datapoints>
22           <unit symbol="ºC" type="Grau Celsius" />
23        </data>
24     </environment>
25  </eeml>
```

Figura 3.19. Trecho de representação EEML referente à temperatura de um *switch* H3C® em um CPD da UFRN.

3) Desenvolvimento de aplicação mashup em EMML

Por fim, uma vez que a EcoDiF já está recebendo dados provenientes dos dispositivos integrados à plataforma através dos respectivos *drivers*, aplicações *mashup* escritas na linguagem EMML podem ser criadas para fazer uso de tais dados, desde a simples exibição deles de forma agregada através de uma página Web até a realização de tarefas mais complexas, que podem envolver, por exemplo, o processamento desses dados e consequente geração de novas informações de valor agregado para os usuários.

A título de exemplo, foi desenvolvida uma aplicação *mashup* que possui como objetivo monitorar diferentes variáveis referentes a um *switch* H3C® em um CPD da UFRN. Esses dados, disponibilizados na EcoDiF na forma de *feeds* e obtidos pelo *driver* SNMP anteriormente descrito, são recuperados pela aplicação por meio de requisições HTTP GET à plataforma e organizados em uma página Web, a ser exibida na interface Web da EcoDiF. A Figura 3.20 mostra um trecho de código na linguagem EMML que implementa a aplicação em questão. Através da diretiva `directinvoke`, a

aplicação realiza requisições HTTP GET aos endereços dos *feeds* que armazenam os dados referentes a temperatura e porcentagem de uso de processamento do *switch* monitorado (linhas 18-20 e 24-26, respectivamente), armazenando os resultados em variáveis declaradas no *mashup*, respectivamente `feed1` (declarada na linha 11) e `feed2` (declarada na linha 13). Nas linhas 30 a 46, o *mashup* constrói uma estrutura baseada em XML para organizar os dados obtidos através das requisições HTTP GET, utilizando as diretivas `<constructor>` e `<append>`. A fim de obter não apenas o valor dos *feeds* (no caso, temperatura e porcentagem de uso de processamento), mas também outras informações relevantes para serem exibidas, é possível navegar sobre o documento XML armazenado nas variáveis `feed1` e `feed2` utilizando expressões XPath (*XML Path Language*)[43]. Por exemplo, as expressões XPath especificadas nas linhas 35 a 37 navegam sobre o documento EEML representando o *feed* de temperatura (`feed1`) a fim de recuperar a descrição, (`description`), valor atual (`value`) e unidade (`unit`) desse *feed*. Por fim, essa nova estrutura XML resultante da agregação das informações dos *feeds* de temperatura e porcentagem de uso de processamento são tratadas por um *script* XSL (*Extensible Stylesheet Language*)[44], uma linguagem para processamento de documentos baseados em XML. Esse *script* recebe como entrada a estrutura XML que foi construída e retorna o código HTML a ser renderizado na interface Web da EcoDiF como resultado da execução do *mashup*, armazenado na variável de saída `result` (linha 50).

[43] XML Path Language (XPath) 3.0: http://www.w3.org/TR/xpath-30/
[44] The Extensible Stylesheet Language Family (XSL): http://www.w3.org/Style/XSL/

```
<mashup name="cpd-sinfo"
   xmlns="http://www.openemml.org/2009-04-15/EMMLSchema"
   xsi:schemaLocation="http://www.openemml.org/2009-04-15/EMMLSchema
      ../schema/EMMLSpec.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:macro="http://www.openemml.org/2009-04-15/EMMLMacro">

   <output name="result" type="document“ />

   <variables>
      <variable name="feed1"  type="document" />
      <variable name="_feed1" type="document" />
      <variable name="feed2"  type="document" />
      <variable name="_feed2" type="document" />
   </variables>

   <!-- Temperatura -->
   <directinvoke
endpoint="http://recife.voip.nce.ufrj.br/EcodifAPI/api/feeds/1/datastreams/1"
method="GET" outputvariable="$feed1" />
   <xslt script="eeml.xsl" inputvariable="feed1" outputvariable="_feed1" />

   <!-- Uso de CPU -->
   <directinvoke
endpoint="http://recife.voip.nce.ufrj.br/EcodifAPI/api/feeds/2/datastreams/2"
method="GET" outputvariable="$feed2" />
   <xslt script="eeml.xsl" inputvariable="feed2" outputvariable="_feed2" />

   <!-- Construção de estrutura baseada em XML para os dados -->
   <constructor outputvariable="merge">
      <data/>
   </constructor>
   <appendresult outputvariable="merge">
      <feed>
         <description>{$_feed1//description/string()}</description>
         <value>{$_feed1//current_value/string()}</value>
         <unit>{$_feed1//unit//@symbol/string()}</unit>
      </feed>
   </appendresult>
   <appendresult outputvariable="merge">
      <feed>
         <description>{$_feed2//description/string()}</description>
         <value>{$_feed2//current_value/string()}</value>
         <unit>{$_feed2//unit//@symbol/string()}</unit>
      </feed>
   </appendresult>

   <!-- Saida para HTML com XSL -->
   <xslt script="cpd-sinfo.xsl" inputvariable="merge" outputvariable="result" />
</mashup>
```

Figura 3.20. Trecho de código EMML referente à aplicação que agrega dados coletados de um *switch* H3C® em um CPD na UFRN.

