Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

SmartMetropolis – Plataformas e Aplicações para Cidades Inteligentes

WP5 – Middleware

Requisitos e Plataformas de Middleware para Cidades Inteligentes

Natal-RN, Brasil
Maio de 2016

Equipe Técnica

Docentes

Prof.ª Dra. Thais Vasconcelos Batista (Coordenadora) - DIMAp-UFRN
Prof. Dr. Augusto José Venâncio Neto - DIMAp-UFRN
Prof. Dr. Carlos Eduardo da Silva - IMD-UFRN
Prof.ª Dra. Flavia Coimbra Delicato - DCC-UFRJ
Prof. Dr. Frederico Araújo da Silva Lopes - IMD-UFRN
Prof. Dr. Gibeon Soares de Aquino Junior - DIMAp-UFRN
Prof. Dr. Jair Cavalcanti Leite - DIMAp-UFRN
Prof. Dr. Nélio Alessandro Azevedo Cacho - DIMAp-UFRN
Prof. Dr. Paulo de Figueiredo Pires - DCC-UFRJ

Discentes

Arthur Emanoel Cassio da Silva e Souza
Claudio Silva Trindade
Everton Ranielly de Sousa Cavalcante
Helber Wagner da Silva
Juliana de Araújo Oliveira
Lucas Cristiano Calixto Dantas
Marcus Vinicius Alves Maia
Stefano Momo Loss

Sumário

1 Introdução

A massiva proliferação de dispositivos de diversas escalas e com capacidades computacionais, bem como a convergência das tecnologias de informação e comunicação (TICs), vem tornando realidade a ideia de cidades inteligentes. O conceito de *cidades inteligentes* refere-se ao conjunto de serviços avançados para tornar as cidades mais eficientes e mais sustentáveis, provendo novas maneiras de tratar problemas sociais e proporcionando uma melhor qualidade de vida para os cidadãos. Gerenciamento de tráfego, controle de poluição, suporte a coleta de lixo e segurança são exemplos de alguns problemas que uma cidade inteligente deve tratar [1]. Cidades inteligentes realizam a *Internet de Todas as Coisas (do inglês, Internet of Everything - IoE)*, conectando pessoas, processos, dados e coisas, conforme ilustra a Figura 1.

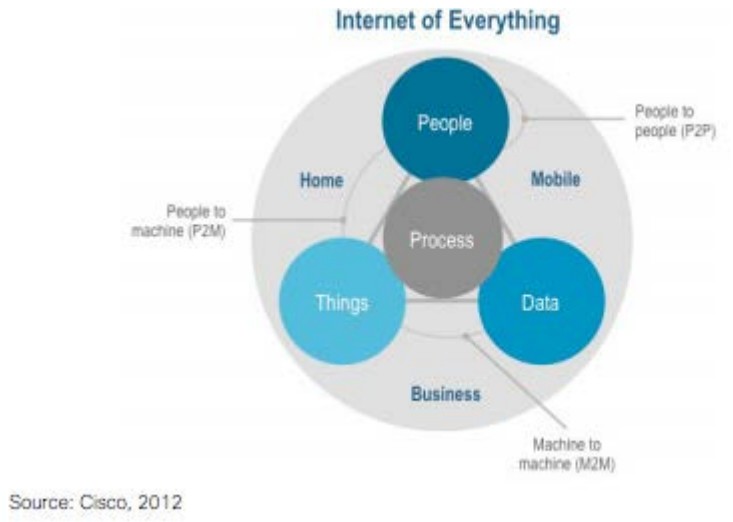

Figura 1. *Internet of Everything - IoE*

Esse contexto dinâmico e altamente heterogêneo, decorrente da inerente diversidade de tecnologias de *hardware* e *software*, faz com que o desenvolvimento de software para cidades inteligentes envolva uma série de desafios, tais como heterogeneidade de dispositivos, gerenciamento de grande volume de dados, incluindo coleta e análise de informações geradas por diversas fontes, escalabilidade, políticas de privacidade, entre outros. Dessa forma, plataformas de *middleware* têm surgido como soluções promissoras para facilitar o desenvolvimento de aplicações, provendo interoperabilidade para possibilitar a integração de dispositivos, pessoas, sistemas e dados, e uma série de serviços adicionais necessários para as aplicações [2]. 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 [3]. Ou seja, uma plataforma de *middleware* fornece abstrações para dispositivos e

aplicações, diversos níveis de transparência e interoperabilidade, bem como múltiplos serviços para usuários finais e aplicações, ocultando 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 [4].

Há várias propostas de plataformas de *middleware* para desenvolvimento de aplicações para cidades inteligentes, cada uma com suas especificidades, endereçando um conjunto específico de serviços. Contudo, observa-se que ainda não há nenhum padrão e, por ser uma área de pesquisa recente, a maioria das propostas existentes ainda não atingiram maturidade. Nessa perspectiva, este relatório tem como objetivo elucidar os requisitos de plataformas de *middleware* para cidades inteligentes, apresentar brevemente algumas plataformas que têm sido usadas para esse propósito, analisar se as plataformas apresentadas atendem os requisitos elucidados, e apresentar, em mais detalhes a plataforma que atender ao maior número de requisitos.

Este relatório está estruturado da seguinte forma. O Capítulo 2 descreve os requisitos de plataformas de *middleware* para cidades inteligentes. O Capítulo 3 apresenta cinco plataformas de *middleware* que têm sido usadas no suporte ao desenvolvimento de aplicações para cidades inteligentes, finalizando com uma análise comparativa acerca de tais plataformas à luz dos requisitos levantados. O Capítulo 4 descreve os elementos da plataforma FIWARE, identificada como a que mais atende os requisitos de *middleware* para cidades inteligentes, bem como alguns casos sucesso de uso dessa plataforma. O Capítulo 5 contém as considerações finais deste relatório.

2 Requisitos de Middleware para Cidades Inteligentes

Uma plataforma de *middleware* consiste em uma camada de *software* que reside entre a camada de aplicação e a infraestrutura de suporte (comunicação, processamento, sensoriamento), fornecendo acesso padronizado aos dados e serviços providos pelos objetos 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. Pires *et al.* [5] elencam um conjunto de requisitos de *middleware* para o paradigma de Internet das Coisas (ou IoT, do inglês, *Internet of Things*) [6, 7, 8]. Considerando que cidades inteligentes representam um domínio de aplicação do paradigma de IoT, uma plataforma de *middleware* para cidades inteligentes deve ter inerentemente os mesmos requisitos das plataformas voltadas para IoT, além de requisitos adicionais específicos desse domínio de aplicação.

Os requisitos fundamentais para plataformas de *middleware* voltadas para IoT, frequentemente mencionados na literatura, são: (i) interoperabilidade; (ii) descoberta e gerenciamento de dispositivos; (iii) adaptação dinâmica; (iv) ciência de contexto; (v) escalabilidade; (vi) tratamento de grandes volumes de dados, e; (vii) segurança. Além desses sete requisitos, outros dois novos foram elencados como importantes para o contexto de cidades inteligentes através de buscas e análise de trabalhos disponíveis na literatura, a saber, gerenciamento de dados e ferramentas para o desenvolvimento de aplicações. Esses trabalhos foram coletados a partir de três das principais bases bibliográficas internacionais na área de Computação (IEEEXplore[1], ACM Digital Library[2] e ScienceDirect.com[3]) utilizando a seguinte *string* de busca, que foi adaptada para a busca em cada uma das bases:

((*smart city* OU *smart cities* OU *smartcity* OU *smartcities*) E
(*middleware* OU *framework* OU *architecture*) E (*requirements* OU *requisites*))

Adicionalmente, no contexto de cidades inteligentes, o acesso a dados abertos é um fator chave para a criação de novos negócios, aplicações e serviços inovadores para a cidade, permitindo que esses dados possam ser analisados e gerados novos dados com um maior valor agregado. Assim, pode-se afirmar que a disponibilização de portais de dados abertos, é um requisito fundamental para uma cidade inteligente. Os portais de dados abertos devem ser integrados aos serviços das plataformas de *middleware*. A seguir, cada um dos requisitos supracitados será brevemente discutido.

2.1 Interoperabilidade

A interoperabilidade entre os diversos dispositivos, serviços, aplicações, sistemas, com as plataformas disponíveis no ambiente é um dos requisitos indispensáveis para uma

[1] http://ieeexplore.ieee.org/
[2] http://dl.acm.org/
[3] http://www.sciencedirect.com/

plataforma de *middleware* para IoT. A integração e comunicação de uma grande quantidade de dispositivos, serviços, aplicações e sistemas heterogêneos, tanto em termos de *hardware* quanto de *software*, protocolos, formatos de dados, etc, é um dos principais desafios para a concretização desse paradigma de IoT. Delicato et al. [9] destacam que a integração de dispositivos atinge diversos níveis, desde (i) um mais baixo nível, no qual é necessário integrar os dispositivos físicos de maneira transparente e ocultar os detalhes como relação à rede, formatos de dados empregados, e até mesmo à semântica das informações, passando por (ii) um nível intermediário, no qual é necessário integrar e disponibilizar dados providos pelos dispositivos, de forma a prover serviços de valor agregado aos usuários, até, por fim, (iii) um mais alto nível, no qual a agregação e transformação das informações dos dispositivos são providas por um modelo padronizado de programação, de forma a permitir que desenvolvedores de aplicações não necessitem ter o conhecimento acerca das especificidades dos dispositivos físicos e do ambiente de rede subjacente.

O contexto de cidades inteligentes não se resumo exclusivamente à integração de dispositivos heterogêneos, mas de sistemas inteiros heterogêneos e complexos que são integrados para prover novas funcionalidades. A título de exemplo, considere-se o cenário de monitoramento de rios que atravessam uma cidade utilizando principalmente sensores. Uma vez que esses sensores transmitem informações em tempo real acerca de variáveis ambientais tais como índices pluviométricos e o nível da água de rios, tais informações obtidas por sensores espalhados nas proximidades de um rio podem ser utilizadas de forma integrada com outros sistemas de informação (a exemplo de sistemas meteorológicos) e de tomada de decisão no intuito de prevenir inundações. Assim, a combinação de sensoriamento remoto para tarefas de monitoramento com mecanismos de ação e alerta pode compor um sistema complexo de larga escala para notificar autoridades e cidadãos acerca de um desastre iminente, que pode trazer danos à cidade e afetar a vida dos cidadãos. Tal sistema caracteriza-se como um *sistema de sistemas* (ou SoS, do inglês *System-of-Systems*), uma classe de sistemas resultantes da integração entre sistemas independentes que cooperam para formar um sistema maior e mais complexo a fim de oferecer novas funcionalidades [10]. Dessa forma, promover a interoperabilidade entre esses sistemas constituintes, que são geralmente distribuídos, independentes e desenvolvidos com diferentes tecnologias e para diversas plataformas é, portanto, um dos requisitos imperativos no contexto de cidades inteligentes [11].

2.2 Descoberta e Gerenciamento de Dispositivos

No contexto de cidades inteligentes há uma variedade de objetos (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 para alcançar objetivos comuns. É bastante comum que os dispositivos estejam em constante mudança, entrando e/ou saindo dos ambientes em que estão inseridos. Dessa forma, a topologia para infraestrutura de comunicação é dinâmica e frequentemente desconhecida, visto que os dispositivos podem ser integrados ao ambiente e

utilizados de maneira oportunista e não previamente planejada [12]. Portanto, a descoberta e gerenciamento de dispositivos é um outro requisito de igual importância para uma plataforma de *middleware* para IoT, de forma a permitir haja um esquema de endereçamento único para que os dispositivos possam ser descobertos e gerenciados, fornecendo informações a respeito de sua localização e do seu estado. Além disso, outras funcionalidades importantes são desconectar um 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.

2.3 Adaptação Dinâmica

Além da descoberta e gerenciamento dos dispositivos, a alta dinamicidade dos dispositivos no ambiente de IoT exige que plataformas de *middleware* promovam estratégias para adaptação dinâmica (ou reconfiguração dinâmica), i.e., a realização de mudanças na estrutura e/ou comportamento de um sistema enquanto ele está em execução a fim de garantir a sua disponibilidade e qualidade [13, 14]. Essas mudanças a serem realizadas devem respeitar algumas propriedades importantes, principalmente o fato de a adaptação não causar erro ou inconsistência na operação do sistema.

Adaptação dinâmica é um requisito é de essencial importância para plataformas de *middleware* para IoT pois os dispositivos podem tornar-se indisponíveis por diversos motivos como, por exemplo, falha, capacidade energética, indisponibilidade de conexão à rede, mobilidade de usuário, entre outros. A capacidade de se realizar adaptação dinâmica é particularmente importante principalmente em aplicações de domínios críticos, a exemplo de aplicações de saúde (*health care*), nas quais a falha de um dispositivo sem que a mesma seja identificada e corrigida pode causar danos a vida de um paciente. Portanto, a adaptação dinâmica tem como papel manter a aplicação disponível e funcionando adequadamente nesse ambiente dinâmico, coletando, analisando e reagindo de acordo com as mudanças no contexto em que as aplicações e os dispositivos estão inseridos.

2.4 Ciência de Contexto

Em um ambiente de IoT, os dispositivos não possuem apenas alta dinamicidade com relação a entrada e saída nos ambientes, mas também uma dinamicidade ainda maior com relação a variação das informações de contexto, i.e., qualquer informação que pode ser utilizada para representar uma pessoa, lugar ou objeto considerado relevante ao ambiente em questão [15]. Um ambiente de IoT é sensível ao contexto por natureza e, de um modo geral, a alteração da informação contextual associada a um dado dispositivo de IoT deve gerar reação de acordo com o contexto. Dessa forma, conhecer, examinar e reagir de acordo as mudanças de contexto, passado e presente, de forma a tentar antecipar possíveis ações no futuro, são características de extrema importância para um ambiente de IoT.

Pires et al. [5] apontam a ciência de contexto como outro requisito que deve ser atendido pelas plataformas de *middleware* para IoT, atribuindo a responsabilidade pela coleta, gerenciamento e processamento de informações de contexto providas por múltiplas fontes, tais como o estado do objeto, seus vizinhos e sua localização. A ideia é efetuar ações ou reagir a estímulos com base nos dados extraídos [16], liberando as aplicações e usuários da tarefa de manipulá-las, tornando transparente tal manipulação.

