Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Felipe Lauermann Vielitz

CMFRAME: Framework para o Gerenciamento de Históricos de Contextos Dinâmicos e Hierárquicos

São Leopoldo, 2018

Felipe Lauermann Vielitz

CMFRAME: FRAMEWORK PARA O GERENCIAMENTO DE HISTÓRICOS DE CONTEXTOS DINÂMICOS E HIERÁRQUICOS

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre pelo Programa de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos — UNISINOS

Orientador:
Prof. Dr. Jorge Luis Victória Barbosa

SÃO LEOPOLDO

2018

V659c Vielitz, Felipe Lauermann.
CMFrame : framework para o gerenciamento de históricos de contextos dinâmicos e hierárquicos / Felipe Lauermann Vielitz. – 2018.
116 f. : il. ; 30 cm.

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, 2018.
"Orientador: Prof. Dr. Jorge Luis Victória Barbosa."

1. Framework. 2. Contexto. 3. Históricos de contextos. I. Título.

CDU 004

Dados Internacionais de Catalogação na Publicação (CIP)
(Bibliotecário: Flávio Nunes – CRB 10/1298)

Felipe Lauermann Vielitz

CMFRAME: Framework para o Gerenciamento de Históricos de Contextos Dinâmicos e Hierárquicos

Dissertação apresentada à Universidade do Vale do Rio dos Sinos – Unisinos, como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

Aprovado em 23 de março de 2018

BANCA EXAMINADORA

Prof. Dr. Jorge Luis Victória Barbosa – UNISINOS

Prof. Dr. Leandro Krug Wives – UFRGS

Prof. Dr. Kleinner Silva Farias de Oliveira – UNISINOS

Prof. Dr. Jorge Luis Victória Barbosa (Orientador)

1. Visto e permitida a impressão
São Leopoldo,

Prof. Dr. Sandro José Rigo
Coordenador PPG em Computação Aplicada

RESUMO

Com o crescimento da disponibilidade de dispositivos capazes de captar informações em seu entorno e da expansão da conectividade em dispositivos móveis, soluções relacionadas à Internet das coisas (IoT*)* estão passando a integrar cada vez mais aspectos da sociedade. Para que soluções de IoT possam emergir no mercado com sucesso, se torna necessário mais do que o tradicional uso da computação móvel dos *smartphones* e portáteis, passando a utilizar os objetos do dia-a-dia de uma forma interligada e adicionando fatores de inteligência aos ambientes. A busca pela criação de ambientes capazes de proatividade irá refinar as experiências da população tanto em questões vinculadas ao trabalho quanto ao lazer, propiciando objetos com características inteligentes que se adaptam a situação atual sem a necessidade de intervenção humana. De forma a prover essa adaptação ou resposta as situações em seu entorno, torna-se necessário que os ambientes estejam cientes de sua circunstância em dado momento, para tanto, propõe-se o desenvolvimento do CMFrame, um *framework* para o gerenciamento de informações de contexto captadas em um ambiente físico, na forma de histórico, através do uso de entidades hierárquicas e dinâmicas. A partir desse conceito, as entidades podem modificar sua organização hierárquica para assim redefinir a quem um determinado contexto está atrelado naquele instante. Os contextos atrelados a cada entidade também são dinâmicos, e podem armazenar diferentes quantidades de valores a qualquer instante, variando indefinidamente conforme a necessidade. O *framework* foi avaliado através do desenvolvimento de duas aplicações, em que foram averiguadas as características oferecidas pelo CMFrame quanto ao uso de suas entidades hierárquicas e dinâmicas. Os resultados dessa avaliação mostram que o *framework* atende as necessidades como um gerenciador de históricos de contextos, para situações que utilizem ambientes físicos com elementos inteligentes e aplicações que fazem a gerência do ambiente através da atuação sobre os mesmos.

Palavras-chave: framework, contexto, históricos de contextos.

ABSTRACT

With the increasing availability of devices capable of capturing information about their surroundings and the expansion of connectivity in mobile devices, Internet of Things solutions have come to integrate more and more aspects of society. In order for IoT solutions to emerge successfully in the market, it becomes necessary to employ more than the traditional mobile computing from smartphones and portables, instead making use of day-to-day objects in an interconnected way and adding intelligence to environments. The search for environments capable of proactivity will empower the experiences of the population both in matters related to work as those of leisure, providing objects with intelligent characteristics that adapt to the current situation without the need for human intervention With the intention of providing this adaptation or response to their surroundings, it becomes necessary that the environments be aware of their circumstance at any given moment, for this purpose, the development of CMFrame is proposed, a *framework* for managing context information captured in a physical environment, in the form of a history, through the use of hierarchical and dynamic entities. From this concept, entities can modify their hierarchical organization to redefine to whom a certain context is linked at that moment. The contexts attached to each entity are also dynamic and can store different amounts of values at any given time, varying indefinitely as the need arises. The *framework* was evaluated through the development of two applications, in which the characteristics offered by CMFrame regarding the use of their hierarchical and dynamic entities were investigated. The results of this evaluation show that the *framework* serves the needs as a context history manager, for situations that use physical environments with intelligent elements and applications that manage the environment by acting on them.

KEYWORDS: framework, context, contexts histories.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACM	*Association for Computing Machinery*
BSON	*Binary JSON*
CPS	*Cyber-Physical Systems*
GPS	*Global Positioning System*
IEEE	*Institute of Electrical and Electronics Engineers*
IoT	*Internet of Things*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
M2M	*Machine-to-Machine*
MQTT	*Message Queue Telemetry Transport*
MVC	*Model-View-Controller*
NFC	*Near Field Communication*
RFID	*Radio-Frequency Identification*
WSN	*Wireless sensor networks*

SUMÁRIO

1. INTRODUÇÃO

O crescimento tecnológico da sociedade traz consigo a tendência de reduzir o tamanho de componentes eletrônicos, tornando-os mais eficientes e confiáveis, com capacidades de processamento cada vez maiores. Atrelado a este fato tem ocorrido uma redução de custos de produção pelo desenvolvimento da tecnologia e utilização de técnicas mais eficientes em sua produção. Com a constante expansão de redes *wireless,* presenciada recentemente e sua integração com outros tipos de tecnologias sem fio, se tornará possível prover o suporte a aplicações de computação móvel e obter um desempenho tão eficiente quanto nas redes cabeadas (NICOPOLITIDIS et al, 2003). Essa expansão complementa a visão de Weiser (1991), que define a necessidade de uma rede que una os distintos dispositivos como um dos fatores chave para a computação ubíqua, em que dispositivos passarão a ser utilizados inconscientemente pelos seus usuários em diversos ambientes a fim de auxiliar em tarefas cotidianas.