A Figura 3.21 mostra uma captura de tela da interface Web da EcoDiF com o resultado da execução da aplicação. Na EcoDiF, aplicações *mashup* escritas em EMML são processadas pelo motor de execução da linguagem, implantado na plataforma, e o resultado de tal execução é renderizado em sua interface Web. Através dessa interface, os usuários podem acompanhar, em tempo real, o estado de diferentes variáveis referentes ao *switch*, tais como temperatura e porcentagem de uso de processamento, bem como medidas coletadas pelas interfaces de rede do equipamento, e.g., número de octetos de entrada e saída, e número de pacotes de saída em estado de falha.

Figura 3.21. Exibição da aplicação de gerenciamento de um switch H3C em um CPD da UFRN, através do portal Web da EcoDiF.

3.5.2. Sensoriamento Participativo

A proliferação de dispositivos móveis conectados à Internet permite a coleta de diversos tipos de informação acerca do ambiente, tempo, tráfego, pessoas etc. a um custo relativamente baixo [Goldman *et al*. 2009]. Nessa perspectiva, o conceito de *sensoriamento participativo* (*participatory sensing*, em Inglês) [Burke *et al*. 2006] surgiu como uma forma sistemática de recuperar tais informações enfatizando o envolvimento ativo dos usuários como provedores desses dados, que podem ir desde observações pessoais à combinação de dados providos por milhares de indivíduos acerca do local onde vivem [Campbell *et al*. 2008]. Através de redes de sensores e/ou redes móveis, usuários espalhados por um determinado lugar, por uma cidade ou mesmo pelo mundo, podem prontamente enviar dados a servidores, que os processam e eventualmente os integram com outros tipos de dados, tais como mapas de sistemas de informações geográficas e relatórios meteorológicos. Nos últimos anos, essa abordagem de sensoriamento participativo tem se mostrado bastante útil em diversos cenários, a exemplo de planejamento de transportes, saúde pública, e sustentabilidade ambiental.

Um exemplo de aplicação em sensoriamento participativo é a estimativa de presença de público em determinados ambientes com o objetivo de melhor prover recursos de acordo com a quantidade de pessoas neles presentes. Em uma conferência,

por exemplo, pode ser interessante saber quantas pessoas participaram das diversas sessões do evento (*workshops*, tutoriais, sessões técnicas, etc.) ou obter informações acerca de parâmetros do ambiente, tais como temperatura, luminosidade e ruído. Nesse cenário, foi desenvolvida uma aplicação que visava informar a quantidade de pessoas nas salas de um centro de convenções no qual estava ocorrendo uma conferência. Para fins de validação, participantes da conferência voluntariamente fizeram o *download* e instalação de um *driver* passivo, implementado como um aplicativo para o sistema operacional Android (versão 2.3 Gingerbread ou superior) e disponibilizado publicamente para *download* através da Google Play Store[45]. A principal função desse aplicativo era ler códigos QR (*QR codes*)[46] fixados nas salas do prédio no qual estava ocorrendo a conferência e mostrar para os usuários as atividades programadas para a respectiva sala e horário no qual eles fizeram a leitura do código QR. No contexto de sensoriamento participativo, dado que os usuários geralmente proveem informações de maneira voluntária, haver algum tipo de retorno às ações por eles realizadas (retorno esse que possua um valor agregado para tais usuários) é importante para que os motive a estarem envolvidos no provimento das informações necessárias aos sistemas que irão fazer uso delas. Dessa forma, a disponibilização da programação atualizada do evento de acordo com o local/horário do *check-in* foi uma forma encontrada para incentivar as pessoas que estavam presentes a participarem da demonstração.