2.5 Escalabilidade

Um desafio importante a ser endereçado no contexto de IoT refere-se à enorme quantidade de dispositivos físicos através da rede. Com efeito, diversas previsões apontam que bilhões de dispositivos estarão conectados à Internet em 2020 e, por isso, infraestruturas de IoT necessitam ser escaláveis o suficiente para assimilar essa crescente quantidade de dispositivos heterogêneos e requisições, além de manter-se funcionando corretamente mesmo em situações de uso intenso.

Plataformas de *middleware* para IoT devem ser escaláveis, permitindo a alocação e liberação dos recursos computacionais conforme a demanda, ao mesmo tempo que mantém o sistema como um todo em estado operacional e com nível de desempenho satisfatório. Nesse cenário, o paradigma de Computação em Nuvem [17] surge como uma solução promissora para endereçar o problema da escalabilidade em ambientes de IoT, devido à sua capacidade de prover e liberar recursos computacionais sob demanda. Outro paradigma que recentemente tem ganho atenção na comunidade científica como meio de promover escalabilidade é o de *Fog Computing* (ou *Edge Computing*), que visa trazer o processamento para mais perto dos dispositivos de IoT, minimizando a latência e obtendo melhores taxas de tempo de resposta para usuários e aplicações [18, 19].

2.6 Tratamento de Grandes Volumes de Dados

O crescimento exponencial do número de dispositivos em um ambiente de IoT gera não somente problemas relacionados a escalabilidade, mas também resulta no crescimento gigante do volume de dados providos e transmitidos através da rede. Esse cenário exige que as plataformas de *middleware* sejam capazes de tratarem esses grandes volumes de dados. Dessa forma, o tratamento de grandes volumes de dados é mais um requisito indispensável para as plataformas de *middleware* para IoT, de forma que as plataformas possam acompanhar a demanda de coleta e análise de dados e, consequentemente, prover respostas, decisões e/ou atuações de maneira eficiente.

Novos desafios surgem como desdobramentos desse requisito tais como 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. *Big Data* [20, 21] e Computação em Nuvem têm surgido como potenciais propostas

para solucionar alguns desses desafios, permitindo lidar com um imenso volume de dados, diversos e não estruturados.

2.7 Segurança

Por diversas vezes os dispositivos integrados têm como função coletar dados (privados ou não) e estes podem ser transportados por redes com segurança de baixa qualidade. Por isso, uma plataforma de *middleware* deve ter, sem dúvidas, uma segurança adequada para que seja preservada a integridade e privacidade desses dados, assim como, proteger os dispositivos e recursos que são expostos à rede. Assim, algumas técnicas como *tamperproofing* e ofuscação de código são usadas para dar segurança aos dispositivos. Já para os recursos, são usadas as estratégias de bloqueio de portas abertas e não usadas, e o uso de protocolos, sejam eles de: segurança, de autorização e/ou autenticação.

2.8 Gerenciamento de Dados

A forma de acesso e organização dos dados providos pelos milhares de dispositivos, serviços e aplicações inerentes a um ambiente de cidade inteligente são detalhes importantes e que devem ser endereçados pelas plataformas de *middleware*. Contudo, além do tratamento do grande volume de dados providos por uma cidade inteligente, o gerenciamento desses dados caracteriza-se como sendo mais um requisito fundamental para uma plataforma de *middleware* voltada para cidades inteligentes. O gerenciamento de dados envolve a definição da forma como os dados serão acessados, podendo prover acesso de forma síncrona ou assíncrona, como também a definição de como os dados serão organizados, definindo padrões e seus metadados descritivos associados.

2.9 Ferramentas para Desenvolvimento de Aplicações

Como em qualquer outro tipo de plataforma que permita o desenvolvimento de aplicações, a disponibilização de ferramentas que auxiliem no processo de desenvolvimento e criação de novas aplicações e serviços, é algo extremamente importante. Assim, a disponibilização dessas ferramentas para o desenvolvimento de aplicações é mais um requisito que deve ser atendido por plataformas de *middleware* para cidades inteligentes. Ferramentas podem ser desde uma API que permita a criação e uso dos serviços providos pelas plataformas, até mesmo um conjunto de ferramentas para desenvolvimento ou uma interface gráfica que auxilie a criação das aplicações.

3 Plataformas de *Middleware* para Cidades Inteligentes

Essa seção apresenta plataformas de *middleware* que têm sido utilizadas no desenvolvimento de aplicações para cidades inteligentes. A Seção 3.1 apresenta a plataforma EcoDiF, cujo protótipo foi testado no contexto de aplicações para o domínio de cidades inteligentes. A Seção 3.2 apresenta a plataforma Kaa, um *middleware* de código aberto para o desenvolvimento, gerenciamento e integração de aplicações de IoT. A Seção 3.3 apresenta SOFIA, uma arquitetura de *middleware* que usa o conceito de ambiente inteligente, que representa um ecossistema de objetos que interagem. A Seção 3.4 apresenta o *Future Internet Ware* (FIWARE), uma plataforma genérica e extensível proposta com apoio da Comunidade Europeia e que define um conjunto de especificações abertas disponibilizadas por interfaces de aplicações (APIs, do inglês *Application Programming Interfaces*) e implementadas em componentes denominados habilitadores genéricos (*generic enablers*). A Seção 3.6 apresenta a plataforma CityHub, utilizada pelas cidades de Vancouver, no Canadá, e Lancaster, no Reino Unido, possibilitando a criação e implantação de soluções de *software* na área de cidades inteligentes. Por fim, a seção 3.7 apresenta uma tabela que mostra uma visão geral os requisitos supracitados e posiciona as plataformas descritas quanto ao suporte a cada um dos requisitos.

Além das plataformas analisadas neste trabalho, Pires et al. [5] analisaram onze plataformas de IoT, aplicáveis para o contexto de cidades inteligentes, e reportaram como cada uma delas atende aos requisitos para IoT. Através da análise das plataformas EcoDiF, Xively, Carriots, LinkSmart, OpenIoT, RestThing, WoT Enabler, S^{3}OIA, UbiWare, WSO2 e INRIA ARLES, os autores concluíram que nenhuma dela atende a todos os requisitos importantes a serem endereçados por plataformas de *middleware* para IoT. No presente trabalho, outras quatro plataformas são analisadas, Kaa, Sofia, FIWARE e CityHub. A justificativa de incluir a EcoDiF nesse relatório é o fato de ter sido desenvolvida com participação da UFRN, em parceria com a UFRJ, o que faz ter um domínio total da sua implementação e permite o seu reuso parcial ou completo no contexto do Projeto *SmartMetropolis*.

3.1 EcoDiF

A EcoDiF (*Ecossistema Web de Dispositivos Físicos*) [22] é uma plataforma Web aberta que objetiva integrar dispositivos físicos heterogêneos com aplicações e/ou usuários e permitir o fácil gerenciamento de tais dispositivos e dos dados por eles providos. Os dados providos pelos dispositivos físicos e disponibilizados através da plataforma podem ser consumidos tanto por usuários quanto por aplicações, que inclusive podem, a partir dos mesmos, gerar novas informações a serem registradas na plataforma.

A EcoDiF foi projetada tendo como base princípios definidos no estilo arquitetural REST (*REpresentational State Transfer*) [23], utilizando assim o protocolo HTTP (*HyperText Transfer Protocol*) como mecanismo padrão de suporte a todas as interações entre os objetos endereçáveis através de suas principais operações (GET, PUT, POST, DELETE). A

plataforma provê, dessa forma, uma interface bem definida para expor as funcionalidades dos dispositivos físicos integrados na Web, padronizando e simplificando o desenvolvimento de aplicações, bem como suplantando limitações com relação à incompatibilidade entre diferentes fabricantes de dispositivos, protocolos proprietários e formatos de dados [24]. 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 altamente distribuídos e heterogêneos.

A arquitetura lógica da EcoDiF é composta de diversos módulos, ilustrados na Figura 2 e apresentados a seguir.

Figura 2. Arquitetura lógica 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. Na EcoDiF existem dois tipos de *drivers*, síncronos e assíncronos. *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* assíncronos (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*) [25], 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 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. 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* [26], 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*) [27], 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 validada através de um conjunto de aplicações desenvolvidas como provas de conceito em diferentes cenários reais [5], 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. A primeira aplicação tem como objetivo permitir o monitoramento remoto e integrado de diversos equipamentos (servidores, dispositivos de rede, dispositivos de armazenamento, etc.) em centros de processamento de dados (CPDs) e notificar gerentes em caso de possíveis anormalidades em sua operação. A segunda aplicação, inspirada no conceito de sensoriamento participativo (*participatory sensing*, em inglês) [28], visa informar, através de um mapa de presença, a quantidade de pessoas nas salas de um centro de convenções no qual está ocorrendo uma conferência. Esse dado é importante para um melhor provimento recursos e ajustes de iluminação, som, temperatura do ambiente de acordo com a quantidade de pessoas neles presentes. A terceira aplicação possui como objetivo monitorar, em tempo real, o fluxo de água e gás em redes de dutos de companhias públicas da cidade do Rio de Janeiro, possibilitando que operadores e gerentes das companhias proprietárias dos dutos visualizem as últimas medidas aferidas pelos sensores e serem notificados em caso de condições anormais de operação quando, por exemplo, os valores mensurados indicam um possível vazamento de água/gás no duto. Por fim, a quarta aplicação tem como finalidade fornecer e agregar informações acerca das condições atuais de saúde de pacientes monitorados remotamente através de sensores corporais a eles implantados para coleta de sinais vitais, permitindo também visualizar dados históricos mensurados, disparar alertas em caso de condições anormais e também contribuir para melhor diagnóstico por parte de equipes médicas.

3.2 Kaa

Kaa [29] é uma plataforma de *middleware* de código aberto para o desenvolvimento, gerenciamento e integração de aplicações fim-a-fim completas de IoT possibilitando a comunicação bidirecional entre dispositivos de qualquer tipo através do gerenciamento proporcionado pela plataforma. Além do gerenciamento da comunicação entre os dispositivos, a plataforma apresenta diversas facilidades para a integração com ferramentas responsáveis pelo armazenamento, tratamento e análise dos dados obtidos, além de já contar com elementos que garantem a segurança e integridade dos dados, e suportar o balanceamento de carga entre nós e gerenciamento de servidores ativos.

Entre as principais funcionalidades oferecidas pela plataforma estão [30]:

- Mecanismos de entrega de mensagens de eventos entre dispositivos conectados (em transmissão *unicast* ou *multicast*);
- Armazenamento temporário de registros de dados (*logs*) em qualquer estrutura previamente definida com o *upload* periódico ao servidor;
- Presença de sistema de notificação baseado em tópicos que permite a entrega de mensagens em formato predefinido a clientes que assinaram o tópico;
- Uso do conceito de perfil *endpoint*, composto pelos lados perfil cliente (conjunto de dados expostos pelo cliente para acesso por uma *aplicação Kaa*) e perfil servidor

(conjunto de dados que é controlado pelos usuários do *servidor Kaa* através de interface administrativa ou de uma API REST por outras aplicações servidoras).
- Permite a realização de atualizações de dados operacionais (como dados de configuração) do servidor para os clientes;
- Conta com uma arquitetura que abstrai os canais utilizados para a distribuição de dados entre *endpoints* e o servidor;
- Possibilita que sejam servidas múltiplas entidades de negócio e aplicações independentemente em uma única instância de servidor.

A plataforma facilita a troca de dados entre dispositivos conectados, a nuvem de IoT criada com o uso da plataforma, analisadores e sistemas de visualização de dados e outros componentes que compõe o ecossistema de IoT. Essa comunicação pode ser realizada entre dispositivos de diferentes naturezas e fabricantes, através das integrações já existentes da plataforma com principais plataformas atuais ou com a utilização do SDK disponibilizado atualmente para as linguagens Java, C e C++ para a realização dessa comunicação com plataformas ainda não integradas.

Essa capacidade de comunicação entre dispositivos heterogêneos é possível devido a habilidade de manipulação de modelos de dados estruturados e não estruturados pela plataforma, fazendo com que detalhes de implementação específicos de cada plataforma sejam abstraídos durante a comunicação, além de possibilitar a realização do armazenamento, manipulação e análise desses dados a partir de integrações existentes da plataforma com ferramentas de terceiros usadas para tais finalidades, como bancos NoSQL (como MongoDB e Cassandra), ferramentas de processamento de dados (como Apache Spark e Hadoop), entre outras ferramentas.

A plataforma possui uma arquitetura composta pelo *servidor Kaa* e por *SDKs endpoints*. O *servidor Kaa* é responsável pela implementação da parte de backend, além de expor interfaces para integração e oferecimento de capacidades administrativas. O *SDK endpoint* pode ser usado para definir *clientes Kaa*, que são instalados em dispositivos que desejem utilizar as funcionalidades da plataforma, sendo responsáveis pelo processamento dos dados estruturados provenientes do *servidor Kaa* (como configurações e notificações, por exemplo) e fornecer dados para as interfaces de retorno (como logs, por exemplo).

Um *cluster Kaa* consiste em *nós Kaa* que utilizam o *Apache Zookeper* para a coordenação de serviços. Um *cluster Kaa* também requer instâncias de bancos de dados SQL e NoSQL. Um *nó cluster Kaa* é compreendido pelos serviços de *Controle*, *Operações* e *Bootstrap*, que podem ser atividados ou desativados individualmente por um *administrador Kaa* (Figura 3).

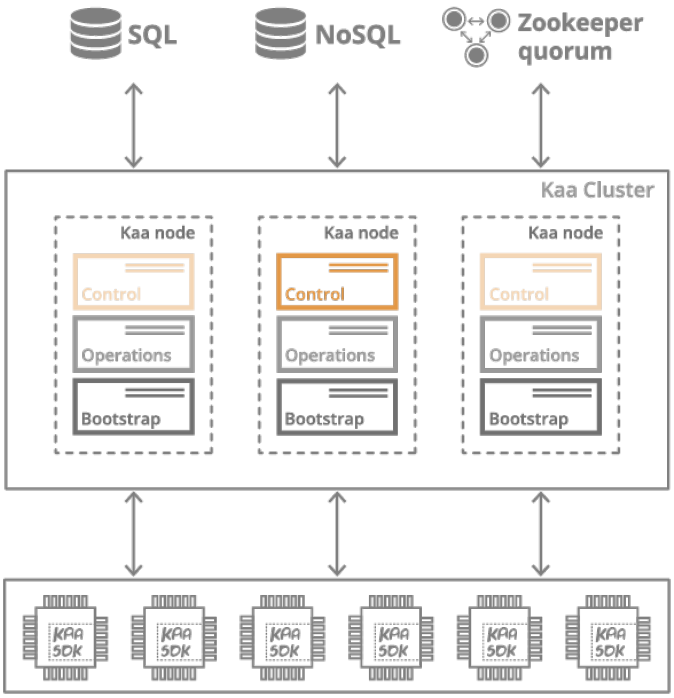

Figura 3. Arquitetura de alto nível Kaa

O serviço de *Controle* é responsável pela gestão global do sistema de dados, processamento de chamadas à API provenientes de uma UI web ou sistemas externos integrados, e entrega de notificações a servidores de *Operações*. O serviço de *Operações* é um serviço "operário" responsável por concorrentemente manipular múltiplas requisições de múltiplos clientes, como registro de *endpoints*, entrega de notificações, entre outras. O serviço de *Bootstrap* é responsável pelo direcionamento de *endpoints* para serviços de *Operação*.