Em 2001, Satyanarayanan expandiu a visão de Weiser ao definir que a computação ubíqua, ou computação pervasiva, seria uma evolução dos campos de sistemas distribuídos e computação móvel, adicionando quatro pontos de pesquisa aos conceitos de mobilidade por considerar uma parte integral da vida cotidiana, sendo eles: a melhor utilização de ambientes inteligentes, invisibilidade, escalabilidade localizada e ocultação do condicionamento desigual. Satyanarayanan (2001) define que ambientes inteligentes apresentam a fusão de dois mundos, o físico e o virtual, permitindo a captação de dados de sensores e o controle de um mundo a partir de outro.

O controle de um mundo a partir do outro nos apresenta a possibilidade de criar ambientes capazes de proatividade, ao abstrair certas tarefas que podem ocorrer em determinado ambiente, através do processamento de informações coletadas por sensores ao longo do tempo, e assim inferir sobre quais ações seriam as mais apropriadas para aquele momento específico. Diante disso, se torna apropriado criar mecanismos que visem facilitar a interação de aplicações que atuem sobre um determinado ambiente sem a necessidade de intervenção humana.

Ambientes proativos requerem uma análise robusta acerca das condições que os cercam, tornando necessário que exista uma coleção de informações de contexto e ações do usuário. Quando essas informações dizem respeito à situações do passado, passam a ser denominadas históricos de contextos (HONG et al, 2009). A fim de facilitar a integração dos elementos que compõem um ambiente físico com os dados de históricos, foi aplicada a ideia de entidades reais e virtuais, em que alguns elementos de um ambiente são representados através de uma entidade real, ou física, enquanto que outros por uma entidade completamente virtual. A entidade virtual jamais existiria fisicamente no ambiente, porém ainda é capaz de afetá-lo. Atrelado a essas características, as entidades reais e virtuais ainda farão parte de uma hierarquia que tem por objetivo definir a relação que cada entidade possui com as demais dentro de um sistema ou aplicativo que gerencie o ambiente. Com isso as entidades podem modificar sua hierarquia durante o ciclo de vida do sistema, de forma dinâmica, e assim melhor representar um ambiente mutável, com entidades móveis que em determinados períodos de tempo podem migrar para outros ambientes. A esse conceito é atribuído o nome de entidades hierárquicas e dinâmicas.

1.1 Motivação

Considerando a importância de reduzir a carga de tarefas atrelada aos indivíduos que frequentam um ambiente no dia-a-dia, bem como a adição de características que venham a melhorar a qualidade de vida desses usuários, surge a busca pela automação de tarefas diversas em ambientes físicos, fazendo com que os mesmos apresentem características proativas que visem torná-los mais inteligentes de um ponto de vista humano.

Nesse contexto, ambientes inteligentes, também chamados de inteligência ambiente, podem ser descritos como sensores e atuadores embutidos dentro de uma sala ou ambiente, reagindo de forma automática aos usuários. Nesses casos, os sensores mesclam-se ao ambiente, tornando-se escondidos da percepção do usuário e eliminando a necessidade de se interagir explicitamente com eles (TORUNSKI et al, 2012). Outra definição atribui ambientes inteligentes como sendo um mundo físico interconectado através de uma rede contínua, vasta e invisível, com sensores,

atuadores e unidades computacionais embutidas de forma transparente nos objetos do dia-a-dia (COOK; DAS, 2004).

Quando se aborda o tópico de sensores, surge a expressão Internet das Coisas (IoT*,* do inglês, *Internet of Things*), que se refere a objetos, ou coisas, unicamente identificados, que são capazes de interagir entre si e cooperar com objetos próximos para alcançar objetivos comuns (ATZORI; IERA; MORABITO, 2010). IoT apresenta características e componentes comuns a outros sistemas, como sistemas máquina a máquina (M2M) (CHEN; WAN; LI, 2012) e redes de sensores sem fio (WSN) (YICK; MUKHERJEE; GHOSAL, 2008) e sistemas cibernéticos-físicos (CPS) (GEISBERGER; BROY, 2015), já que estes fornecem a base para seu funcionamento. Deve-se atentar ao fato de que o conceito inicial de internet das coisas não leva em consideração questões referentes à inteligência ambiente e o controle autônomo, como visto em outros sistemas que adotam conceitos de IoT (CHEN; WAN; LI, 2012).

A utilização de sensores para a captura de dados do mundo físico e a interação com atuadores para afetar o mundo real tornam necessário o uso de informações vindas do ambiente ao qual se deseja afetar; essa informação por sua vez torna-se contexto (DEY, 2001). Caso seja relevante a uma determinada aplicação ter conhecimento sobre os contextos previamente capturados pelo sistema, torna-se mandatório manter um registro contendo uma coleção de contextos para aquele ambiente, na forma de um histórico. Esses contextos registrados podem ser utilizados posteriormente por uma aplicação que, através do uso de inferência, determinará qual o melhor curso de ação a ser tomado em um ambiente, em dado momento.

Dependendo da quantidade de ambientes inteligentes controlados por uma ou mais aplicações, pode-se ter uma vasta quantidade de objetos conectados, atrelando esse fator à cadência com que os dados são coletados, bem como sua variedade, torna-se possível alcançar o que é considerado Big Data. Big Data é comumente classificada como uma coleção de dados muito vasta, contendo uma grande diversidade de tipos de dados, a ponto de dificultar seu processamento utilizando-se de técnicas de processamento de dados tanto tradicionais quanto de última geração (CHEN; ZHANG, 2014). A existência desse grande volume de dados disponível para consulta permite que os módulos responsáveis por realizar a inferência sobre um determinado ambiente sejam refinados, garantindo a maior confiabilidade de uma determinada aplicação e sua chance de atingir o resultado esperado. Sendo assim,

torna-se interessante utilizar mecanismos de gerenciamento de históricos de contextos para incrementar as operações em ambientes inteligentes. Atrelado a isso, pode-se citar a grande quantidade de configurações distintas para os dispositivos que se encontram conectados a rede, bem como os diferentes dados passíveis de transmissão em dado momento. Esses fatores permitem que ambientes inteligentes possam modificar os elementos internos que os compõem, conforme a demanda, a fim de suprir uma nova necessidade. Esse dinamismo tanto de estrutura quanto dos tipos de dados necessita de uma ferramenta que seja capaz de lidar com tais variações, provendo funcionalidades compatíveis com esse modo de operação. Para tanto, foi desenvolvido um *framework* para o gerenciamento de históricos de contextos através do uso do conceito de entidades hierárquicas e dinâmicas, a fim de prover maior dinamismo e flexibilidade aos ambientes que desejam utiliza-lo como mecanismo de gerenciamento de seus contextos.