Ao executar o aplicativo, os usuários apontavam a câmera dos seus dispositivos (*smartphones* e/ou *tablets*) para os códigos QR e, feito isso, os dados de localização eram estruturados no formato EEML e enviados à EcoDiF através de requisições HTTP PUT, a fim de registra-los na plataforma. De modo ideal, o sensor de GPS dos dispositivos poderia ser utilizado para prover a localização dos usuários, porém devido à imprecisão de leitores GPS em ambientes fechados, optou-se por prover tal informação através da leitura dos códigos QR contendo a informação de sua localização. A fim de utilizar as informações de localização providas pelos usuários, foi desenvolvida uma aplicação *mashup* que recupera tais dados na EcoDiF na forma de *feeds*. Como resultado, a aplicação exibe um mapa de presença (similar a uma planta do prédio) com o número de participantes presentes às sessões da conferência, conforme mostrado na Figura 3.22.

[45] Google Play Store: https://play.google.com/store
[46] QR code: http://en.wikipedia.org/wiki/QR_code

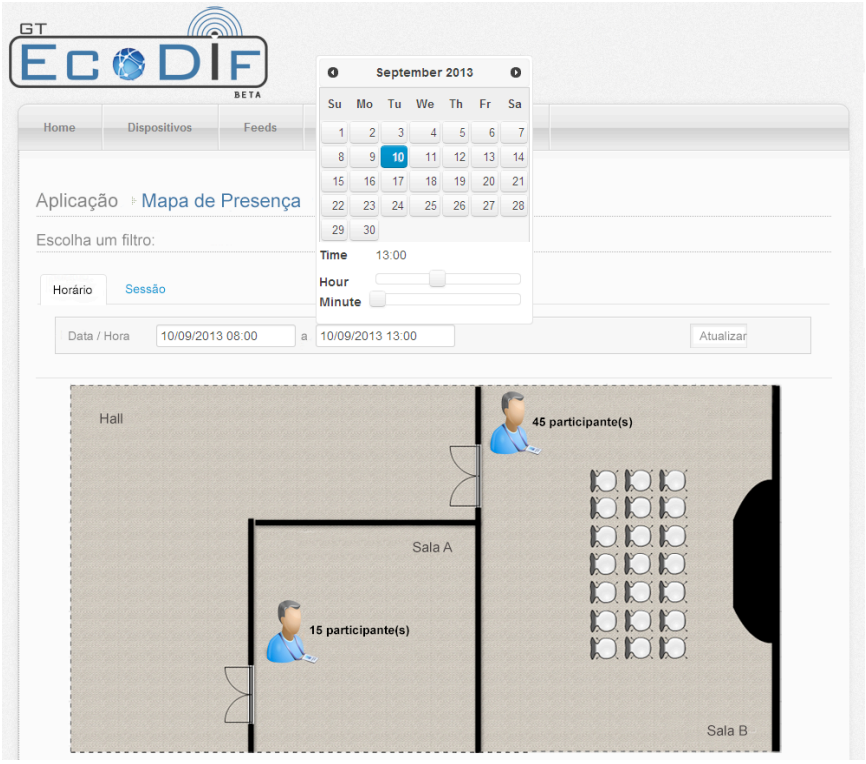

Figura 3.22. Mapa de presença referentes às salas de uma conferência exibido através da interface Web da EcoDiF.

3.5.3. Monitoramento de redes de dutos de água e gás

A aplicação desenvolvida neste contexto possui como objetivo monitorar o fluxo de água e gás em redes de dutos de companhias públicas da cidade do Rio de Janeiro. Para permitir a coleta de dados, sensores Dialog 3G™ Universal[47] foram implantados nos dutos a serem monitorados a fim de medir a vazão desses fluidos em tais dutos. Uma vez coletados, os dados mensurados são estruturados por um *driver* construído para esse tipo de sensor no formato EEML e, em seguida, enviados à EcoDiF em tempo real. Dessa forma, usuários (a exemplo de operadores e gerentes das companhias proprietárias dos dutos) podem visualizar as últimas medidas aferidas pelos sensores através da interface Web da EcoDiF, bem como podem ser notificados em caso de condições anormais de operação quando, por exemplo, os valores mensurados indicam um possível vazamento de água/gás no duto.