Uma *instância Kaa* (Figura 4) é uma implementação particular da plataforma e consiste em um *cluster Kaa* (representando um conjunto de nós de *servidores Kaa* interconectados) e *endpoints*. Um *endpoint* é uma abstração para representar uma entidade gerenciada separadamente dentro de uma instância, ou seja, um cliente Kaa registrado, ou aguardando registro, dentro de uma instância. Ele é uma aplicação Kaa que reside em um dispositivo em particular conectado e utiliza o SDK cliente para comunicar-se com o servidor Kaa, gerenciar dados locais na aplicação cliente bem como prover APIs de integração.

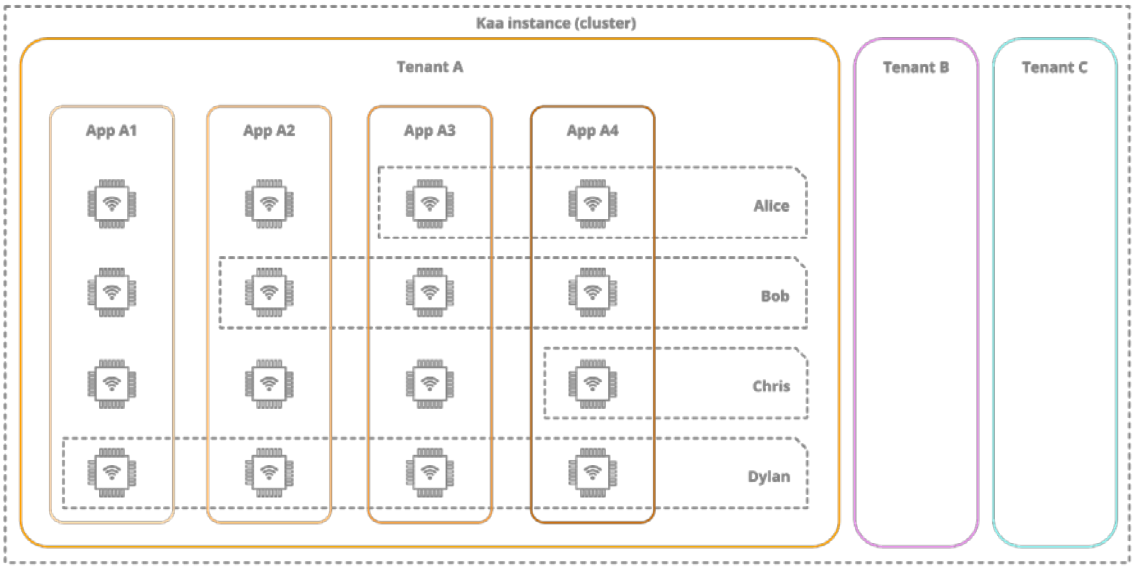

Figura 4. Arquitetura de uma instância Kaa (cluster)

Uma *aplicação Kaa* representa uma família de implementações disponíveis de uma aplicação de software específica usada pelos *endpoints*. Um *inquilino* é uma entidade de negócio separada que possui seus próprios *endpoints*, aplicações e usuários. Uma *instância Kaa* pode suportar múltiplos *inquilinos*, com múltiplas aplicações em cada um deles.

Embora seja uma plataforma relativamente nova relativamente nova, e ainda não muito difundida e amplamente utilizada, o *Kaa* tem como principais benefícios a possibilidade de realizar a comunicação, controle e monitoramento de dispositivos de diferentes naturezas, por meio de uma arquitetura que possibilita a escalabilidade quanto ao número de dispositivos, abstraindo também detalhes de como é realizada essa comunicação entre eles e possibilitando a integração com diversas ferramentas que possibilitam a análise dos dados obtidos de forma melhor.

3.3 SOFIA

SOFIA (*Smart Objects for Intelligent Applications*) [31] é uma plataforma que resultou do projeto ARTEMIS, com duração de três anos e concluído em março de 2012. Nove participantes de quatro países da Europa, incluindo Nokia, Philips, Fiat, Indra, estiveram envolvidos neste projeto. A plataforma foi construída em torno do conceito de um ambiente inteligente, que representa um ecossistema de objetos que interagem: sensores, dispositivos móveis, aplicações, sistemas embutidos. Estes objetos têm a capacidade de se autoorganizarem, oferecer serviços federados, processar e fornecer dados complexos.

O propósito de SOFIA é ser uma arquitetura de *middleware* que permita a interoperabilidade de diversos sistemas e dispositivos. Esse conjunto de sistemas forma um ambiente inteligente onde o armazenamento da informação é aberto e distribuído e os procedimentos de busca são comuns a todas as aplicações, independentemente das tecnologias

utilizadas no desenvolvimento das mesmas. A plataforma visa permitir a conexão entre o mundo físico e o mundo da informação, através da disponibilização desta informação para serviços inteligentes. Além disso, também visa promover a inovação através da criação de novos conceitos de interface e interação com o usuário, enriquecendo sua experiência e permitindo que se beneficiem dos ambientes inteligentes.

SOFIA é *open-source*; multi-linguagem, portável para Java, C++, JavaScript, Arduino; multi-plataforma, disponível para Windows, iOS, Android e Linux; comunicação agnóstica. A plataforma possui implementações dos protocolos TCP, MQTT, HTTP, REST e WebServices.

Após a finalização do projeto ARTEMIS, a companhia Indra continuou a evoluir o projeto original de SOFIA, criando uma plataforma que tem foco no uso empresarial. A versão atual dessa plataforma é chamada SOFIA2. A plataforma manteve os principais conceitos de SOFIA, incluindo novas características, como: gestão de emergências e evacuações, automação cooperativa de construções, infraestruturas públicas e processos industriais, IoE (Internet of Energy), gerenciamento inteligente de redes de energia através da internet de forma segura, aperfeiçoamento de interfaces homem-máquina para tecnologias integradas em rede.

Dentre as funcionalidades da plataforma podemos citar:

- Coleta de dado de sensores e dispositivos presentes numa cidade inteligente
- Integração com sistemas existentes na cidade
- Auxílio no processo de tomada de decisão através da avaliação de regras e da CEP Engine
- Subscrição de eventos, alarmes, etc
- Suporte a múltiplos dispositivos
- Coleta de informação de dispositivos presentes em residências
- Armazenamento, processamento e tomada de decisão sobre grandes volumes de informação
- Gerenciamento dos diversos dispositivos presentes em residências
- Regras para ações a serem tomadas com base em eventos
- Funciona como um canal de comunicação entre sistemas de saúde
- Armazenamento de dados históricos
- APIs para diferentes tipos de plataformas e linguagens

A plataforma SOFIA pode ser conceitualizada através de quatro conceitos:

O **Smart Space** representa um ambiente virtual onde diferentes aplicações e dispositivos interagem uns com os outros para fornecer uma funcionalidade mais complexa. O núcleo do Smart Space é o SIB, geralmente ele possui apenas um SIB, mas pode em alguns casos possuir uma federação de SIBs. Um Smart Space pode se comunicar com outros Smart Spaces.

O **SIB (Semantic Information Broker)** é responsável por receber, processar e armazenar toda a informação de aplicações conectadas na plataforma, agindo como uma espécie de barramento. Todos os conceitos existentes no domínio (refletidos nas ontologias) e seus estados atuais (instâncias específicas das ontologias) estão refletidos no SIB. A plataforma propõe o uso de JSON para troca de informações e definição de ontologias.

Há implementações de SIBs em várias linguagens e plataformas, os SIBs oferecem diferentes tipos de conectores para comunicação com diferentes tipos de clientes: REST, para clientes que usam Java Script e smartphones; MQTT, para comunicações bidirecionais e dispositivos limitados; Web Services/JMS para aplicações empresariais; e outros, incluindo Bluetooth e Zigbee.

O **KP (KnowLedge Processor)** representa cada elemento (aplicação ou dispositivo) que interage com o Smart Space através de um SIB. Cada aplicação funciona com instâncias de conceitos relevantes no domínio (representados através de ontologias) para o qual foi designada. Há três tipos de KP: Produtor, é um KP que insere informação no SIB, mas não recupera nenhuma informação dele; Consumidor, é um KP que recupera informação do SIB, mas não insere nenhuma informação nele; Prosumidor (*prosumer*), é um KP que insere no SIB e também recupera informação.

O **SSAP (Smart Space Access Protocol)** é um protocolo de mensagem padrão para a comunicação entre SIBs e KPs. Este protocolo é independente do tipo de rede utilizada (GPRS, 3G, WIFI, Bluetooth, HFC, Zigbee). Há dois tipos de implementação do protocolo: SSAP-XML, que usa o formato XML quando há uma largura maior de banda; SSAP-JSON, as mensagens são adaptadas para o formato JSON quando a comunicação é com dispositivos móveis, web browsers, etc.

A Figura 5 mostra como ocorre a interação dos quatro conceitos apresentados anteriormente. Podemos observar, na Figura, que diferentes tipos de KP interagem com o Smart Space através dos SIBS, utilizando o protocolo SSAP. Os KPs podem se conectar a mais de um Smart Space, a fim de recuperar ou inserir informações. E assim, Smart Spaces podem se comunicar a outros Smart Spaces. Através desta Figura vemos claramente a proposta de SOFIA, que é permitir a conexão entre o mundo físico, representado pelos KPS e o mundo da informação, representado pelo Smart Space.

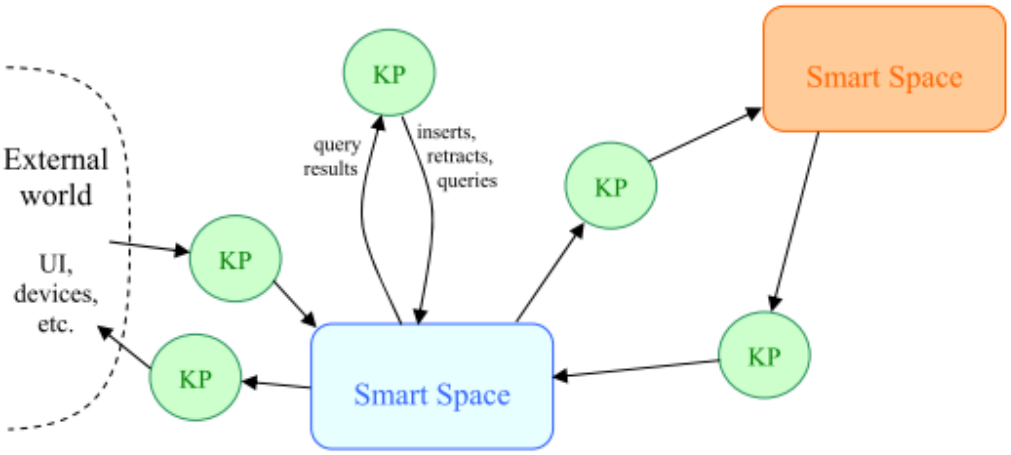

Figura 5 - Principais Conceitos de SOFIA

A Figura 6 mostra algumas operações definidas pelo protocolo para comunicação entre um KP e o SIB. Podemos ver na figura que há dois tipos de KPs interagindo com o SIB. O KP consumidor pode realizar operações definidas pelo protocolo para recuperar informação do SIB. A operação JOIN, por exemplo, é utilizada para que o KP se conecte ao SIB (requer autenticação, autorização e a criação de uma sessão no Smart Space). A operação LEAVE é utilizada quando o KP quer desconectar do SIB, a operação QUERY é usada quando o KP quer recuperar informação do SIB, esta informação pode vir de um banco de dados com dados históricos ou dados em tempo real.

O KP produtor, por sua vez, realiza operações para inserir informação no SIB. Como podemos ver na Figura 6, ele realiza as operações JOIN e LEAVE, com o KP consumidor, e realiza operações especificas para inserção, atualização e remoção de dados no SIB, representadas pelas operações INSERT, REMOVE e UPDATE na figura.

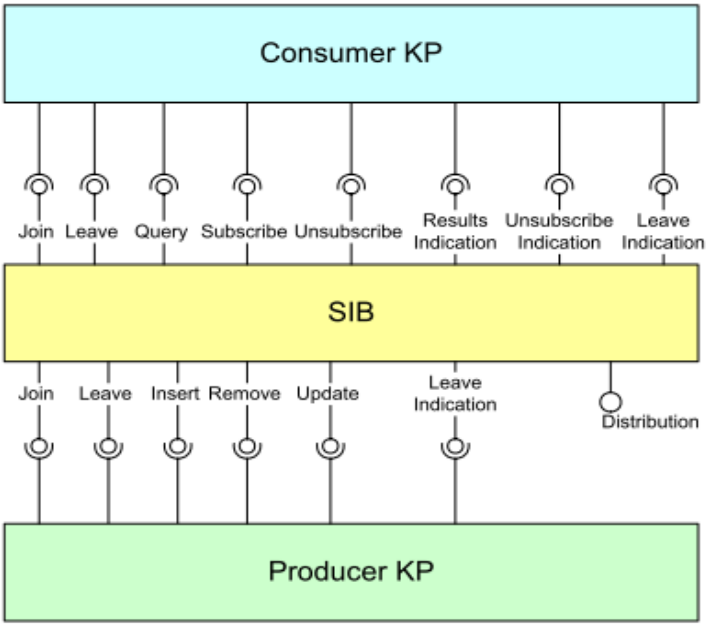

Figura 6 - Operações SSAP

A Figura 7 mostra a arquitetura lógica de SOFIA e seus principais componentes. O componente *Sofia2 Router* oferece conectores multiprotocolo (REST, MQTT, WS, ...) para que os sistemas possam inserir dados. Ele possui o adaptador ISpeed, que integra informações em tempo real. O *Sofia Brain* é o núcleo da plataforma, encarregado de orquestrar e aplicar regras às mensagens recebidas pela plataforma. *Sofia Integral DB* representa o conjunto de repositórios de dados gerenciados pela plataforma.