1.2 Definição do Problema e Questão de pesquisa

A partir dos fatores listados na seção anterior, torna-se interessante a criação de um *framework* que seja capaz de representar a dinamicidade de um ambiente, permitindo o uso de entidades que sejam capazes de transitar a outros ambientes, o que neste trabalho é tratado como uma mudança hierárquica de uma entidade. Outro importante fator para essa dinamicidade é a possibilidade de representar uma quantidade variável de dados de contexto a qualquer momento por parte de uma entidade que compõe o cenário.

Com base no estudo de trabalhos relacionados à dissertação, surge a seguinte questão de pesquisa: “Quais as contribuições e limitações da utilização de históricos de contextos baseados em entidades hierárquicas e dinâmicas, na criação de sistemas sensíveis a contexto?”.

1.3 Objetivos

Com o desenvolvimento deste trabalho, propõe-se a criação do *framework* CMFrame*,* que forneça suporte ao gerenciamento de históricos de contextos em ambientes físicos compostos por sensores e atuadores, permitindo o uso de suas entidades hierárquicas e dinâmicas. As aplicações que venham a afetar um determinado ambiente, e cuja lógica permita o uso de entidades reais e virtuais poderão fazer uso do *framework* proposto*.*

A partir da criação de uma forma padrão em que a informação de contexto é tratada internamente pelo *framework*, tem-se o intuito de diminuir o esforço necessário para que diferentes aplicações possam usufruir do mesmo conjunto de dados de uma forma clara e definida, facilitando possíveis interações futuras entre aplicações. Uma padronização também permitirá uma maior facilidade, por parte do desenvolvedor, de interpretar como as informações de contexto poderão ser acessadas e utilizadas entre diferentes aplicações que venham a utilizar do gerenciamento de históricos, e que afetem ou não a mesma área de influência. Outro fator importante da padronização é o estímulo à adoção do *framework* como ferramenta, visto que sua estrutura possui documentação e permite que aplicações utilizem do mesmo protocolo, permitindo sua coexistência nos mesmos ambientes.

O emprego de sensores e atuadores embutidos em salas ou ambientes físicos, com o intuito de reagir automaticamente aos usuários para prover uma melhor experiência, adiciona aos ambientes o quesito inteligência, tornando-os ambientes inteligentes. Considerando que ambientes inteligentes englobam uma gama de objetos interconectados por uma rede, cada um com seus endereços únicos, torna-se possível afirmar que ambientes inteligentes se traduzem em ambientes povoado de objetos IoT, uma vez que a natureza de seus objetos são as mesmas. Fortino e Trunfio (2014) definem esses objetos heterogêneos interconectados como sensores, atuadores, dispositivos inteligentes, objetos inteligentes, RFID, computadores embutidos, entre outros. Esses objetos heterogêneos distribuídos estrategicamente geram conjuntos de dados constantemente, e ao mesmo tempo permitem que esses dados sejam usufruídos em prol dos usuários desses ambientes. Com base nesses aspectos, tem-se o intuito de desenvolver um *framework* que permita ao desenvolvedor gerenciar as informações de contexto adquiridas em seu ambiente.

Espera-se prover funcionalidades que permitam o armazenamento, consulta e manipulação desses contextos registrados de forma eficiente para diferentes cenários

de IoT, oferecendo ao desenvolvedor uma forma de agrupar os elementos que compõem o seu ambiente em uma única entidade, através da organização hierárquica. Foi desenvolvido um cenário de testes em conjunto com o *framework* a fim de validar as diferentes funcionalidades propostas neste trabalho, aplicando-o em um cenário que caracterize um ambiente inteligente e que venha a utilizar de diferentes informações de contexto previamente coletadas.

A fim de alcançar esse propósito, destacam-se os seguintes objetivos específicos:

- Definir e descrever um *framework* para o suporte ao desenvolvimento de aplicações que utilizem históricos de contextos em ambientes que empreguem entidades hierárquicas e dinâmicas;

- Estruturar o *framework* a fim de que sua expansão futura seja possível, permitindo a criação de novos mecanismos e funcionalidades que venham a agregar valor as suas funções primárias;

- Desenvolver o *framework* que suporte os aspectos atrelados ao gerenciamento de informações de contextos em ambientes físicos, compostos de elementos que podem ser representados por entidades reais e virtuais;

- Avaliar o funcionamento do *framework* através de sua utilização por duas aplicações distintas, com o intuito de averiguar as contribuições e limitações do uso de entidades hierárquicas e dinâmicas.

1.4 Metodologia e Organização do Trabalho

Este trabalho teve seu início a partir da busca por temas que fornecessem um embasamento teórico necessário para o entendimento acerca de funcionalidades e capacidades que deveriam ser providas pelo *framework.* Em seguida, foram realizadas pesquisas nas áreas diretamente relacionadas ao tema proposto. Buscou-

se identificar trabalhos com características semelhantes, a fim de estudar sua abordagem bem como os resultados obtidos. Foi então realizado um comparativo entre esses trabalhos com o intuito de identificar as características apresentadas por cada um e assim refinar a arquitetura do *framework*.

Com base nas etapas realizadas anteriormente, foi realizada a especificação inicial do *framework* a ser desenvolvido, detalhando seu funcionamento e principais características. Com o intuito de avaliar o trabalho proposto, foi desenvolvido o *framework* especificado, expondo suas particularidades quanto a parâmetros de eficiência e desempenho nos cenários propostos, bem como revelando suas limitações. O *framework* foi submetido a avaliação através do desenvolvimento de duas aplicações. Ambas são executadas simultaneamente e operam sobre o mesmo cenário de uma casa inteligente, portanto fazendo uso do mesmo conjunto de dados e empregando as funcionalidades que o *framework* oferta.