A Figura 3.23 mostra os últimos dados mensurados referente ao fluxo de gás em um duto na Cinelândia, no Centro da cidade do Rio de Janeiro, coletados pelos sensores Dialog 3G™ Universal e enviados à EcoDiF. No gráfico exibido, é possível observar que o fluxo de gás mantém-se em um valor constante (14,66 m^3/s), indicando que o duto está sob condições normais de operação.

[47] Dialog 3G™ Universal: http://aradtec.com/Product.aspx?ID=6

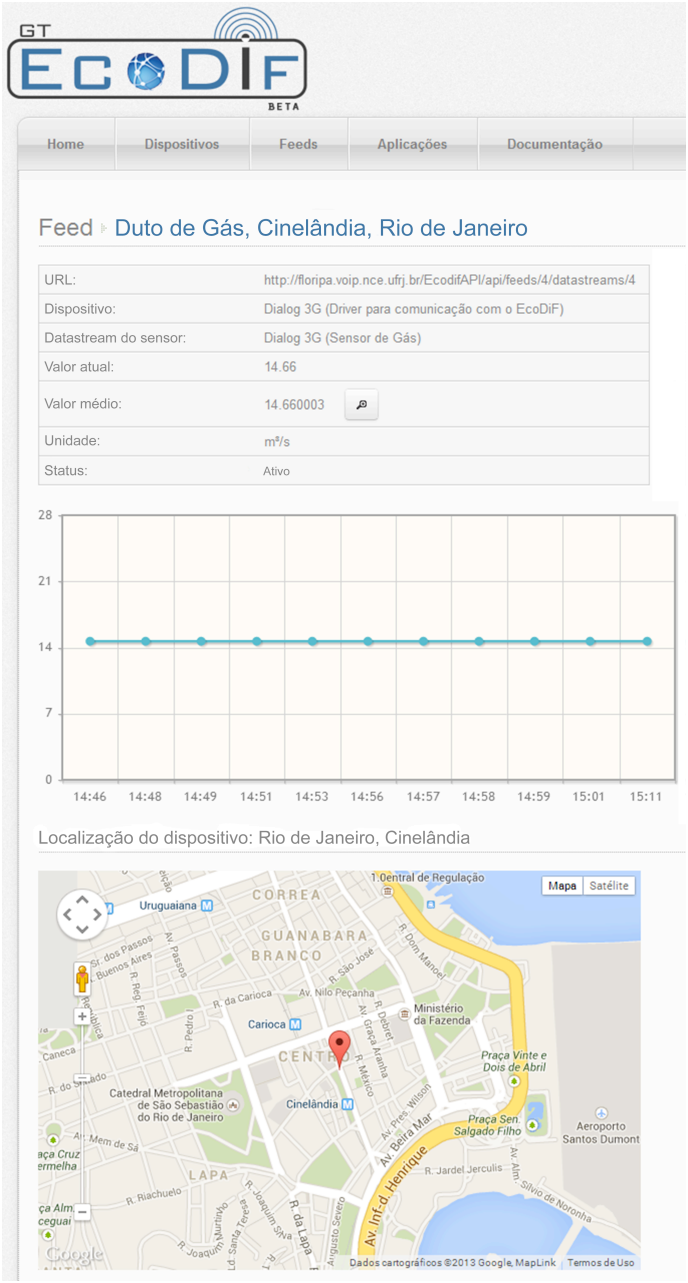

Figura 3.23. Monitoramento de um duto de gás no Centro da cidade do Rio de Janeiro através da interface Web da EcoDiF.

3.5.4. Monitoramento remoto de pacientes

No contexto de *e-health*, sensores corporais são pequenos dispositivos implantados no corpo humano ou sob a roupa e que são geralmente utilizados para coletar diversos dados relativos aos sinais vitais de quem os utiliza [Yuce 2013]. Esses sensores possuem capacidades de comunicação sem fio, aumentando assim o conforto e a mobilidade do usuário e não impedindo suas atividades normais enquanto o monitoramento é realizado [Sebestyen *et al*. 2014]. Nessa perspectiva, tais dispositivos são capazes de contribuir para o aumento da qualidade de serviços médicos pelo fato de permitir, a um custo relativamente baixo, a detecção de situações de emergência de

maneira mais rápida, bem como um diagnóstico mais preciso a ser elaborado por profissionais de saúde.