Sofia2 Config é responsável pelo gerenciamento de informação de toda a plataforma (Web Console mais interfaces REST). O componente *Sofia Security* gerencia os aspectos de segurança da plataforma, como autenticação e autorização de dados e processos. *Sofia API Manager* permite a publicação de dados através de uma API REST e o *Sofia Developer Kit* é o portal do desenvolvedor, oferecendo documentação, exemplos e demonstrações de uso das APIs.

Algumas aplicações para cidades inteligentes foram desenvolvidas utilizando os recursos que SOFIA oferece. O *Cyber-Physical Systems Engineering Labs* (CPSE Labs) é uma iniciativa financiada pela União Europeia para apoiar empresas europeias, oferecendo financiamento para atividades de inovação e conectando essas empresas a um dos melhores institutos de pesquisa da Europa. Utilizando SOFIA, o CPSE desenvolveu um sistema de turismo, como estudo de caso, para as cidades Rías Baixas, Coruña e Pontevedra. Este sistema permite que autoridades locais numa cidade inteligente, entendam quais as necessidades dos turistas visitantes, e saber o nível de satisfação dos turistas [32].

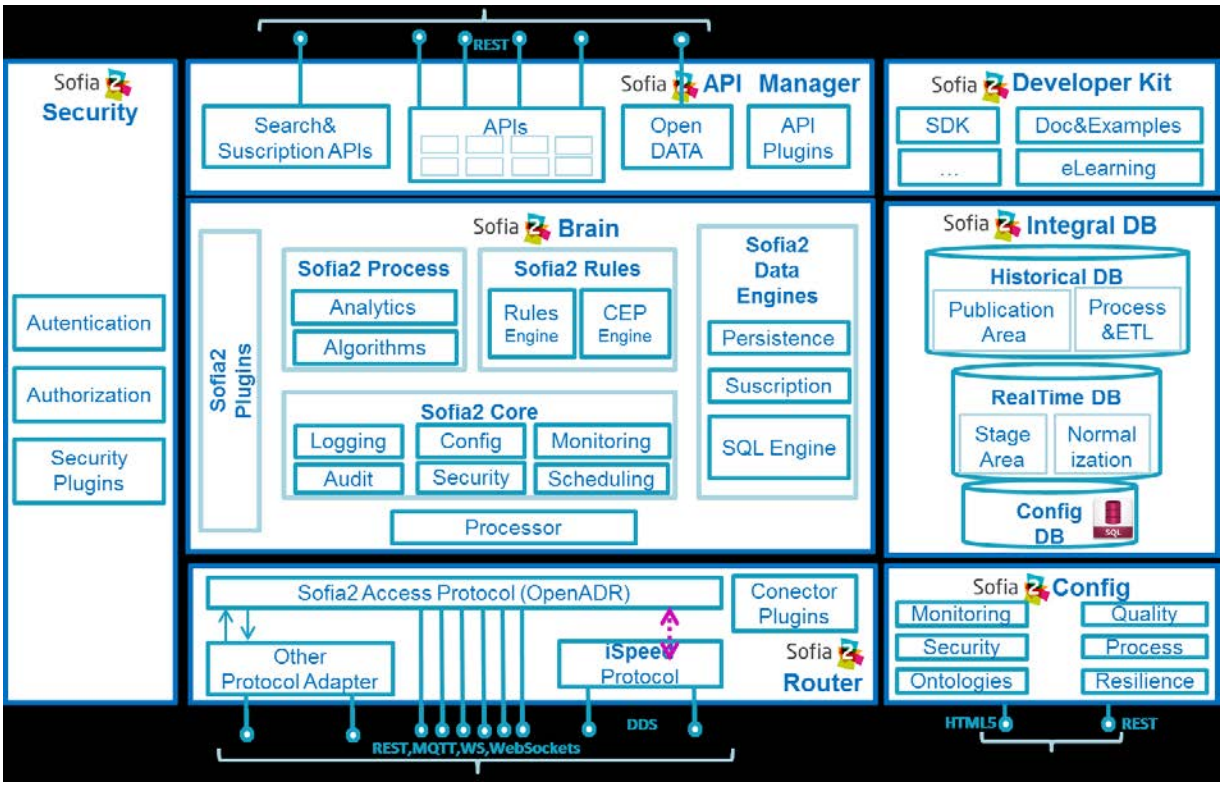

Figura 7 - Arquitetura Lógica de SOFIA

Na cidade de Toledo, Espanha, o CPSE, desenvolveu um sistema que foi implantado no National Hospital for Paraplegic People, que permite que os locais dos pacientes dentro do hospital, sejam monitorados em tempo real, a partir de um centro do monitoramento de controle centralizado [33]. Outro exemplo do uso de SOFIA pelo CPSE foi na cidade de Valparaíso, Chile, onde foi implementado um marketplace para operadores logísticos, que facilita o monitoramento e adequação entre oferta e procura de serviços portuários em toda a cidade, com as condições contratuais associadas a cada serviço [34].

No site oficial de SOFIA há demonstrações do uso da plataforma para diversos propósitos, como dashboard aplicado a área da saúde, visualização de dados, entre outros [35]. A cidade de Coruña em 2014 possuía 14 projetos pilotos utilizando SOFIA, dentre eles: Um sistema inteligente de irrigação, sistema de informação de eventos, guias para turistas visitantes, sistemas para melhorar a eficiência do uso de energia em prédios, etc. [36]

Dos requisitos de middleware para cidades inteligentes apresentados na seção 2 deste relatório, SOFIA atende a todos, exceto o requisito de ciência de contexto, subseção 2.4. Apesar da plataforma não dá suporte direto a este requisito, o trabalho [37] propôs uma microarquitetura para ciência de contexto, utilizando alguns componentes de SOFIA. SOFIA foi construída para fornecer interoperabilidade, sua arquitetura permite que vários tipos de dispositivos e aplicações acessem os Smart Spaces e se comuniquem uns com os outros, consumindo e produzindo informação, através dos componentes *Sofia2 Router* e *Sofia2 API*

*Manage*r. A plataforma também dá suporte ao gerenciamento de múltiplos dispositivos. O componente *Sofia2 Security* garante que a plataforma atende ao requisito de segurança.

SOFIA também tem suporte ao gerenciamento e processamento de grandes volumes de dados, o componente Sofia2 Integral DB representa um conjunto de repositórios que armazena dados históricos, em tempo real e dados de configuração, que a plataforma gerencia. A plataforma possui suporte a adaptação dinâmica, é escalável e provê ferramentas para o desenvolvimento de aplicações, através do componente *Sofia Developer Kit*.

SOFIA utiliza ontologias para definir as entidades de informação, essas ontologias são escritas em formato JSON. Os componentes *Sofia2 Route*, *Sofia2 API Manage*r, e *Sofia2 Config* garantem que o requisito de Gerenciamento de Dados é atendido pela plataforma, o primeiro oferece conectores para que os KPs possam inserir ou acessar dados, o segundo permite acesso aos dados quando APIs REST são disponibilizadas para as ontologias e o *Sofia2 Config* gerencia toda a plataforma.

3.4 FIWARE

A Internet do Futuro (do inglês, *Future Internet* - FI) agrega as pesquisas direcionadas a construção de uma nova arquitetura para basear o futuro da internet [38, 39]. Ela abrange desde aspectos tecnológicos relacionados à infraestrutura física, redes sem fio, redes móveis, protocolos de rede, assim como, modelos de negócio e aplicações/dispositivos inteligentes (computação ubíqua). A Comissão Europeia[4], um órgão executivo da União Europeia, consciente dos desafios apresentados na formação da FI, lançou uma parceria público privada voltada para o futuro da Internet, a *Future Internet PPP*[5] (FI-PPP). Através da FI-PPP foi criado o projeto *Future Internet Ware* (FI-WARE) com o objetivo de construir uma plataforma chave para a internet do futuro. Assim, surgiu a plataforma FI-WARE, ou simplesmente FIWARE [40].

A plataforma FIWARE surgiu com o objetivo de construir tecnologias voltadas para Internet do Futuro. A internet atual foi concebida como uma pequena rede direcionada a uma pequena comunidade de pesquisadores. Inicialmente, em 1985, continha cerca de 50 páginas e 1000 máquinas conectadas. O crescimento acelerado da Internet conhecida hoje foi impulsionado por avanços nas tecnologias de comunicação em rede (fibra ótica, redes sem fio, 3G, 4G), novos computadores e dispositivos móveis, chegando a atingir a marca de três bilhões de usuários, em maio de 2015. O elevado e rápido desenvolvimento incluíram ainda o surgimento de comércio eletrônico, redes sociais, aplicações em tempo real, computação em nuvem, internet das coisas, bem como, o consumo não só de páginas web mais de mídias web, serviços web, aplicações, etc. Em consequência desse aumento astronômico, as tecnologias atualmente empregadas na internet têm apresentado um esgotamento de recursos, isto é, uma

[4] http://ec.europa.eu/index_pt.htm
[5] https://www.fi-ppp.eu/

necessidade de mudança no panorama atual [17, 19], criando uma oportunidade para o desenvolvimento de novas tecnologias.

O objetivo principal da plataforma FIWARE é ofertar uma plataforma genérica e extensível para serviços de Internet do Futuro por meio de um conjunto de especificações abertas, disponibilizadas por interfaces de aplicações (do inglês: *Application Programming Interfaces*, APIs) e implementadas em componentes denominados habilitadores genéricos (do inglês: *Generic Enablers*, GEs). O desenvolvimento dos GEs tem como objetivo compor o núcleo da plataforma FIWARE. Formalmente, os GEs são as peças basilares à construção funcional do FI-WARE. As implementações de cada GE constituem-se por um conjunto de componentes que, juntos, suportam um conjunto concreto de funcionalidades fornecidas através de APIs, provendo assim, interfaces interoperáveis em conformidade com as especificações abertas definidas para cada GE. A especificação de um GE deve conter toda a informação necessária para a construção de produtos compatíveis e capazes de trabalhar com os outros GEs, assim como, a possibilidade de substituir uma implementação de referência de um GE desenvolvida pelo próprio FIWARE. Nesse intuito, a especificação de um GE deve incluir, embora não esteja limitada, as informações:

1. Descrição do escopo, evidenciando o comportamento e a utilização pretendida do GE;
2. Termos, definições e abreviaturas para tornar claro os significados da especificação;
3. Assinatura e comportamento das operações que serão expostas pelas APIs; as assinaturas das APIS podem ser especificadas e uma linguagem particular ou no modelo REST;
4. Descrição de protocolos para interação com outros GEs;
5. Descrição de características não funcionais.

Os *Generic Enablers* estão agregados nos capítulos técnicos (do inglês: *Technical Chapters*) que estão relacionados a um conjunto de funcionalidades especificas providas pela FIWARE, por exemplo, segurança, infra-estrutura, etc. Atualmente, existem sete capítulos técnicos referentes ao release 4 da especificação da FIWARE, são estes: *Cloud Hosting*; *Data/Context Management*; *Internet of Things (IoT) Services Enablement*; *Applications, Services and Data Delivery*; *Security*; *Interface to Networks and Devices (I2ND) Architecture* e *Advanced Web-based User Interface*. Os capítulos técnicos são, também, os elementos delineadores da arquitetura da FIWARE exibida na Figura 8.

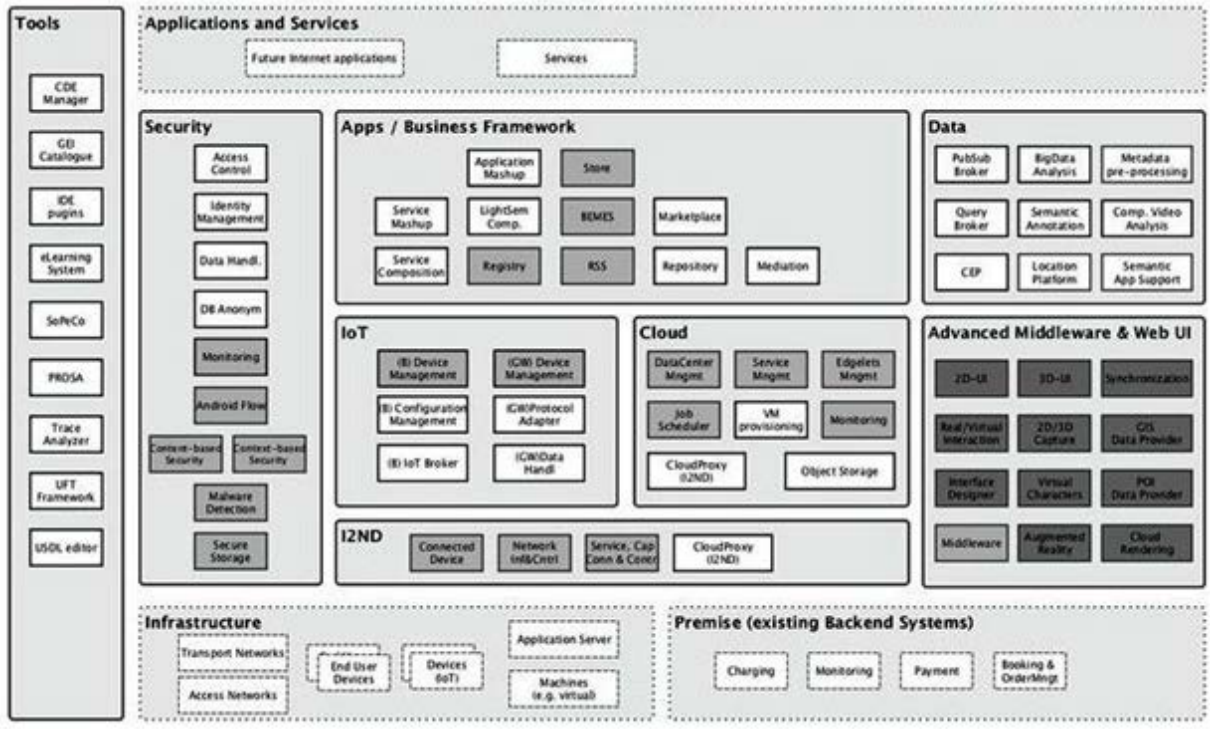

Figura 8 - Arquitetura FIWARE, Release 4.