A disposição dos itens deste trabalho visa a exploração de seus tópicos de forma a facilitar o entendimento por parte do leitor. Dessa forma, o Capítulo 2 apresenta uma fundamentação sobre os conceitos atrelados ao trabalho desenvolvido, visando situar o leitor sobre os temas relacionados, e garantir um maior entendimento sobre o escopo abordado. No Capítulo 3 são apresentados os trabalhos relacionados e uma comparação entre os mesmos. O *framework* desenvolvido, com a descrição de sua arquitetura e funcionalidades é exposto no Capítulo 4. Os aspectos de implementação e metodologia para a avaliação serão descritos no Capítulo 5. Por fim, o Capítulo 6 apresenta as considerações finais e conclusão.

2. REFERENCIAL TEÓRICO

Este capítulo tem como objetivo esclarecer conceitos que são abordados ao longo do trabalho. Primeiramente são abordados os temas sobre contextos e históricos de contextos, na seção 2.1. A seção 2.2 apresenta uma explanação acerca de *frameworks*. Em seguida são descritos os conceitos relacionados à Internet das Coisas (IoT). A seção 2.4 apresenta os conceitos atrelados a sistemas cibernéticos-

físicos, que se encontram atrelados a sistemas IoT. Por fim são apresentadas considerações sobre o capítulo.

2.1 Contextos e Históricos de Contextos

Existem diferentes conceitos para a definição de contexto ao longo dos anos, (SCHILIT; ADAMS; WANT, 1994; PASCOE, 1998), sendo que, em 2001, Dey forneceu uma categorização que é utilizada por muitos autores até hoje. Sua classificação procurou refinar as definições existentes, visto que o contexto sempre será distinto conforme a situação que é relevante a uma determinada aplicação e seu conjunto de usuários, portanto definiu como contexto toda aquela informação que pode ser utilizada a fim de caracterizar a situação de uma entidade. Entidade nesse sentido pode se referir a uma pessoa, lugar ou até mesmo objetivo, desde que o mesmo seja considerado relevante para a interação entre um usuário e uma aplicação.

Atrelado a essas definições foram encontrados sistemas que são considerados sensíveis ao contexto (*Context-Aware*). A computação sensível ao contexto foi discutida primeiramente por Schilit e Theimer (1994) como um software que é capaz de se adaptar de acordo com a sua localização, e também conforme o conjunto de pessoas e objetos em sua proximidade, bem como com as mudanças que esses objetos sofrerão ao longo do tempo. Dey (2001) utiliza uma definição mais ampla para esses sistemas, na qual se caracterizaria como sensível ao contexto um sistema que emprega os dados de contexto para fornecer informações relevantes a seus usuários, ou até mesmo serviços, sendo que o quesito relevância é relativo à tarefa que o usuário irá desempenhar. Além disso, as três categorias propostas que uma aplicação sensível ao contexto pode prover dizem respeito a, apresentação da informação e de serviços a um usuário; a execução automática de serviços para o mesmo; e a conversão de contextos para uma informação a fim de que possa ser empregada futuramente pelo sistema.

A partir de aplicações que apresentam características de sensibilidade a contexto, existem aquelas que não somente empregam informações de contexto relativas ao presente, mas também aquelas que observam eventos do passado,

mediante ao registro dessas informações para uso posterior (ROSA et al, 2015). A coleção de contextos de eventos passados, registrados em uma ordem cronológica, é chamada de histórico de contextos (HONG et al, 2009; CIARAMELLA et al. 2010), sendo que alguns autores consideram esse histórico como trilhas (Driver, 2008; CAMBRUZZI, 2015). A utilização de históricos varia conforme a aplicação, porém no geral encontra-se associada a questões temporais, como agrupamentos de uma determinada área de influência que são relevantes a uma aplicação em um determinado espaço de tempo. Normalmente seu valor acentua-se quando agregada a outros tipos de informação como, por exemplo, o contexto do presente de um determinado cenário, bem como outras informações relevantes àquele sistema que podem ou não estar presentes previamente dentro do contexto.

2.2 Framework

Um *framework* pode ser considerado como uma aplicação semiacabada empregada no desenvolvimento de softwares, sendo considerado o sistema inteiro, ou até mesmo um subsistema (CRESPO, 2000). Outra definição foi provida por Mattsson (1996), que define um *framework* orientado a objetos como uma arquitetura desenvolvida com o máximo nível de reuso em mente, sendo representada como uma coleção de classes abstratas e concretas, e que encapsulam o seu comportamento em especializações de subclasses. *Frameworks* fornecem um ambiente reutilizável, com funcionalidades particulares que são incorporadas dentro de um ambiente de software de maior escopo. Seu intuito é o de facilitar o desenvolvimento de aplicações de software ao permitir que os desenvolvedores dediquem uma maior parcela de seu tempo a atingir os requerimentos de software, ao invés de lidar com a complexidade presente no desenvolvimento das funcionalidades em baixo nível, assim reduzindo o tempo total de desenvolvimento.

Uma das características principais que diferem as bibliotecas dos *frameworks* é a ideia de inversão de controle. Da maneira tradicional, as bibliotecas disponibilizam as aplicações componentes e funções que podem ser invocadas conforme a necessidade e desejo do desenvolvedor, a partir de um trecho de código em uma aplicação, porém, com a inversão de controle, é o *framework* que torna-se

responsável por realizar a invocação de um código específico, ou tarefa (FAYAD; SCHMIDT; JOHNSON,1999). Um exemplo desse princípio de design pode ser percebido em *callbacks*, *schedulers*, entre outros. Dessa forma, a inversão de controle é utilizada para aumentar a modularidade de um programa, bem como torna-lo extensível (JOHNSON; FOOTE, 1988).

Os principais objetivos encontrados em *frameworks* (MATTSSON, 1996) são:

- Manter o conhecimento de uma organização sobre aquele domínio de aplicação dentro da própria organização, visto que esta possui um ponto de partida (o *framework*) quando começa a desenvolver uma nova aplicação no mesmo domínio;
- Minimizar a quantidade de código necessário na implementação de aplicações similares;
- Deve ser ao mesmo tempo genérico a ponto de ser utilizado em mais de uma situação específica, bem como possuir um conjunto de pré-definido de subclasses que o usuário do *framework* possa utilizar imediatamente.

2.2.1 Classificação dos frameworks

Os *frameworks* podem ser caracterizados por diferentes dimensões, sendo elas: quanto ao domínio do problema a que o *framework* se aplica, a estrutura interna do *framework* e de que forma ele pretende ser utilizado (TALIGENT, 1994).

Quanto ao domínio do problema

Frameworks aplicados a um domínio do problema têm como enfoque o reuso do software, ao prover uma estrutura concisa através da identificação e definição de informações e componentes com o propósito de torná-los reutilizáveis na criação de novos sistemas. Dessa forma é construída uma arquitetura geral do domínio, que descreve a estrutura típica das aplicações para o domínio e sintetiza o conhecimento adquirido no mesmo.