Os dados provenientes de sensores corporais podem fornecer importantes informações acerca das condições de saúde atuais de um paciente. A título de exemplo, o monitoramento da temperatura corporal pode ser útil para detectar infecções e doenças, de modo que tal monitoramento constante permite melhor administrar o medicamento adequado. Por outro lado, outros sensores poderiam ser utilizados para confirmar alguns sintomas detectados com base na aferição da temperatura ou frequência respiratória. Através do uso de um sensor de eletrocardiograma (ECG) e da verificação da frequência de batimentos cardíacos, é possível identificar uma variedade de problemas de saúde que podem afetar um paciente, confirmando a sua condição clínica. Entretanto, a inerente heterogeneidade dos sensores que podem ser empregados dificulta a utilização sistêmica desses dados que, uma vez, combinados, permitiriam um diagnóstico mais preciso por parte dos médicos. Dessa forma, uma plataforma de middleware para IoT como a EcoDiF desempenha um papel fundamental nesse cenário, por permitir a integração desses diferentes sensores corporais acoplados a um paciente e agregação das informações por eles providas, ao mesmo tempo que possibilita monitorar, em tempo real, seus sinais vitais, e disparar alertas em caso de condições anormais.

Nessa prova de conceito, três sensores corporais foram acoplados a um voluntário para monitoramento de seus sinais vitais e disponibilização destes na plataforma, conforme ilustrado na Figura 3.24: (i) *um sensor de respiração*, através de uma máscara fixada no nariz do indivíduo, para medir ciclos de inalação/exalação em intervalos de um minuto; (ii) *um sensor de temperatura*, fixado no braço do indivíduo, e; (iii) *um sensor de ECG* com eletrodos fixados no tórax do indivíduo para a medição de batimentos cardíacos por minuto (BPMs). Dessa forma, os dados coletados pelos sensores são enviados à EcoDiF, que permite acompanhar as condições atuais do paciente com base em tais dados.

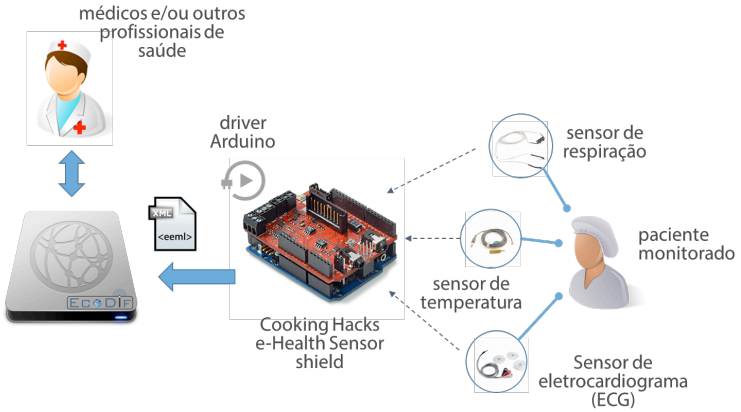

Figura 3.24. Ilustração do monitoramento remoto de respiração, temperatura e batimentos cardíacos de um paciente, com envio e disponibilização de dados para a EcoDiF.

Para possibilitar a comunicação desses dispositivos com a EcoDiF, os sensores de temperatura, respiração e ECG foram conectados a um *shield* Cooking Hacks e-

Health Sensor[48] acoplado a uma placa de prototipação Arduino Uno[49], combinação essa que permite a integração de diversos tipos de sensores biométricos. Os batimentos cardíacos, a temperatura corporal e a frequência respiratória do indivíduo monitorado foram registrados na EcoDiF como *feeds*, de modo que um *driver* ativo desenvolvido para o Arduino Uno (i) coleta os dados aferidos pelos sensores, (ii) estrutura-os no formato EEML, e (iii) envia-os via requisições HTTP PUT para que sejam armazenados na plataforma. Com isso, através da interface Web da EcoDiF, é possível visualizar o histórico de eventos e as variações ocorridas nas medidas aferidas para cada variável. No caso da medição de batimentos cardíacos, os dados históricos disponibilizados pela EcoDiF também podem fornecer informações acerca de períodos nos quais o paciente manteve-se em alta e baixa atividade durante o dia, sendo possível, por exemplo, determinar seu consumo calórico. A Figura 3.25 apresenta a leitura dos valores referentes aos batimentos cardíacos do indivíduo monitorado em um intervalo de três minutos.