A arquitetura FIWARE (Fig. Z) é apresentada como uma grade de quadros de borda completa representativos dos capítulos técnicos (quadros: *Security*, *I2ND*, *IoT*, *Cloud*, *Apps / Business Framework*, *Data*, *Advanced Middleware & Web UI*) e das ferramentas (quadro *Tools*) auxiliares ao desenvolvimento voltado aos GEs da FIWARE. Os quadros adjacentes de borda pontilhada representam elementos externos que são utilizados pela FIWARE (quadros: *Infrastructure* e *Premise*) ou que utilizam a FIWARE como suporte (quadro *Applications and Services*). A descrição detalhada dos capítulos técnicos, bem como, o suporte da FIWARE aos requisitos apresentados no Capítulo 2 serão abordados no Capítulo 4 deste relatório.

A ampla arquitetura, os componentes abertos e o alto investimento da FI-PPP têm proporcionado a FIWARE uma crescente área de atuação com abrangência internacional. Aliado a isso, a grande diversidade dos GEs, bem como, o foco na provisão de elementos utilizáveis por vários domínios levou a FIWARE a níveis de intercontinentalidade. A vista dessas qualidades, o portal acessível em [30] apresenta ferramentas de sucesso (vide exemplos na subsessão 4.1) que já utilizam os recursos da FIWARE e que estão presentes em 14 países diferentes, a saber: Bélgica, Brasil, Dinamarca, Finlândia, França, Alemanha, Grécia, Irlanda, Itália, Holanda, Portugal, Eslováquia, Eslovênia, Espanha e Reino Unido. Os campos de atuação das ferramentas são ainda mais diversos, indo desde agricultura, saúde, energia e meio ambiente a transporte, segurança urbana, tecnologia da informação e mídias sociais.

Não obstante o sucesso já atingido pela FIWARE, tem-se a iniciativa *The Open & Agile Smart Cities* (OASC) que foi assinada por 31 cidades dos países: Finlândia, Dinamarca,

Bélgica, Portugal, Itália, Espanha e Brasil. O objetivo da OASC é incentivar as cidades a adotar uma API padrão de licença aberta e implementada na plataforma FIWARE com o intuito de fornecer um meio leve e simples para reunir, publicar, consultar e inscrever-se a informação de contexto descrevendo o que acontece na cidade a qualquer tempo.

Dessa forma, a utilização da FIWARE como *middleware* para cidades inteligentes é totalmente encorajada e, como em alguns casos, já ocorre de forma proeminente. Outro ponto altamente relevante à adoção da FIWARE como *middleware* para cidades inteligentes deve-se ao fato dela satisfazer a todos os requisitos apresentados na seção 2 deste documento.

Os requisitos de Interoperabilidade, Descoberta e Gerenciamento de Dispositivos, Adaptação Dinâmica e Ciência de Contexto são atendidos pelos GEs dos Capítulos *Data/Context Management* e *Internet of Things (IoT) Services Enablement.* Em específico, os requistos de interoperabilidade, descoberta e gerenciamento de dipositivos são atendidos pelos GEs *DeviceManagement* e *ProtocolAdapter*. A adaptação dinâmica fica sob a responsabilidade *IoTDiscovery* e a ciência de contexto pelos *IoTBroker*, *Publish/Subscribe Broker* e *Complex Event Processing*.

A escalabilidade é atendida pelo do Capítulo de *Cloud Hosting*, principalmente pelos GEs: *IaaS* e *Cloud Application Management Service*. Outro Capítulo relacionado diretamente a um requisito é o *Security*. Os requisitos de Tratamento de Grande Volume de dados e Gerenciamento de dados são alcançados pelos *BigData Analysis*, *Query broker* e *DataVisualization* dos capitulos de *Data/Context Management* e *Applications, Services and Data Delivery* (Para maiores informações sobre os GEs vide subsessão 4.1). Por fim, o requisito de Ferramentas de desenvolvimento de Aplicações é satisfeito por uma série de ferramentas disponíveis em [32] que ofertam *plugins*, servidores de testes, *frameworks* e clientes *REST* voltados integralmente ao desenvolvimento para a FIWARE.

3.6 CityHub

A plataforma CityHub [41] surgiu da necessidade de fornecer solução para o fluxo de dados dentro da cidade de modo que tais dados sejam facilmente acessados através de uma interface para uma infraestrutura IoT e o suporte para uma metodologia de sistemas-de-sistemas, de modo que um *hub*[6] baseado em computação em nuvem possa integrar vários sub-sistemas que, coletivamente, formam a infraestrutura de software da cidade. Tal plataforma vem sendo utilizada pelas cidades de Vancouver, no Canada e Lancaster, no Reino Unido, possibilitando a criação e implantação de soluções de software na área de cidades inteligentes. O CityHub tanto gerencia o acesso a dados estáticos, como os provenientes de uma plataforma de dados abertos, quanto dados dinâmicos, como os captados em tempo real através do uso de tecnologias IoT, dados adquiridos através de *crowdsourcing*, etc.

[6] Um *Hub* atua como um portal para que usuários finais (desenvolvedores de aplicações) consigam utilizar serviços de dados com interfaces padronizadas e bem definidas.

O CityHub oferece serviços *PaaS* (plataforma com serviço) como ponto central de acesso aos dados gerados no ambiente da cidade. Cada cidade tem os seus próprios *hub*, que agregam os vários sistemas da cidade, de modo que aplicações podem consumir dados provenientes de tais sistemas. Isso é necessário uma vez que é impraticável que um único hub seja o servidor de sistemas oferecidos por diferentes organizações públicas e/ou privadas que oferecem serviços na cidade. Assim, o CityHub permite que os *hubs* sejam interoperáveis entre si, permitindo o reaproveitamento de sistemas e aplicações que funcionem no escopo de múltiplas cidades. Para isso, o CityHub oferece abstrações para endereçar a interoperabilidade entre nuvens hibridas.

O principal desafio atendido pelo CityHub é conseguir coletar, gerenciar e entregar dados de diversos tipos, desde dados estáticos, provenientes de sistemas convencionais até dados em tem real, provenientes de sistemas críticos e com altas taxas de geração e atualização de dados. Essa não é uma tarefa trivial uma vez que, além de tratar tipos distintos de dados, o CityHub deve interagir com as características distintas dos diferentes provedores daqueles dados. Tudo isso é realizado de modo que o desenvolver consegue desenvolver aplicações que consumam esses dados através de uma API uniforme e RESTful.

A Figura 9 exemplifica a arquitetura do CityHub. Em tal arqutietura, o CKAN [42] é responsável por armazenar e gerenciar o acesso aos dados abertos e estáticos da cidade. O CKAN é uma das plataformas de dados abertos utilizada por muitas cidades. Uma de suas principais características é o armazenamento de dados estáticos e temporais. Nela podem ser armazenados tanto arquivos como relatórios (em pdf, por exemplo) legíveis por humanos quanto planilhas em formatos processáveis por software, por exemplo através de arquivos CSV. Em geral, os relatórios podem ser acessados diretamente pelo portal Web provido pelo CKAN. Já os arquivos processáveis por software podem ser acessados através de APIs Web.

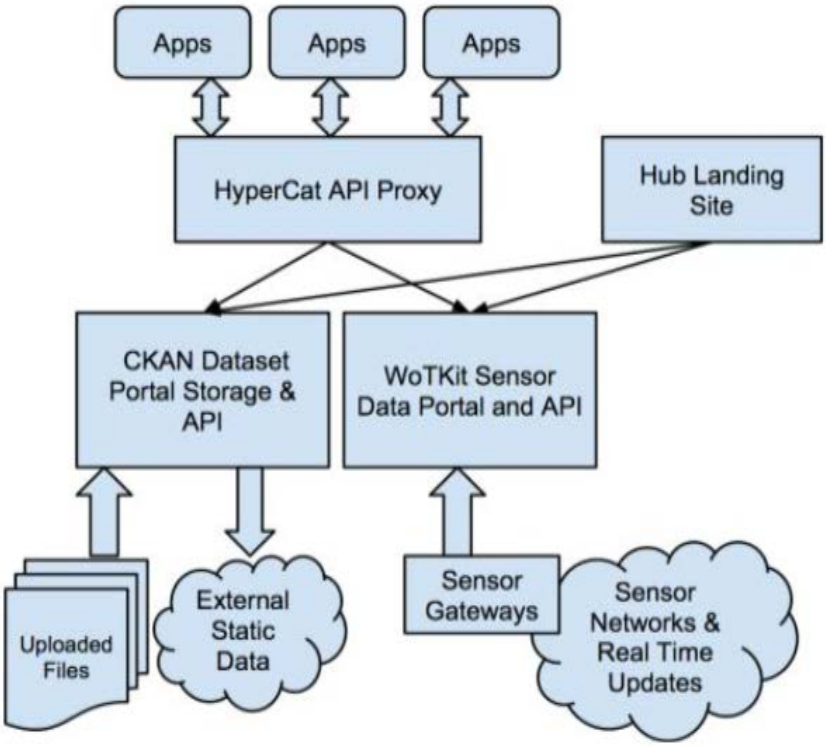

Figura 9 - Exemplo de um CityHub.

O WoTKit [43] é um kit de ferramentas IoT que permite gerenciar "coisas" e exibir os seus comportamentos em tempo real. O usuário (desenvolvedor de aplicações) pode criar abstrações de sensores baseados em hardware ou software, através de uma API ou de uma interface gráfica. A abstração de um sensor pode tanto receber dados de um sensor real quanto dar comandos a um atuador. Além disso, sensores podem sera grupados e também, associados a metadados para facilitar a sua busca. O WotKit foi projetado para oferecer às aplicações dados em tempo real e em larga escala.

A arquitetura do WoTKit é baseada em camadas, onde cada camada é responsável por realizar funções específicas, como pode ser visto na figura 10. A camada mais inferior é responsável pelo o armazenamento dos dados captados pelos sensores e pelo *message broker* (componente responsável pela troca de mensagens entre as camadas da plataforma). Na camada intermediária é responsável pelo gerenciamento dos dados e modelo. Na última camada, a do topo, é responsável pelo processamento e gerenciamento dos dados que serão mostrados na interface do usuário, acessado pelo *browser*, e pela a conectividade, por meio uma API RESTfull, com aplicações *mashups*, sensores/atuadores através de gateways.

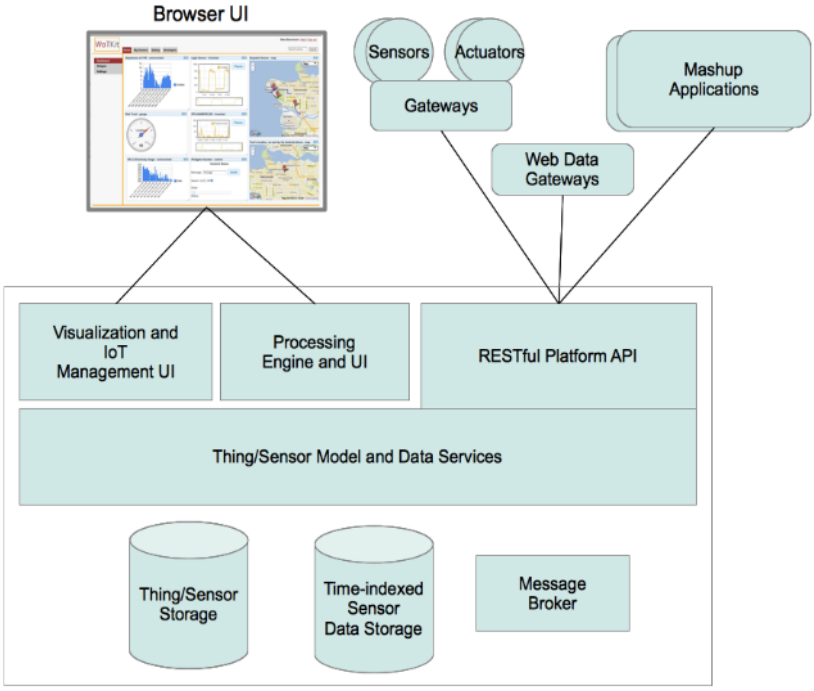

Figura 10 - Arquitetura do WoTKit

Como pode ser percebido, o CKAN e o WoTKit se complementam, de modo que o CityHub usa ambos para gerenciar todas as fontes de dados da cidade. Entretanto, um dos problemas de tal plataforma é que esses dois componentes não se conversam entre sí. Assim, é muito difícil conseguir relacionar dados estáticos e dinâmicos ao nível de plataforma. Por exemplo, considerando uma aplicação que precise consumir dados estáticos e dinâmicos sobre o transporte público, a própria aplicação terá que fazer a relação entre os dados estáticos provenientes do CKAN e os dados dinâmicos provenientes do WotKit.

O Hypercat [44] é o componente que abstrai para as aplicações o acesso à informações de qualquer natureza, independentemente se são dados estáticos, dinâmicos, provenientes de sensores baseados em hardware ou em software, etc. Tal componente especifica um catalogo hipermídia para consultas de dados e metadados. Os recursos expostos são descritos como uma lista de comandos que fornecem informações sobre o formato e o significado de cada recurso disponível para acesso.

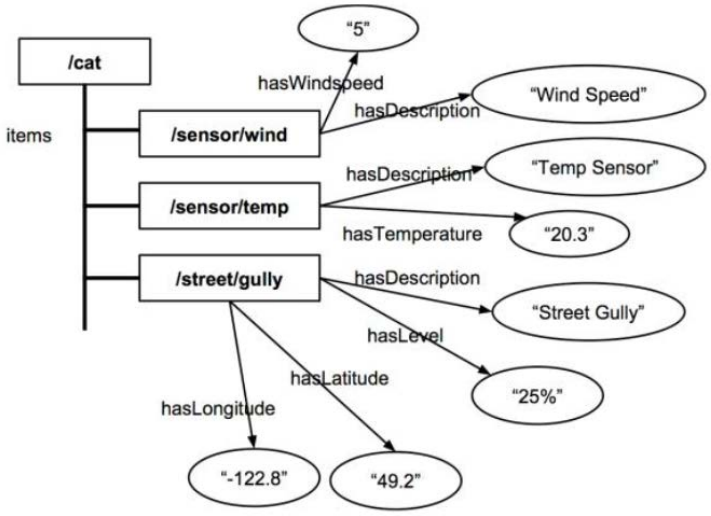

Figura 11 - Catalogo Hypercat: Lista de recursos com relacionamento e valores.