- *Frameworks* de aplicação possuem funcionalidades que podem ser aplicadas a diferentes domínios como, por exemplo, aqueles que trabalham com interfaces gráficas de usuário (WEINAND; GAMMA, 1994);
- *Frameworks* de domínio acabam por capturar o conhecimento e técnica em um problema de domínio específico como, por exemplo, *frameworks* de controle de manufatura e multimídia (SCHMIDT, 1995);
- *Frameworks* de suporte oferecem serviços de baixo nível como, por exemplo, *drivers* para dispositivos, bem como o acesso a arquivos. (ANDERT, 1994).

Quanto à estrutura do *framework*

A partir da descrição da estrutura interna de um *framework* seu entendimento torna-se mais fácil. Buschmann (1994) define como *frameworks* arquiteturais aqueles que foram projetados de forma a capturar a estrutura principal de uma arquitetura de softwares orientada a objetos. Os princípios gerais para a estrutura interna de um *framework* orientado a objetos são descritos pela arquitetura do *framework*.

As arquiteturas de *frameworks* definidas por Buschmann (1994) são:

- *Framework* arquitetural em camadas: Auxilia na estruturação de aplicações que podem ser descompostas em grupos de subtarefas, cada uma com diferentes níveis de abstração;
- *Framework* arquitetural de *Pipes and Filters*: É utilizado para estruturar aplicações que podem ser divididas em várias subtarefas completamente independentes, que devem ser realizadas de forma fortemente sequencial ou em paralelo;
- *Framework* com arquitetura MVC (*Model-View-Controller*): Define uma estrutura para aplicações interativas, que desacopla os aspectos de interface de seu núcleo de funcionalidades;
- *Framework* com arquitetura *Presentation-Abstraction-Controller*: Demonstra-se útil para estruturar softwares que utilizam de alta interatividade com usuários, permitindo múltiplas apresentações e controles de seus modelos de abstração, e que podem ser compostos de subfunções independentes entre si;

- *Framework* com arquitetura reflexiva: Utilizada em aplicações que devem considerar a adaptação futura de ambientes mutáveis, tecnologias, bem como requisitos, mas sem uma alteração explicita a sua estrutura e implementação;
- *Framework* com arquitetura *Microkernel*: Utilizado nos softwares que proveem diferentes visões sobre sua funcionalidade e que tenham que se adaptar a novos requerimentos de sistema como, por exemplo, sistemas operacionais;
- *Framework* com arquitetura *Blackboard*: Auxilia na estruturação de aplicações complexas que envolvem subsistemas altamente especializados para diferentes domínios. Tais subsistemas cooperam entre si para construir soluções ao problema abordado;
- *Framework* com arquitetura *Broker*: É utilizado em sistemas distribuídos, em que componentes desacoplados interagem através de chamadas de operação remota no estilo cliente-servidor.

Quanto ao uso do *framework*

Um *framework* orientado a objetos pode ser utilizado de duas formas, em uma delas seus usuários criam novas derivações de suas classes, ou então combina classes já existentes.

O primeiro método é chamado *architecture-driven* ou *inheritance-focused*, e sua proposta consiste em desenvolver aplicações que dependam do mecanismo de herança, sendo assim, os usuários farão a adaptação do *framework* através da derivação de classes e de operações de sobrescrita. Um problema encontrado nesse tipo de *framework* reside no fato de que pode ser difícil adaptá-los, já que o usuário deve prover a implementação para o comportamento desejado.

O segundo, por sua vez, é chamado de *data-driven* ou *composition-focused,* e sua premissa está na composição de objetos existentes. Quais objetos que podem ser combinados são definidos pelo *framework*, porém as funções realizadas pelo mesmo são dependentes de quais objetos foram passados pelo usuário para o *framework*. Normalmente esses *frameworks* tendem a ser mais simples em sua utilização, porém ao mesmo tempo mais limitados em suas funcionalidades.

2.3 Internet das Coisas

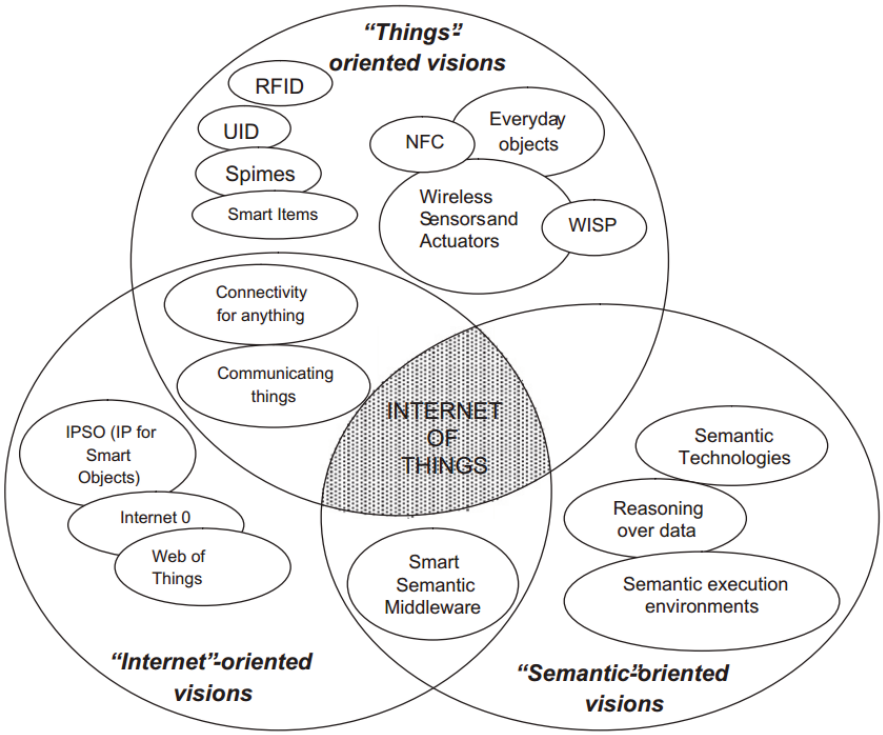
A Internet das Coisas (IoT, do inglês, *Internet of Things*), também conhecida como Internet dos Objetos, refere-se a interconexão em rede de objetos presentes no dia-a-dia. É descrita como uma rede wireless de sensores autoconfiguráveis cujo propósito se encontra em interconectar todas as coisas. Esse conceito foi originalmente introduzido pelo *Auto-ID Labs*, fundado em 1999 *no Massachusetts Institute of Technology (*MIT*).* Hoje em dia o Auto-ID Labs é o líder de pesquisas acadêmicas no campo de redes em RFID, sendo composta por sete das mais renomadas universidades de pesquisa, localizadas em quatro continentes distintos, que foram selecionadas, em conjunto com a EPCglobal, para definir assuntos relacionados a Internet das Coisas (MÖLLER, 2016; POSTCAPES, 2017).