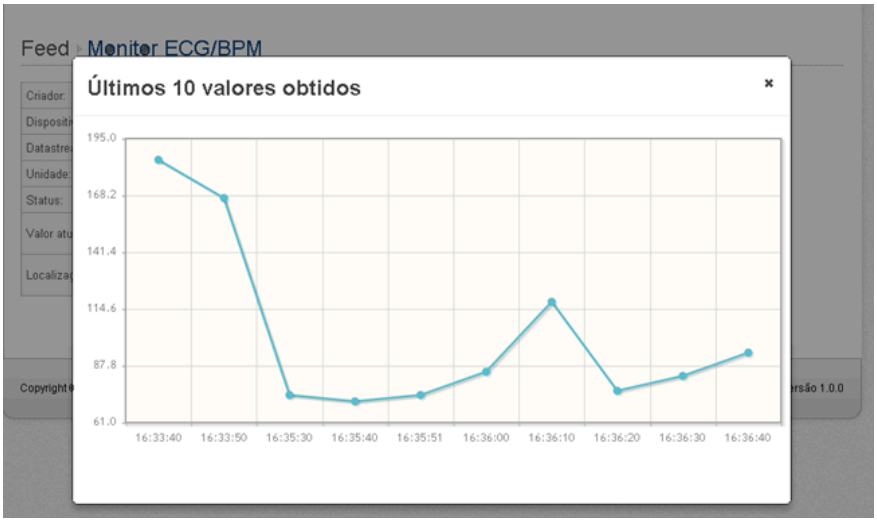

Figura 3.25. Histórico de monitoramento dos batimentos cardíacos de um paciente visualizado através da interface Web da EcoDiF.

3.6. Conclusão

A ideia básica por trás do conceito de IoT é a presença generalizada de uma variedade de objetos (tais como etiquetas RFID, sensores, atuadores, telefones celulares, etc.) que, através de esquemas de endereçamento único e outros mecanismos de suporte baseados em padrões e protocolos ubíquos, são capazes de interagir uns com os outros e cooperar com seus vizinhos para alcançar objetivos comuns. Dessa forma, esse paradigma tem o potencial de contribuir em vários aspectos da vida contemporânea, propiciando uma vasta gama de aplicações que facilitem tarefas cotidianas. Domínios de aplicação como domótica, ambientes de vida assistida, monitoramento ambiental, automação industrial, transporte inteligente e gestão de negócios são apenas alguns dos possíveis cenários de aplicação em que esse novo paradigma irá desempenhar papeis primordiais em um futuro próximo.

---

[48] e-Health Sensor Platform v2.0 for Arduino and Raspberry Pi: http://goo.gl/tdRDzr.

[49] Arduino Uno: http://arduino.cc/en/Main/arduinoBoardUno.

Diversos avanços tecnológicos nas últimas décadas permitiram a emergência do paradigma de IoT. Contudo, muitas questões desafiadoras ainda precisam ser abordadas e vários obstáculos tecnológicos ainda precisam ser suplantadas para a concretização e a ampla utilização desse paradigma. Tais desafios, anteriormente mencionados, vão desde soluções que permitam a interoperação e a integração dos diversos componentes que compõem os ambientes de IoT até a facilitação do desenvolvimento de aplicações para esses ambientes, passando por questões relativas à escalabilidade por conta do grande número de objetos (*coisas*) envolvidos e do grande número de eventos que podem ser gerados espontaneamente por esses mesmos objetos. Além disso, os objetos que irão compor a IoT são caracterizados por baixos recursos computacionais e de energia. Consequentemente, as soluções propostas precisam ter especial consideração por questões relativas à eficiência dos recursos. Nesse contexto, plataformas de *middleware* devem satisfazer a um conjunto de requisitos a fim de satisfazer as demandas dos desafios apresentados. Questões de interoperabilidade, descoberta e gerenciamento de dispositivos, interfaces de comunicação, ciência de contexto, escalabilidade, gerenciamento de dados, segurança e adaptação dinâmica emergem frequentemente na literatura.