A Figura 11 ilustra a descrição dos recursos no HyperCat. Como pode ser observado, tal componente utiliza Web semântica para descrever tais recursos. Por exemplo, os tipos de recursos que são monitorados (temperatura, vento e bueiro) e as informações que estão disponíveis por cada sensor (velocidade do vento, descrição do sensor, localização do bueiro, etc.). Isso permite que os aplicativos procurem os recursos adequados e compreendam os dados recuperados. Devido à sua simplicidade, os desenvolvedores podem publicar descrições dos recursos que são oferecidos e as aplicações podem facilmente consultar as informações necessárias de maneira direta. Por meio de um Proxy API que fornece um catalogo Hypercat, é possível realizar as consultas tanto para a fontes de dados estáticos quanto fontes em tempo real fornecidos pelo *Hub* exposto como parte do CityHub. Quando uma aplicação necessita de alguma informação provida pelo CKAN ou pelo WoTKit, ela requisita tal informação ao HyperCat Proxy API. Ao receber tal solicitação, ele rastreia a origem do dado que está sendo buscado e encaminha a solicitação, seguindo a API do respectivo provedor (CKAN ou WoTKit) daquele tipo de dado e repassa a resposta de volta à aplicação.

Por fim, o *Hub Landing Site* oferece a interoperabilidade entre diversos *hubs*, permitindo que aplicações implantadas sobre um *hub* consigam consumir dados providos por fontes de dados conectadas a outros *hubs*.

Como já mencionado anteriormente, o CityHub vem sendo utilizado como base para a criação de várias aplicações voltadas para cidades inteligntes, como por exemplo, Smart Streets. No Reino Unido o Smart Streets Hub [45] está implantado sobre uma nuvem Amazon. Já em Vancouver, ele está implantado em uma nuvem acadêmica da University of British Columbia. Esses dois Hubs estão administrando dados provenientes de mais de 64.000 sensores, gerenciados pelo WoTKit, e uma grande variedade de conjuntos de dados estáticos gerenciados pelo CKAN. Esses dados são provenientes de amplo conjunto de dados abertos e privados sobre o transporte, estrada, tráfego e rodovias, que vão desde dados de tráfego em

tempo real, como definir condições climáticas como: qualidade do ar, informações meteorológicas e inundações.

3.7 Plataformas de *middleware versus* requisitos

A partir dos requisitos apresentados na Seção 2, a saber: (i) interoperabilidade; (ii) descoberta e gerenciamento de dispositivos; (iii) adaptação dinâmica; (iv) ciência de contexto; (v) escalabilidade; (vi) gerenciamento de grandes volumes de dados; (vii) e segurança. A Tabela 1 a seguir, expõe os requisitos que são atendidos por cada uma das plataformas mencionadas anteriormente. Onde, 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 1. Tabela comparativa entre plataformas à luz dos requisitos de *middleware* para cidades inteligentes.

Requisitos	EcoDiF	Kaa	SOFIA	FIWARE	CityHub
Interoperabilidade	✓	✓	✓	✓	✓
Descoberta e gerenciamento de dispositivos	○	○	✓	✓	○
Adaptação dinâmica	✘	○	✓	✓	○
Ciência de contexto	○	○	✘	✓	○
Escalabilidade	○	✓	✓	✓	✓
Tratamento de grandes volumes de dados	✘	✓	✓	✓	✓
Segurança	✓	✓	✓	✓	✘
Gerenciamento de dados	○	✓	✓	✓	✓
Ferramentas para desenvolvimento	○	✓	✓	✓	✓

4 FIWARE

Como mencionado na Seção 3.4, a plataforma FIWARE é um *middleware* projetado para desenvolvimento e provisão de aplicações relativas à internet do Futuro. Tecnicamente, a FIWARE é disponibilizada por uma composição de elementos denominados *Generic Enablers* (GEs), habilitadores genéricos em português. Os GEs são compostos por componentes que provêem um conjunto de APIs genéricas, de forma que a FIWARE pode ser vista, simplesmente, como um conjunto de APIs genéricas disponibilizadas por componentes agrupados em GEs.

As funcionalidades dos GEs são diretamente relacionadas aos domínios compositores da Internet do Futuro e têm o intuito geral de serem totalmente reutilizáveis. A organização e definição dos GEs são delimitadas por grupos de funcionalidades gerais da Internet do Futuro, os chamados Capítulos Técnicos da FIWARE. A especificação atual da FIWARE, *release* 4, define sete capítulos técnicos, a saber: *Cloud Hosting*; *Data/Context Management*; *Internet of Things (IoT) Services Enablement*; *Applications*, *Services and Data Delivery*; *Security*; *Interface to Networks and Devices (I2ND) Architecture* e *Advanced Web-based User Interface.*

A descrição dos Capítulos Técnicos, incluindo os GEs que os compõem, é apresentada na Seção 4.1. Em sequência, a Seção 4.2, apresenta aplicações de sucesso que já utilizam os GEs da FIWARE.

4.1 Capítulos Técnicos

O Capítulo ***Cloud Hosting*** é composto por GEs comprometidos com a concepção de uma moderna infraestrutura de hospedagem em computação na nuvem, isto é, uma nuvem FIWARE. A infraestrutura deve ser provida para desenvolver, implementar e gerenciar aplicativos e serviços da internet do futuro baseados na FIWARE. A provisão deve ocorrer em vários níveis de abstração de recursos, podendo atender às necessidades de diferentes aplicativos hospedados na nuvem. A especificação dos GEs do *Cloud Hosting* foi baseada na plataforma de nuvem Openstack[7]. Dessa forma, os elementos que compõem esse capítulo são na maioria componentes Openstack. Os componentes do capítulo de *Cloud Hosting*, são:

- FIWARE *IaaS* GE: deve fornecer uma camada de infraestrutura de nuvem, visando provisionamento e gerenciamento de computação, armazenamento e recursos de rede. *IaaS* GE é subdividido em vários componentes:
    - FIWARE *Cloud Compute Service*: tem como base o componente *Nova* do Openstack. Fornece a capacidade de provisionar e gerenciar máquinas virtuais (do inglês: *Virtual Machines*, VMs) e Containers Linux, bem como, recursos e

[7] http://www.openstack.org/

artefatos associados a estes. É o componente fundamental de uma nuvem FIWARE, permitindo a hospedagem de aplicações diversas - incluindo outros GEs da FIWARE e aplicações customizadas.

    - FIWARE *Cloud Image Service*: provê a capacidade de gerenciar Imagens previamente configuradas com sistemas operacionais e um conjunto de aplicações, a serem utilizadas no fornecimento de VMs ou *Containers Linux* (Elementos que geralmente contém apenas os arquivos do próprio aplicativo e suas dependências, ao invés de um sistema operacional completo). Esse GE foi baseado no componente *Glance* da Openstack.
    - FIWARE *Cloud Volume Service*: baseado no Cinder da OpenStack, é responsável pela capacidade de disponibilizar e gerenciar o armazenamento em bloco que pode ser anexado a VMs ou *Containers Linux* para manter o estado persistente nas aplicações hospedadas em nuvem FIWARE.
    - FIWARE *Cloud Network Service*: possibilita a criação e a gerencia de redes virtuais, conectando VMs e *Containers Linux* entre eles, bem como, a redes externas. Foi baseado no *Neutron* da Openstack.
    - FIWARE *Cloud Orchestration Service*: é fundamentado no Heat da OpenStack. Fornece a capacidade de orquestrar o provisionamento e gerenciamento contínuo (por exemplo, auto escalonamento na nuvem) das coleções de recursos básicos (VMs, redes, etc.), incluindo interdependências entre eles.
- FIWARE *Cloud Object Storage*: é baseado no *Swift* da OpenStack, fato que proporciona o armazenamento e recuperação de objetos "blobs", seus metadados, bem como, a execução de computações (*storlets*) em objetos, de uma forma eficiente, flexível e escalável.
- FIWARE *Cloud Application Management Service*: tem como base o *Murano* da OpenStack e fornece a capacidade de gerenciar o provisionamento e configuração de aplicações complexas, o que inclui, desde recursos básicos à configuração de software dentro das VMs ou Containers.
- FIWARE *Cloud Policy Service:* torna possível definir regras e aplicar ações em resposta a determinados eventos associados com recursos de nuvem e ao seu estado.
- FIWARE Cloud Monitoring Service: provê a capacidade de recolher e distribuir métricas de recursos associados com VMs ou nós da nuvem. Atualmente, está sendo realocado no Capítulo do FIWARE OPS, vide [seção sobre FiOPS].

O Capítulo ***Data/Context Management*** visa proporcionar uma plataforma com alto desempenho na gestão, processamento e exploração de informações de contexto. Para isso, deve fornecer GEs facilitadores do desenvolvimento e da provisão de aplicações inovadoras que requeiram gerencia de contexto, bem como, fluxos de dados em tempo real e em grande escala. Além disso, deve permitir a integração com os GEs provenientes do Capítulo *Applications, Services and Data Delivery*. No intuito de atingir essas premissas foram especificados os seguintes GEs [21]:

- *Publish/Subscribe Broker* GE: permite que as aplicações realizem o intercâmbio de eventos heterogêneos, seguindo o paradigma *Publish/Subscribe.*
- *Complex Event Processing* GE: visa permitir o processamento de fluxos de eventos em tempo real, proporcionando uma visão geral imediata e permitindo aos aplicativos responderem instantaneamente à mudança nas condições de certo cliente ou objeto (entidades, tais como dispositivos, aplicativos, sistemas, etc.).
- *BigData Analysis* GE: provê uma analise do tipo *Map-Reduce* sobre dados armazenados previamente ou dados correntes.
- *Multimedia Analysis Generation* GE: realiza a extração automática ou semiautomática de conhecimento (meta informação) com base na analise de conteúdo multimídia.
- *Unstructured data analysis* GE: permite a extração de meta-dados com base na análise de informações não estruturadas obtidas a partir de recursos da web.
- *Meta-data pre-processing* GE: facilita a geração de objetos programáveis a partir de vários formatos de meta-dados.
- *Location* GE: fornecer informações de geolocalização como uma informação de contexto obtida a partir de dispositivos.
- *Query Broker* GE: fornece um mecanismo uniforme de acesso via consultas para a recuperação de dados armazenados em formatos heterogêneos.

O Capítulo ***Internet of Things (IoT) Services Enablement*** é responsável por possibilitar que coisas (do inglês, *Things*) se tornem disponíveis, pesquisáveis, acessíveis e utilizáveis como recursos de um contexto para fomento à base de interação de aplicativos FIWARE com objetos da vida real. O conceito de coisa é definido como qualquer objeto físico, organismo vivo, pessoa ou conceito interessante do ponto de vista de um aplicativo e cujos parâmetros são total ou parcialmente ligados a sensores, atuadores ou combinações desses. Nesse caminho, os GEs desse Capítulo são organizados em dois principais tópicos: *IoT Backend* e *IoT Edge*. O *IoT Backend* (IB) compreende um conjunto de funções, recursos lógicos e serviços hospedados em uma nuvem. O IB deve interagir com Capítulo *Data/Context Managment*, para que os recursos da Internet das coisas sejam traduzidos em Entidades de contexto compatíveis com a NGSI[8]. Por outro lado, o IB deve se conectar com os elementos do *IoT Edge*. O *IoT Edge* (IE) representa todos os elementos físicos da infraestrutura da Internet das coisas necessárias à conexão dos dispositivos físicos as aplicações da FIWARE. Normalmente, o IE compreende: nós finais de internet das coisas, *gateways* da internet das coisas e redes da Internet das coisas.

Os objetivos traçados pelo Capítulo ***Internet of Things (IoT) Services Enablement*** são atingidos por um conjunto de seis GEs, a saber:

- No nível de *IoT Backend*

[8] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/NGSI-9/NGSI-10_information_model.

    - *DeviceManagement GE*: é o principal habilitador do IB, sendo responsável por conectar dispositivos físicos à plataforma FIWARE. Gerencia as entidades NGSI representativas dos elementos de internet das coisas e, por fim, permite a integração, capacidade de configuração, operação e monitoramento de nós finais, gateway e redes de internet das coisas.
    - *IoTBroker GE*: é um facilitador da comunicação com os dispositivos da Internet das Coisas. Funciona como um *middleware*, a ser executado na nuvem FIWARE, que oferta acesso rápido aos dados da internet das coisas.
    - *IoTDiscovery GE*: é responsável por garantir a disponibilidade da fonte de dados contextuais, devendo ser baseado na NGSI.
- No nível de *Iot Edge*
    - *DataHandling GE*: aborda a necessidade de filtragem, agregação e fusão de dados em tempo real a partir de diferentes fontes. Através da tecnologia de processamento de eventos complexos (CEP) é possível produzir uma série de dados agregados, suprindo as necessidades das aplicações na nuvem FIWARE.
    - *ProtocolAdapter GE*: lida com o tráfego de mensagens de entrada e saída entre os Gateways e dispositivos registrados nas redes de Internet das Coisas. Recebe vários protocolos específicos de dispositivos, traduzindo-os em uma API interna e uniforme.
    - *Gateway DeviceManagement GE*: é responsável pela comunicação com o servidor de Internet das coisas e os dispositivos que não sejam da Internet das coisas. Gerencia, também, a comunicação com os recursos da internet das coisas, isto é, os dispositivos conectados à aos *gateways* de Internet das Coisas (que pode estar *on-line* ou *off-line*), e os recursos hospedados pelo *gateway*.