Para IoT, uma coisa, objeto ou entidade se classifica como qualquer item possível no mundo real que é capaz de se juntar a rede de comunicação. Seu objeto principal é o *Radio-Frequency Identification* (RFID), portanto, IoT pode ser considerado como uma infraestrutura de *tags* RFID, gerando uma camada wireless acima da Internet (MÖLLER, 2016). Outra definição de maior abrangência apresenta internet das coisas como a presença pervasiva de uma variedade de coisas e objetos no ambiente, que através de conexões com e sem fio e de seu sistema de endereçamento único são capazes de interagir entre si e cooperar com outros objetos e coisas para criar novas aplicações e serviços e alcançar objetivos comuns (PATEL, K.; PATEL, S., 2016). A concepção de coisas e objetos presentes em *IoT* remete a objetos do dia-a-dia, que são passíveis de leitura, reconhecíveis, localizáveis e endereçáveis através de um objeto com capacidades sensores e que pode ou não ser controlado pela internet independente de seu meio de comunicação. Dentre os objetos do cotidiano, é possível encontrar não só objetos de maior teor tecnológico, como veículos e equipamentos, mas também aqueles que não são considerados eletrônicos, como é o caso de alimentos, peças de roupa, cadeiras, árvores, entre outros (PATEL, K.; PATEL, S., 2016).

De forma simplificada, em IoT é possível encontrar uma rede de computadores interconectada, que se comunica com uma rede de objetos interconectada, constantemente rastreando e processando milhões de objetos. Muitos desses objetos possuem seu próprio endereço IP (*Internet Protocol)*, encontram-se imersos em

sistemas complexos, e utilizam de sensores para obter diversas informações sobre o ambiente em que se encontram inseridos (MÖLLER, 2016). Um exemplo pode ser visto em um cenário com produtos alimentícios que, ao longo de sua passagem pelos processos da indústria tem seus dados de temperatura monitorados, e dessa forma pode fazer uso de atuadores para interagir com o mundo físico ao longo de todo o processo.

Figura 1: Convergência de diferentes visões para a criação do IoT



Fonte: (ATZORI; IERA; MORABITO, 2010).

O termo IoT não pode ser definido unicamente de uma perspectiva, porque muitas vezes sua aplicação é mais voltada de uma perspectiva "orientada a Internet" ou "voltada às coisas", dependendo dos objetivos a que se deseja alcançar. A Figura 1 apresenta as três principais visões, com os conceitos e tecnologias que melhor se encaixam em cada uma. A visão orientada a "coisas" demonstra os objetos incorporados pelo sistema e interligados entre si. A visão orientada a internet representa o endereçamento utilizado para interligar os diferentes objetos, mantendo-os no mesmo patamar de comunicação. Por fim, a visão semântica descreve as representações que são aplicadas a esses diferentes conceitos. A partir dessa figura é possível notar que a internet das coisas resulta da interação e agregação desse conjunto principal de visões.

O crescente poder de processamento disponível até mesmos nos menores dispositivos na computação em rede, bem como o crescimento de tecnologias de sensores e RFID, tem tornado IoT um conceito muito importante na economia global, visto que a tecnologia wireless está facilitando a interação com IoT em qualquer lugar a qualquer momento, agregando valor na conexão de diversos ambientes, como sistemas de energia, ambientes de saúde, sistemas de transporte, entre outros. Esse cenário abre espaço para novos produtos e serviços baseados em ubiquidade, que trazem um grande nível de inovação e alto impacto na sociedade e nos negócios (MÖLLER, 2016).

Abaixo lista-se algumas tecnologias que tornam o IoT possível:

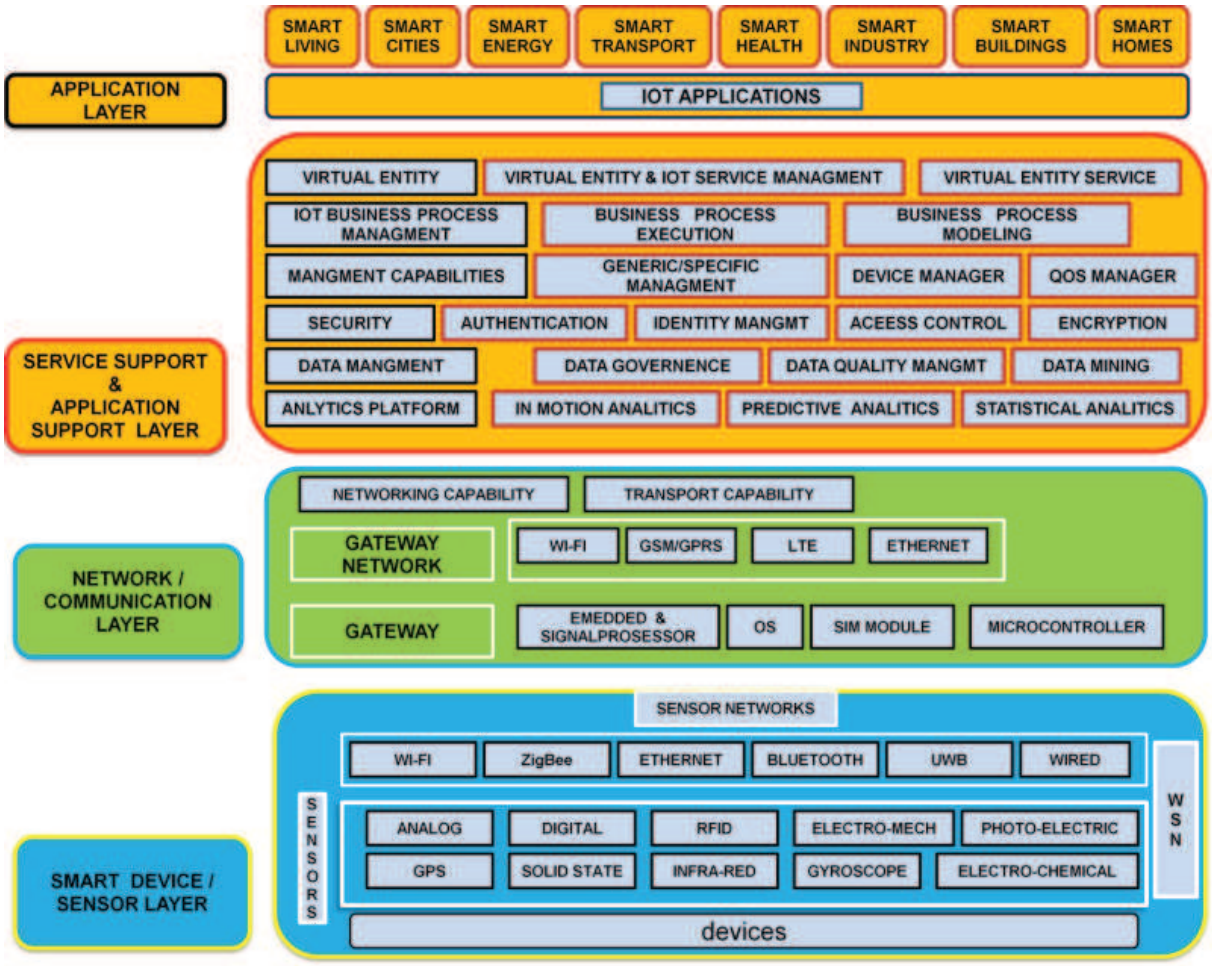
- RFID;
- Sensores e Atuadores;
- Nanotecnologia;
- Entidades Inteligentes;
- Wi-Fi;
- M2M;
- WSN;
- GPS.