Neste capítulo, discutiu-se como arquiteturas de referência e plataformas de *middleware* existentes lidam com esses desafios e foram apontadas suas deficiências. Tais deficiências indicam que o estado da arte para plataformas de *middleware* no contexto de IoT ainda não permite a plena concretização desse paradigma, fomentando a emergência de novas plataformas de *middleware* que abordem os requisitos levantados por completo. Por fim, a utilização de uma plataforma de *middleware*, a EcoDiF, foi demonstrada através de uma série de estudos de caso, comprovando os potenciais benefícios que tais plataformas podem oferecer.

Referências

Angelov, S., Grefen, P., Greefhorst, D. (2009) "A classification of software reference architectures: Analyzing their success and effectiveness". Proceedings of the 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. USA, IEEE, pp. 141-150.

Atzori, L., Iera, A., and Morabito, G. (2010) "The Internet of Things: A survey". Computer Networks vol. 54, no. 15, pp. 2787-2805.

Bassi, A., Bauer, M., Fiedler, M., Kramp, T., van Kranenburg, R., Lange, S., Meissner, S. eds. (2013) "Enabling things to talk: Designing IoT solutions with the IoT Architectural Reference Model". Germany, Springer Berlin Heidelberg.

Bandyopadhyay, S., Sengupta, M., Maiti, S., & Dutta, S. (2011) "Role of middleware for internet of things: A study". International Journal of Computer Science & Engineering Survey, vol. 2, no. 3, pp. 94-105.

Bernstein, P. A. (1996) "Middleware: a model for distributed system services". Communications of the ACM, vol. 39, no. 2, 86-98.

Burke, J. A., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., & Srivastava, M. B. (2006) "Participatory Sensing". Proceedings of the First Workshop on World-Sensor-Web: Mobile Device Centric Sensory Networks and Applications. USA, ACM.

Campbell, A. T. *et al*. (2008) "The rise of people-centric sensing", IEEE Internet Computing, vol. 12, no. 4, pp. 12-21.

Case, J., Fedor, M., Schoffstall, M., Davin, J. (1990) "A Simple Network Management Protocol (SNMP) – RFC 1157". USA, IETF.

Chaqfeh, M. A., Mohamed, N. (2012) "Challenges in middleware solutions for the Internet of Things", Proceedings of the 2012 International Conference on Collaboration Technologies and Systems. USA: IEEE, pp. 21-26.

Chen, M., Mao, S., Liu, Y. (2014) "Big Data: A survey", Mobile Networks and Applications, vol. 19, no. 2, pp. 171-209.

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E., Bone, M. (2010) "The concept of reference architectures". Systems Engineering, vol. 13, no. 1, pp. 14-27.

Delicato, F.C., Pires, P.F., Batista, T. (2013a) "Middleware solutions for the Internet of Things". United Kingdom, Springer London.

Delicato, F. C., Pires, P. F., Batista, T., Cavalcante, E., Costa, B., Barros, T. (2013b) "Towards an IoT ecosystem". Proceedings of the First International Workshop on Software Engineering for Systems of Systems. USA, ACM, pp. 25-28.

Dey, A. K. (2001) "Understanding and using context", Personal and Ubiquitous Computing, vol. 5, no. 1, pp. 4-7.

Fielding, R. (2000) "Architectural styles and the design of network-based software architectures". PhD dissertation, University of California at Irvine, USA.

Fremantle, P. (2014) "A reference architecture for the Internet of Things – version 0.8.2". Whitepaper, WSO2, USA.

Gao, L., Zhang, C., Sun, L. (2011) "RESTful Web of Things API in sharing sensor data", Proceedings of the 2011 International Conference on Internet Technology and Applications. USA, IEEE, pp. 1-4.

Gartner (2014) "Gartner says 4.9 billion connected 'things' will be used in 2015', http://www.gartner.com/newsroom/id/2905717.

Goldman, J. *et al*. (2009) "Participatory Sensing: A citizen-powered approach to illuminating the patterns that shape our world". Whitepaper, Woodrow Wilson International Center for Scholars, USA.

Guinard, D., Trifa, V. (2009) "Towards the Web of Things: Web mashups for embedded devices", Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web.

Katasonov, A., Terziyan, V. (2008) "Semantic agent programming language (S-APL): A middleware platform for the Semantic Web", Proceedings of the 2008 IEEE International Conference on Semantic Computing. USA, IEEE, pp. 504-511.