O capítulo ***Applications, Services and Data Delivery*** busca apoiar a criação de um ecossistema de aplicações, serviços e dados que seja sustentável e promova a inovação. Nesse intuito, foram definidos três GES:

- *ApplicationMashup GE*: especifica uma plataforma Web que permita aos usuários, facilmente e visualmente, criar e executar os seus próprios *mashups* de aplicativos. Para isso, é baseado em um modelo de composição projetado especificamente para capacitar os usuários finais, com poucas ou nenhumas habilidades de programação, a criar e compartilhar seus próprios aplicativos web compostos de uma forma totalmente visual.
- *DataVisualization GE*: possibilita análises de documentos analíticos. O documento analítico é um objeto que especifica como os dados devem ser recuperados a partir de fontes de dados, bem como, os dados devem ser exibidos (um gráfico, uma tabela, um mapa, etc) ao usuário final. Os documentos analiticos exibem desde dados brutos à dados agregados ou parcialmente processados, provenientes de diferentes fontes de dados (RDBMS, armazenamentos de Big Data, arquivos, serviços REST, casos CKAN, etc). As visualizações devem ocorre de acordo com as necessidades dos

utilizadores e permitir a interação do usuário através de operações de filtragem, ordenação, cálculos de mercadorias conforme o tipo de análise demandar.

- *BusinessAPIEcosystem GE*: deve fornecer um conjunto padrão de APIs para permitir a gestão e monetização de diferentes tipos de ativos (serviços, aplicativos, *mashups*, visualizações, etc). Assim, envolve uma grande variedade de parceiros, durante todo o ciclo de vida do serviço desde a sua criação, sua oferta e a sua cobrança. Deve gerenciar a contabilidade, as receitas de liquidação e partilha de fundos, sendo assim, responsável pela gestão de ofertas e vendas.

O capítulo ***Security*** busca demostrar que os conceitos de segurança podem ser incorporados à realidade do design da internet do futuro. Nesse caminho, são específicados seis GEs que juntos fornecem um conjunto abrangente de serviços para aplicações, possibilitando-as cumprir com os requisitos mais importantes de segurança, tais como: autenticação, autorização e confiabilidade. O capítulo Security define, dessa forma, os Ges:

- *IdentityManagement GE*: é reponsável por alguns dos aspectos envolvidos com o acesso de usuários a: redes, serviços e aplicações. Isso inclui autenticação segura e privada de usuários para dispositivos, redes e serviços, bem como, a gerencia de perfil do usuário e gestão de autorização para domínios de serviço, inclusive Federação de Identidade para aplicações. Esse GE visa realizar a tarefa complexa de lidar com as diversas tecnologias nos domínios de segurança e fornecer tecnologias de fácil utilização, colocando o usuário final e suas necessidades diretamente no centro da arquitetura (abordagem centrada no usuário), enquanto protege sua privacidade.
- *PEPProxy GE*: é um proxy HTTP reverso, responsável por expor pontos de cumprimento de políticas (do inglês: *Policy Enforcement Point,* PEP). O PEPProxy se interpõe aos serviços REST, interceptando as requisições, ele verifica o token de acesso com o *IdentityManagement*, em seguida, procede a autorização do pedido junto ao *AuthorizationPDP*.
- *AuthorizationPDP GE*: realiza a administração das políticas de autorização em formato XACML. Quando solicitado pelo PEPProxy, baseia-se nas políticas para impor decisões de autorização ou negação aos pedidos de acesso aos serviços.
- *CyberSecurity GE*: define uma ferramenta para monitoramento, mapeamento de riscos e simulação de ataques à uma infraestrutura de nuvem. Nesse contexto destaca as funcionalidades:
    - *Cyber Data Extraction*: responsável por extrair e processar toda a entrada de dados de segurança cibernética relevantes sobre o sistema de informação: topologia da rede, as regras de firewall, os resultados da verificação de vulnerabilidade, etc.
    - *Attack Graph Engine* (MulVAL): é um analisador de vulnerabilidade topológica (do inglês: *Topological Vulnerability Analyser, TVA*), a partir da saída de *Cyber Data Extraction* gera um gráfico de ataque reagrupando todos os caminhos de ataque possíveis. O *Attack Graph Engine* agrega os dados em

conjunto com *Scored Atthack Paths* e fornece uma análise quantitativa de vulnerabilidades individuais.
    - *Scored Attack Paths*: analisa os principais caminhos de ataque tendo como base o gráfico de ataque fornecido pelo *Attack Graph Enginge* e calcula uma pontuação para cada caminho de ataque com base em seu impacto nos negócios e probabilidade de ocorrência. A pontuação representa basicamente o nível de risco.
    - *Remediation engine*: busca mitigar os riscos e tomar medidas eficazes, em conformidade com a política de segurança, processando diferentes meios para quebrar (remediar) o gráfico de ataque e estimar um custo para cada um dos riscos.
    - *Visualization interface*: deve fornecer uma interface de usuário para um operador comandar os componentes citados anteriormente e analisar dinamicamente o risco a um sistema de informação. Deve, ainda, permitir que o operador de segurança visualize os possíveis caminhos de ataque que pode enfrentar seu sistema de informação, apresentando o seu nível de atrito e as remediações que podem corrigi-los.
    - *Privacy-Preserving (Crisis) Data Sharing (P2DS)*: permite que uma organização possa compartilhar dados em relação a uma crise (ataque bem-sucedido com grande impacto nos negócios) com os outros, sem expor informações confidenciais sobre o seu sistema de informação comprometido.
- *TrustworthyFactory GE*: com o objetivo de aumentar a confiabiliade dos aplicativos disponibilizados na web, esse GE, parte de um perfil de confiabilidade e auxilia os desenvolvedores a desenvolver suas aplicações preparando-as para uma certificação de segurança. Para isso, são definidas duas características principais:
    - *Static and Dynamic Application Trustworthiness Assessment*: tem como objetivo (ainda não tão refinado) auxiliar o desenvolvedor a escrever um melhor código fonte (Java) tendo em conta alguns aspectos de design definidos em uma sequência de diversas melhores práticas. Aliado a isso, ferramentas diversas devem integrar num ambiente de desenvolvimento, podendo ser usadas separadamente ou orquestradas, com o intuito de fornecer avaliações intermediarias de confiabilidade para os desenvolvedores. Por fim, deve oferecer uma maneira de consultar o objetivo de confiabilidade dada como entrada pelo provedor de software e compará-lo com uma avaliação atual.
    - *Semi-automated Workflow for Application Trustworthiness Certification*: a especificação, também definida de forma inicial, prevê a preparação pelo avaliador da fase de certificação.
- *Privacy GE*: o objetivo desse GE é apoiar Credenciais Baseada em Atributo de Preservação da Privacidade (do inglês: *Privacy-Preserving Attribute-Based Credentials*, P2ABC). As P2ABC permitem que pessoas revelem algo sobre si (por exemplo, que elas são mais velhas do que 18 anos, ou que o nome em seu bilhete de

concerto é o mesmo que o nome no seu passaporte), mas ao mesmo tempo, revelando apenas a menor quantidade informação de identificação pessoal (neste caso a sua data real de nascimento ou de seus nomes ou números de passaporte).

O capítulo ***Interface to Networks and Devices (I2ND) Architecture*** tem um foco mais abrangente, sendo norteado por três domínios: Redes definidas por software (SDN) seja no âmbito privado, empresarial ou pública; Dispositivos robóticos e sua integração com os outros GEs da FIWARE; e, Middleware de integração avançada voltado a todos os GEs da FIWARE que necessitem de comunicação avançada. Em consequência desses objetivos, foram definidos os GEs:

- *Network flows for clouds* (Necfloc) *GE*: visa desenvolver elementos avançados para redes definidas por Software (do inglês: *Software Defined Networking,* SDN), em redes na nuvem ou em centro de dados. Além disso, deve melhorar a capacidade de gerir de forma eficiente os recursos em infraestruturas complexas, como as arquiteturas de centros de dados de rede. Deverá, ainda, fornecer a capacidade de gerenciar os problemas de configuração de rede devido ao grande número de máquinas virtuais, redes virtuais e pilhas de rede.
- *Network Information and Control* (Netic) *GE*: destina-se a proporcionar o acesso de modo abstrato aos heterogêneos dispositivos de rede abertos. Assim, a informação do estado da rede pode ser exposta, permitindo certo nível de capacidade de programação dentro da rede (dependendo do tipo de rede e a interface de controle aplicável). Esta programação também deverá permitir a virtualização de rede, ou seja, a captação de recursos de rede física, bem como, o seu controle por um fornecedor de rede virtual.
- *Robotics GE:* utiliza o Framework ROS[9] (*Robotics Operating System*) para gerenciar uma variedade de diferentes robôs. As funções do ROS podem ser alocadas dinamicamente, quer no próprio robô ou no lado da plataforma (por exemplo, em uma máquina virtual) para executar tarefas que exijam maior processamento. Esse GE deve fornecer interfaces dedicadas a desenvolvedores e para interagir com outros GEs FIWARE, assim como, principalmente uma interface para OpenStack (GE de Cloud Hosting) para provisão de máquinas virtuais que atuarão como clones de robô para implementar os recursos de computação adicionais para robôs.
- *Advanced Middleware GE:* consiste de conjunto de ferramentas de compilação, execução e uma biblioteca de comunicação a ser integrado a uma aplicação. O *Advanced Middleware GE* permite a comunicação flexível, eficiente, escalável e segura entre aplicações distribuídas e entre outros GEs da FIWARE. Esse GE deve oferecer uma funcionalidade para encontrar e estabelecer uma comunicação a um serviço, negociar os melhores protocolos de ligação e de transporte, acessar as estruturas de dados de aplicativos e codificar os dados necessários em um formato

[9] http://www.ros.org/

adequado para o protocolo escolhido, e, finalmente, enviar esses dados e, eventualmente, receber resultados em troca.

O capítulo ***Advanced Web-based User Interface*** busca oferecer um conjunto abrangente de serviços para as aplicações para implementarem interfaces do usuário 2D e 3D altamente interativas. A web tem se tornado a principal plataforma das aplicações que utilizam interfaces gráficas ao usuário, por isso, esse GE tem, também, como intuito alavancar esse desenvolvimento e estender HTML com uma série de novos recursos que podem ser usados para melhorar significativamente as capacidades de interface do usuário e melhorar a experiência do usuário. Os componentes da interface Web avançados devem ser inteiramente baseados em HTML e outras tecnologias padrões na internet, tais como, as implementadas por qualquer navegador moderno. Os GEs definidos por esse capítulo são:

- *3D-UI GE*: busca adicionar novos elementos HTML (usando uma implementação Polyfill[10]) para descrever cenas em 3D, incluindo geometria, material, texturas, luzes e câmeras. Define, ainda, um novo mecanismo de fluxo de dados funcional que é usado para ativar animações interativas, processamento de imagem, realidade aumentada, e outros elementos dinâmicos para uma cena.
- *Synchronization GE*: permite que os vários serviços dos GEs desse capítulo, que estejam executando em diferentes clientes, possam realizar sincronização em tempo real. Enquanto isso é obrigatório para a prestação de ambientes 3D compartilhados (por exemplo, para os jogos) também pode ser aplicado para a funcionalidade 2D. Recomenda como um elemento chave o uso de um design altamente flexível e altamente escalável da arquitetura de servidor de sincronização que permitirá que o rearranjo dinâmico do ambiente 3D.
- *GISDataProvider GE:* oferecer acesso a dados 3D do sistema de informação geográfico (GIS) por meio de consultas de localização geográfica que podem ser usadas por qualquer aplicação para processar o conteúdo em um cenário virtual do mundo real.
- *POIDataProvider GE*: permitir o acesso avançado a dados de Pontos de Interesse (POI) que podem ser usados para posicionar o conteúdo 2D ou 3D no contexto de uma cena 3D. Além de fornecer os metadados habituais à especificação de dados POI e possibilitar a definição de elementos de cena 3D (dinâmico) em relação ao POI, incluindo marcadores de realidade aumentada.
- *InterfaceDesigner GE*: busca fornecer uma composição interativa de ferramentas de edição web que permita aos usuários manipular interativamente mundo 3D no mesmo ambiente do navegador que também é usado para executar o aplicativo.
- *AugmentedReality GE*: tendo como base o 3D-UI GE, propicia o processamento acelerado por hardware de imagens e interfaces gráficas computacionais, ponto necessário para a realidade aumentada.

[10] https://en.wikipedia.org/wiki/Polyfill

4.2 Casos de Uso

Diferentes projetos baseados na plataforma FIWARE têm sido desenvolvidos no âmbito das Cidades Inteligentes para oferecer novos produtos e serviços inovadores aos seus habitantes[11]. Esta subseção descreve alguns exemplos desses projetos, indicando os seus benefícios a partir do uso de GEs do FIWARE.

4.2.1 FI-Guardian

O projeto FI-Guardian foi concebido a partir das conclusões de pesquisas recentes que indicam a vulnerabilidade de pessoas diante de eventos, como desastres naturais provocados pelas mudanças climáticas observadas nos últimos anos, além de acidentes, como os químicos ou radioativos. Em linhas gerais, o FI-Guardian tem o propósito de reduzir o impacto de eventos considerados perigosos para as vidas das pessoas a partir do uso de ferramentas baseadas em tecnologia. Os autores do FI-Guardian destacam que as aplicações podem ser aproveitadas pelos habitantes e também pela administração das cidades para reagir de forma rápida e eficaz diante de tais eventos. O FI-Guardian é um projeto reconhecido no contexto das Cidades Inteligentes, tendo sido o vencedor do "Desafio FIWARE" realizado no evento *Campus Party*.

O FI-Guardian representa um monitor inteligente que possui os seguintes objetivos específicos:

1. aprimorar o monitoramento de eventos considerados adversos para a vida das pessoas;
2. ampliar os canais de comunicação;
3. melhorar a eficiência de sistemas de alerta;
4. favorecer a interoperabilidade.

Para atingir esses objetivos, o FI-Guardian é baseado na plataforma FIWARE, que permite o desenvolvimento de novas aplicações mais rapidamente, sabendo que muitas tecnologias necessárias já estão disponíveis como GEs. Além disso, o FIWARE Lab provê um ambiente de computação em nuvem que favorece o acesso fácil, integrado e acessível dos GEs por parte dos desenvolvedores das aplicações. Os principais GEs usados pelo FI-Guardian são os seguintes:

- *Publish/Subscribe Orion Context Broker*. No âmbito do FI-Guardian, o GE *Orion Context Broker* é responsável pelo gerenciamento de todas as informações de contexto e de dados coletados por diferentes grupos de unidades de sensoriamento, além de facilitar os processos de acesso e de consulta a esses dados.
- *BigData Analysis-Cosmos*. O componente é usado para gerenciar uma quantidade massiva de informações. Esse GE mantém um histórico de valores conectados a um contexto geográfico que são usados para a análise de tendências e a geração de gráficos representativos.