Aplicações de IoT orientadas a entidades monitoram seus objetos em tempo real, e dependendo de sua situação, podem reagir automaticamente. Essa premissa resultou na criação de objetos inteligentes, ou ao menos objetos que agem de forma mais inteligente que aqueles que não se encontram interconectados ou equipados com sensores e atuadores (MÖLLER, 2016). De acordo com Chaouchi (2010), a utilização de objetos inteligentes abrirá discussões em tópicos como:

- Endereçamento, identificação e nomeação;
- Escolha de modelos de transporte;
- Modelo de comunicação dos objetos ou coisas conectadas;
- Tecnologia de conexão para objetos ou coisas;
- Impacto econômico e o valor da cadeia de evolução da telecomunicação;
- Interoperabilidade entre objetos e coisas;
- Possibilidade de interação com modelos já existentes, como a Internet;

- Segurança e privacidade;
- Entre outros.

Figura 2: Arquitetura IoT



Fonte: (PATEL K.; PATEL, S., 2016).

A Figura 2 mostra a arquitetura de IoT subdividida em camadas, no topo se encontram as aplicações nos domínios de cidades inteligentes, *smart grids,* sistemas de saúde inteligente, *smart lighting,* automação, casas inteligentes, entre outros. Em seguida são apresentados os mecanismos que proveem suporte a camada de aplicação e que oferecem funcionalidades de IoT na forma de serviços. A camada de rede e comunicação é responsável por oferecer funcionalidades de gateway entre os dispositivos sensores para a camada de suporte, provendo mecanismos de rede. A primeira camada representa dispositivos, sensores, bem como as redes de sensores utilizadas para sua comunicação.

Dentre as características relevantes para *IoT*, destacam-se (PATEL, K.; PATEL, S., 2016):

- Interconectividade: Todos objetos podem ser interconectados dentro da infraestrutura de comunicação;
- Serviços relacionados a coisas: Capacidade de fornecer serviços relacionados a objetos e coisas considerando as limitações que elas possuem, como questões de proteção de privacidade e consistência entre suas representações físicas e virtuais;
- Heterogeneidade: Dispositivos são heterogêneos, com base em diferentes plataformas de hardware e redes;
- Mudanças dinâmicas: O estado dos dispositivos, bem como seu número, muda dinamicamente, como conectado/desconectado e acordado/dormindo;
- Larga escala: A quantidade de dispositivos que necessitam de gerencia e comunicação será de uma maior magnitude que aqueles atualmente conectados à Internet. Nesses casos o gerenciamento dos dados gerador e sua interpretação se tornam ainda mais críticos;
- Segurança: Deve-se considerar vários aspectos de segurança, como a proteção de dados pessoais, durante toda a cadeia do sistema, tornando necessário criar um modelo de segurança escalável;
- Conectividade: Permite a acessibilidade e compatibilidade dentro de uma rede, onde acessibilidade seria a capacidade de acessar uma determinada rede e compatibilidade sua habilidade de consumir e produzir dados.

2.4 Sistemas Ciber-físicos

Sistemas ciber-físicos (CPS, do inglês *Cyber Physical Systems*) foi um termo introduzido em 2006 nos Estados Unidos na *National Science Foundation* (LEE, 2015), para designar aqueles sistemas em que a interconexão entre o mundo físico e os sistemas de computação é de suma importância. Conforme a evolução da tecnologia torna-se cada vez mais difícil identificar sistemas cuja necessidade de expansão e configuração não os classifica como ciber-físicos, fato este que se deve

principalmente a incorporação de componentes eletrônicos e softwares em diversas áreas da vida cotidiana.

CPS possuem uma forte relação com tópicos como IoT, Indústria 4.0, *Internet of Everything*, *Machine-to-machine*, sendo que todos esses apresentam diferentes conceitos de tecnologias que conectam aspectos físicos do mundo para com uma rede de informações.

Esses sistemas podem ser aplicados em uma pequena escala (como marca-passos), até mesmo sistemas gigantescos, como o gerenciamento de uma rede de energia de uma nação, fator este que torna difícil avaliar a efetividade e relevância desse tipo de sistema (TÖRNGREN et al., 2017). Dentre alguns de seus exemplos pode-se elencar sistemas automotivos, sistemas de manufatura, controle de tráfego, geração e distribuição de energia, conservação de energia, sistemas de gerenciamento e distribuição de água, dispositivos médicos, incluindo a assistência de indivíduos a distância, como a telemedicina.