Le-Phuoc, D., Nguyen-Mau, H. Q., Parreira, J. X., Hauswirth, M. (2012) "A middleware framework for scalable management of linked streams", Web Semantics: Science, Services and Agents on the World Wide Web, vol. 16, pp. 42-51.

Muller, G. (2008) "A reference architecture primer". Whitepaper, Embedded Systems Institute, The Netherlands.

Nagy, M., Katasonov, A., Szydlowski, M., Khriyenko, O., Nikitin, S., Terziyan, V. (2009) “Challenges of middleware for the Internet of Things”. In: Rodic, A. D., ed. Automation & control – Theory and Practice. Croatia, InTech.

Nakagawa, E. Y., Antonino, P. O., Becker, M. (2011) “Reference architecture and product line architecture: A subtle but critical difference”. In: Crnkovic, I., Gruhn, V., Book, M., eds. Proceedings of the 5th European Conference on Software Architecture. Lecture Notes in Computer Science, vol. 6903. Germany, Springer Berlin Heidelberg, pp. 207-211.

Nakagawa, E. Y., Oquendo, F., Maldonado, J. C. (2014) “Reference architectures”. In: Oussalah, M. C., ed. Software Architecture 1. United Kingdom, ISTE Ltd / John Wiley & Sons, Inc., pp. 55-82.

Pautasso, C., Zimmermann, O., Leymann, F. (2008) “RESTful Web services vs. ‘big’ Web services: Making the right architectural decision”, Proceedings of the 17th International Conference on the World Wide Web. USA, ACM, pp. 805-814.

Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D. (2014) “Context aware computing for the Internet of Things: A survey”, IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 414-454.

Pires, P. F., Cavalcante, E., Barros, T., Delicato, F. C., Batista, T., Costa, B. (2014) “A platform for integrating physical devices in the Internet of Things”, Proceedings of the 12th IEEE International Conference on Embedded and Ubiquitous Computing. USA, IEEE, pp. 234-241.

Qin, W., Li, Q., Sun, L., Zhu, H., Liu, Y. (2011) “RestThing: A Restful Web service infrastructure for mash-up physical and Web resources”, Proceedings of the 9th IFIP International Conference on Embedded and Ubiquitous Computing. USA, IEEE, pp. 197-204.

Rozanski, N., Woods, E. (2011) “Software systems architecture: working with stakeholders using viewpoints and perspectives”. Addison-Wesley.

Sebestyen, G., Hangan, A., Oniga, S., Gal, Z. (2014) “eHealth solutions in the context of Internet of Things”, Proceedings of the 2014 International Conference on Automation, Quality and Testing, Robotics. Romania, IEEE, pp. 1-6.

Soldatos, J., Serrano, M., Hauswirth, M. (2012) “Convergence of Utility Computing with the Internet-of-Things”, Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. USA, IEEE, pp. 874-879.

Teixeira, T., Hachem, S., Issarny, V., Georgantas, N. (2011) “Service oriented middleware for the Internet of Things: A perspective”. In: Abramowicz, W., Llorente, I. M., Surridge, M., Zisman, A., Vayssière, J., eds. Proceedings of the 4th European Conference on Towards a Service-Based Internet. Lecture Notes in Computer Science, vol. 6994. Germany, Springer Berlin Heidelberg, pp. 220-229.

Tracey, D., Sreenam, C. (2013) “A holistic architecture for the Internet of Things, sensing services, and Big Data”, Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. USA, IEEE, pp. 546-553.

Vega-Barbas, M., Casado-Mansilla, D., Valero, M. A., López-de-Ipina, D., Bravo, J., Florez, F. (2012) "Smart spaces and smart objects interoperability architecture (S3OiA)", Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. USA, IEEE, pp. 725-730.

Yuce, M. R. (2013) "Recent wireless body sensors: Design and implementation". Proceedings of the 2013 IEEE MTT-S International Microwave Workshop Series on RF and Wireless Technologies for Biomedical and Healthcare Applications. USA, IEEE Computer Society, pp. 1-3.

Zanella, A., Bui, N., Castellani, A., Vangelista, L., Zorzi, M. (2014) "Internet of Things for smart cities", IEEE Internet of Things Journal, vol. 1, pp. 22-32.