[11] https://www.fiware.org/success_stories?s=smart-cities

- *Complex Event Processing* (CEP) – *Proactive Technology Online*. Esse GE permite a criação de uma rede inteligente para o processamento de dados que realiza a análise de informações em tempo real e define os parâmetros para o acionamento de alarmes em caso de situações críticas predefinidas.
- *Stream-oriented – Kurento*. Esse GE é usado para a transmissão e manipulação de conteúdo multimídia, como áudio e vídeo. Com o Kurento, o FI-Guardian é capaz de implementar um sistema inovativo de alarme multimídia baseado no componente webRTC para enviar mensagens de difusão (*broadcast*) georeferenciadas para qualquer dispositivo conectado.
- *Application Mashup – Wirecloud*. Esse compoonente do FIWARE suporta o desenvolvimento de cartas, relatórios, mapas georeferenciados de sensores, imagens de câmeras *online*, dentre outros elementos.
- *Identity Management – KeyRock*. Com esse GE, é possível autenticar usuários e garantir o acesso seguro às contas cadastradas no FIWARE Lab.

4.2.2 SmartAppCity

O projeto *SmartAppCity* é uma aplicação para *smartphones* voltada para o escopo administrativo das cidades. A aplicação abrange os serviços públicos da cidade e inclui o setor comercial, agregando valor e melhorando a qualidade de vida tanto dos moradores quanto dos turistas. A partir de um modelo de Parceria Público-Privada (PPP), os administradores poderão disponibilizar os dados da cidade de forma aberta e em tempo real usando a plataforma FIWARE, permitindo que as pessoas usem esses dados para oferecer novos produtos e serviços eficientes à população.

Com o SmartAppCity, um conjunto de funcionalidades podem ser disposnibilizadas aos cidadãos. Ele pode, por exemplo, melhorar o sistema de gerenciamento de tráfego de veículos e torná-lo mais eficiente para as pessoas. Outros exemplos de aplicações baseadas no SmartAppCity incluem informações sobre chegadas e partidas de ônibus, metrô ou aviões, monitoramento de espaços públicos usando câmeras, sistemas de alerta, notícias sobre eventos nas cidades, localização e horário de farmácias, valores de combustível em postos, informações turísticas, zonas de Wi-Fi abertas, e etc.

Diferentes GEs do FIWARE foram integrados ao projeto SmartAppCity para suportar as suas funcionalidades. Os GEs do FIWARE usados no projeto foram os seguintes:

- *BigData Analysis – Cosmos*. O projeto SmartAppCity usa esse GE para monitorar, analisar e controlar todo o Big Data gerado pela massiva quantidade de dados gerados na cidade. Ele permite o carregamento (*upload*) de arquivos com os dados, que podem ser acessados e manipulados pelas aplicações específicas.
- *Identity management – KeyRock*. Com esse GE, o SmartAppCity pode autenticar seus usuários com maior segurança. Além disso, o SmartAppCity é capaz de prover um acesso personalizado dos dados aos cidadãos.

- *Publish/Subscribe Context Broker - Orion Context Broker*. Esse GE permite que o SmartAppCity gerencie e publique informações de contexto sobre as cidades para o uso das aplicações.

4.2.3 SPERO

O projeto SPERO tem o objetivo de desenvolver e comercializar um conjunto de produtos inovadores e baseados em Web para organizações públicas ou privadas para o gerenciamento e a manutenção de facilidade e redes em espaços urbanos. O SPERO permite aos usuários de dispositivos móveis reportarem ocorrências (i.e., anomalias, ocorrências e etc.) georreferenciadas que podem requerer algum reparo ou intervenção. Além disso, o SPERO pode interagir com redes sociais e, com isso, os reportes dos usuários podem ser seguidos, comentados ou verificados para a tomada de decisões considerando aspectos sociais.

Com o uso do FIWARE, espera-se que o projeto SPERO alcance uma (virtual) escalabilidade ilimitada e uma aplicação em escala global. Além disso, um grande volume de dados (500 mil reportes por dia em 2020) deve ser gerado com a aplicação do projeto SPERO. O FIWARE permitirá o gerenciamento desse volume de dados com rapidez e confiabilidade, que são aspectos requeridos para os sistemas IoT.

As funcionalidades do SPERO devem ser suportadas considerando diferentes GEs do FIWARE, descritos a seguir:

- *Big Data GE (Cosmos)*. O Cosmos é considerado uma ferramenta apropriada para o processamento paralelo de dados que permite a categorização e a execução de algoritmos de tratamento de dados no SPERO. Com base nesse tratamento de dados, é possível gerar alertas e relatórios necessários para a tomada de decisões administrativas no escopo das cidades.
- *Object Storage GE*. Esse GE é usado para o armazenamento de um grande volume de dados binários referentes a documentos, como imagens, arquivos e etc.).
- *POI Data Provider*. O SPERO usa esse GE para o armazenamento específico de dados georreferenciados.
- *Context Broker GE (Orion).* O Orion Context Broker permite ao SPERO acessar dados e receber notificações de alterações de dados usados por aplicações de gerenciamento de serviços (e.g., tráfego de veículos, monitoramento de espaços públicos, etc.).

5 Considerações Finais

Este relatório apresentou um conjunto de requisitos de plataformas de *middleware* para o desenvolvimento de aplicações de cidades inteligentes, bem como algumas plataformas existentes para esse domínio. Além disso, analisou as plataformas apresentadas à luz dos requisitos elencados. A única plataforma que atendeu a todos os requisitos, a plataforma FIWARE, foi descrita com mais detalhes e citados alguns casos de sucesso de aplicações desenvolvidas usando tal plataforma. De fato, a plataforma FIWARE tem sido amplamente usada na Europa onde há uma vasta gama de aplicações de cidades inteligentes desenvolvidas e em operação usando a plataforma. Isso justifica-se pelo fato do FIWARE ser uma plataforma aberta, livre de royalties, que provê um conjunto de APIs que facilitam o desenvolvimento de aplicações para cidades inteligentes e disponibiliza implementações de referência para cada API. A plataforma FIWARE fornece uma biblioteca de componentes (*generic enablers*), descritos de forma resumida nesse relatório, com respectivas implementações de referência que permite desenvolvedores usar funcionalidades diversas como a conexão com dispositivos, análise de dados, etc.

Como resultado da fase atual do projeto, além desse relatório, foi instalada uma instância da plataforma na UFRN. O objetivo da próxima fase é usar a plataforma FIWARE e serviços associados como substrato para o desenvolvimento de aplicações de cidades inteligentes. O próximo relatório irá conter a descrição dos serviços FIWARE e como usá-los.

Referências

[1] Milind Naphade, Guruduth Banavar, Colin Harrison, Jurij Paraszczak, Robert Morris (2011) Smarter cities and their innovation challenges. Computer, vol. 44, no. 6, pp. 32-39.

[2] Thiago Teixeira, Sara Hachem, Valérie Issarny, Nikolaos Georgantas (2011) Service oriented middleware for the Internet of Things: A perspective. In: Witold Abramowicz, Ignacio M. Llorente, Mike Surridge, Andrea Zisman, Julien Vayssière (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.

[3] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti, Subhajit Dutta (2011) Role of middleware for internet of things: A study. International Journal of Computer Science & Engineering Survey, vol. 2, no. 3, pp. 94-105.

[4] Philip A. Bernestein (1996) Middleware: a model for distributed system services. Communications of the ACM, vol. 39, no. 2, 86-98.

[5] Paulo F. Pires, Flavia C. Delicato, Thais Batista, Thomaz Barros, Everton Cavalcante, Marcelo Pitanga. Plataformas para a Internet das Coisas. In: Magnos Martinello, Moisés Renato Nunes Robeiro, Antônio Augusto de Aragão Rocha (org.) XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – Minicursos. Porto Alegre, RS, Brasil: SBC, 2015, pp. 110-169.

[6] Luigi Atzori, Antonio Iera, Giacomo Morabito (2010) The Internet of Things: A survey. Computer Networks, vol. 54, no. 15, pp. 2787-2805.

[7] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, Imrich Chlamtac (2012) Internet of Things: Vision, applications and research challenges. Ad Hoc Networks, vol. 10, no. 7, pp. 1497-1516.

[8] Eleonora Borgia (2014) The Internet of Things vision: Key features, applications, and open issues. Computer Communications, vol. 54, pp. 1-31.

[9] Flavia C. Delicato, Paulo F. Pires, Thais Batista (2013) Middleware solutions for the Internet of Things. United Kingdom, Springer London.

[10] Mark W. Maier (1998) Architecting principles for systems-of-systems. Systems Engineering, vol. 1, no. 4, pp. 267–284.

[11] Porfirio Gomes, Everton Cavalcante, Pedro Maia, Thais Batista, Kamilla Oliveira (2015) A systematic mapping on discovery and composition mechanisms for systems-of-systems. Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications. Washington, DC, USA: IEEE Computer Society, pp. 191-198.

[12] Flavia C. Delicato, Paulo F. Pires, Thais Batista, Everton Cavalcante, Bruno Costa, Thomaz Barros (2013) Towards an IoT ecosystem. Proceedings of the First International Workshop on Software Engineering for Systems of Systems. New Yorj, NY, USA, ACM, pp. 25-28.

[13] Peyman Oreizy, Nenad Medvidovic, Richard N. Taylor (1998) Architecture-based runtime software evolution. Proceedings of the 20th International Conference on Software Engineering. Washington, DC, USA, IEEE Computer Society, pp. 177-186.

[14] Philip K. McKinley, Seyed Massoud Sadjadi, Eric P. Kasten, Betty H. C. Cheng (2004) Computer, vol. 37, no. 7, pp. 56-64.

[15] Anind K. Dey (2001) Understanding an using context. Personal and Ubiquitous Computing, vol. 5, no. 1, pp. 4-7.

[16] Charith Perera, Arkady Zaslavsky, Peter Christien, Dimitrios Georgakopoulos (2014) Context aware computing for the Internet of Things: A survey. IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 414-454.

[17] Peter M. Mell, Timothy Grance (2011) The NIST Definition of Cloud Computing. Technical report, National Institute of Standards & Technology, Gaithersburg, MA, USA.

[18] Fabio Bonomi, Rodolfo Milito, Jiang Zhu, Sateesh Addepalli (2012) Fog Computing and its role in the Internet of Things. Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing. New York, NY, USA, ACM, pp. 13-16.

[19] Luis M. Vaquero, L. Rodero-Merino (2014) Finding your way in the fog: Towards a comprehensive definition of Fog Computing. ACM SIGCOMM Computer Communication Review, vol. 44, no. 5, pp. 27-32.

[20] David Tracey, Cormac Sreenan (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. Washington, DC, USA, IEEE, pp. 546-553.

[21] Min Chen, Shiwen Mao, Yunhao Liu (2014) Big Data: A survey. Mobile Networks and Applications, vol. 19, no. 2, pp. 171-209.

[22] Paulo F. Pires, Everton Cavalcante, Thomaz Barros, Flavia C. Delicato, Thais Batista, Bruno Costa (2014) A platform for integrating physical devices in the Internet of Things. Proceedings of the 12th IEEE International Conference on Embedded and Ubiquitous Computing. Washington, DC, USA, IEEE, pp. 234-241.

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

[24] Cesare Pautasso, Olaf Zimmermann, Frank Leymann. (2008) RESTful Web services vs. “big” Web services: Making the right architectural decision. Proceedings of the 17th

International Conference on the World Wide Web. New York, NY, USA, ACM, pp. 805-814.

[25] Extended Environments Markup Language: http://www.eeml.org/

[26] Dominique Guinard, Vlad (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.

[27] Enterprise Mashup Markup Language: https://en.wikipedia.org/wiki/Enterprise_Mashup_Markup_Language

[28] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, M. B. Srivastava (2006) Participatory Sensing. Proceedings of the 4th ACM International Workshop on World-Sensor-Web.

[29] Kaa Open-Source IoT Platform: http://www.kaaproject.org/

[30] Kaa IoT Platform Home: http://docs.kaaproject.org/display/KAA/

[31] SOFIA2: http://sofia2.com/home_en.html

[32] SOFIA2 for Smart Tourism: Case study with Rias Baixas Tourism, Coruña Smart City and DepoGAP (Asset Management of Province of Pontevedra): http://www.cpse-labs.eu/sp_smart_tourism.php

[33] SOFIA2 for Smart Health and digital home welfare: Galician Health Services Case Study: http://www.cpse-labs.eu/sp_smart_health.php

[34] SOFIA2 for port logistics service marketplace: Case study with Valparaíso, Chile: http://www.cpse-labs.eu/sp_port_logistics.php

[35] SOFIA2 Demonstrators: http://sofia2.com/demostradores_en.html

[36] Coruña Smart City: http://www.smart-circle.org/portfolios/coruna-smart-city-2/

[37] Pantsar-Syväniemi, Susanna, Jarkko Kuusijärvi, Eila Ovaska (2011) Context-awareness micro-architecture for smart spaces. In: Jukka Riekki, Mika Ylianttila, Minyu Guo (eds.) Proceedings of the 6th International Conference on Advances in Grid and Pervasive Computing. Lecture Notes in Computer Science, vol. 6646. Germany, Springer Berlin Heidelberg, pp. 148-157.

[38] S. Shenker (2006) Fundamental design issues for Future Internet. IEEE Journal on Selected Areas in Communications, vo. 13, no. 7, pp. 1176-1188.

[39] Jianli Pan, Subharthi Paul, Raj Jain (2011) A survey of the research on Future Internet architectures. Communications Magazine, vol. 49, no. 7, pp. 26-36.

[40] FIWARE: https://www.fiware.org/

[41] Rodger Lea, Michael Blackstock (2014) CityHub: A cloud-based IoT platform for smart cities. Proceedings of the 6th IEEE International Conference on Cloud Computing Tecnology and Science. USA, IEEE, pp. 799-804.

[42] Joss Winn (2013) Open data and the academy: An evaluation of CKAN for research data management. Proceedings of the 2013 IASSIST Annual Conference.

[43] Michael Blackstock, Rodger Lea (2012) IoT mashups with the WoTKit. Proceedings of the 3rd International Conference on the Internet of Things. USA, IEEE, pp. 159-166.

[44] Hypercat IoT Ecosystem Demonstrator Interoperability Action Plan: http://eprints.lancs.ac.uk/69124/1/Interoperability_Action_Plan_1v1_spec_only.pdf

[45] Smart Streets Project: http://www.smartstreetshub.com/