De forma geral, sistemas ciber-físicos são capazes de coletar um grande conjunto de dados sob o cenário no qual atuam, utilizando as mais diversas informações sobre o mundo físico naquele cenário, e então distribuindo essas informações dentro de uma estrutura robusta, que melhor represente a necessidade específica daquele sistema. Essa estrutura organizacional permite que sejam realizadas inferências posteriores, com base nas informações coletadas em um período específico de tempo, e assim modelar o comportamento que o sistema deve tomar sobre as entidades a que gerencia. Considerando o fato de que esses sistemas agregam características, bem como a modelagem de comportamento de diversas entidades simultaneamente, torna-se muito difícil manter um baixo teor de complexidade em sua operação. Essa complexidade, atrelada ao fato de que na grande maioria dos casos esses sistemas não irão operar um ambiente controlado, torna imprescindível a criação de sistemas robustos o suficiente para lidar com condições inesperadas, adaptando seu comportamento para prover a melhor solução possível para uma determinada adversidade (LEE, 2008).

Existem diferentes visões acerca das diferenças entre CPS e IoT, variando conforme a perspectiva de determinado autor e da comunidade científica em que atua. Alguns autores classificam IoT como uma subclasse pertencente a sistemas CPS

(STOJMENOVIC, 2014), outra visão apresenta uma igualdade entre os termos de CPS e IoT, apresentando-os como o mesmo conceito (SIMMON; SOWE; ZETTSU, 2015). Pode-se encontrar também autores que definem tais termos como sistemas com conceitos fortemente relacionados que foram desenvolvidos por dois grupos com perspectivas distintas (NUNES; ZHANG; SILVA, 2015).

O conceito de CPS deriva da perspectiva de engenharia, preocupando-se com o controle e monitoramento de ambientes e fenômenos físicos, utilizando de sistemas que são capazes de sentir e atuar através de diversos dispositivos de computação distribuídos (KOUBÂA; ANDERSSON, 2009).

IoT por sua vez teve seu desenvolvimento a partir da perspectiva da ciência da computação, tendo como objetivo o desenvolvimento de uma rede de computadores conectados a objetos com capacidades de autoconfiguração que funcionam sobre a Internet. IoT preocupa-se com uma série de requerimentos, como o desenvolvimento da inteligência em dispositivos, interfaces e serviços, a garantia de segurança, bem como privacidade, gerenciamento de dados, integração entre sistemas e a interoperabilidade de comunicação (VERMESAN; FRIESS, 2014). Portanto, *IoT* tende a uma maior ênfase em aspectos de rede de dispositivos inteligentes, enquanto que CPS mantem uma maior preocupação na aplicabilidade e modelagem dos processos físicos e resolução de problemas através de sistemas fechados. As diversas similaridades entre esses dois conceitos, como o intenso processamento de informações, robusta utilização de serviços inteligentes e uma troca de dados eficientes tem levado a utilização de ambos os termos no mesmo contexto, sem que seja estabelecida uma clara identificação de seus limites (JESCHKE, 2013; KOUBÂA; ANDERSSON, 2009).

2.5 Considerações sobre o capítulo

Este capítulo abordou questões referentes a contextos, históricos de contextos, *frameworks*, Internet das coisas e sistemas ciber-físicos. Esses temas trazem relação com o trabalho proposto, apresentando as principais vertentes sobre as quais se dará o seu desenvolvimento. O tópico sobre internet das coisas visa demonstrar os conceitos iniciais que irão fornecer a base para a criação de ambientes que utilizem de um modelo que contemple entidades reais e virtuais, e que por sua vez constituem

o cenário necessário para que possa ser desenvolvido um gerenciador de históricos de contexto.

O tema de sistemas cibernéticos físicos foi discutido devido ao fato de que representa um possível direcionamento do modelo de internet das coisas, o levando a um maior escopo que considera novas necessidades a serem preenchidas, tendo uma maior ênfase em sistemas responsivos, que considerem fatores atrelados ao tempo de resposta curto e capacidades de gerenciamento de uma grande estrutura com base nas informações contextuais geradas pelo ambiente. Considerando que o *framework* aborda o gerenciamento de históricos de contextos para ambientes que façam uso de entidades reais e virtuais, uma de suas possíveis aplicações seria o cenário provido por diversos ambientes cibernéticos-físicos, em que o gerenciamento desse histórico comporia uma parcela do sistema e permitiria que o mesmo tivesse um mecanismo que o auxiliasse a lidar com questões complexas de inferência sobre a situação atual.

No próximo capítulo serão apresentados os trabalhos relacionados com a proposta, analisando e elaborando um comparativo entre suas principais características.

3. TRABALHOS RELACIONADOS

Neste capítulo serão analisados os trabalhos de maior relevância em relação ao trabalho proposto com base nas pesquisas acadêmicas realizadas, assim como um comparativo acerca de suas características. A primeira seção apresenta a metodologia empregada na escolha dos trabalhos. Nas seções 3.2 a 3.7 serão apresentados os trabalhos escolhidos para a análise. A seção 3.8 apresenta uma análise comparativa entre os trabalhos selecionados. A última seção aborda as considerações finais sobre o capítulo.

3.1 Metodologia para escolha dos trabalhos

Os trabalhos foram selecionados com pesquisas nas bases de periódicos da CAPES, sendo elas ACM Digital Library, IEEE Xplore Digital Library e ScienceDirect. Essas pesquisas foram guiadas com base no tópico de interesse, e buscavam trabalhos que apresentassem contribuições na área de *frameworks*, com algum tipo de utilização de informações de contexto, não necessariamente gerando históricos, ou seja, registrando essas informações em uma base de dados para uso posterior. Todas as buscas foram realizadas através da junção de termos relacionados, baseados na seguinte expressão: (("context") AND ("framework") AND (“smart environment“) AND ("internet of things" OR "IoT")), com pequenas alterações para satisfazer as condições impostas em cada base de dados. Essa pesquisa não limitou os trabalhos por nenhum filtro referente a datas de publicação, tendo sido englobados artigos até fevereiro de 2017.

A partir da expressão acima é possível perceber que a seleção dos trabalhos levou em consideração a integração de *frameworks* que possuíam algum tipo de relação com contextos, bem como a adição de elementos característicos de ambientes inteligentes (*smart environments*).

A fim de expandir o repertório de trabalhos que abordassem tópicos relevantes ao tema, optou-se pela realização de novas pesquisas com outros conjuntos de expressões, que foram adicionados aos trabalhos já coletados na pesquisa acima. Entre outros termos pesquisados, pode-se citar a adição de termos que contemplam sistemas CPS, visto que apresentam um nível de maior complexidade do que os vistos em ambientes inteligentes, e que são projetados para representar seus elementos através de entidades reais e virtuais. Também buscou-se relacionar a pesquisa mesclando termos mais genéricos, como a busca por *frameworks* levando em considerações termos como *context-aware* e *context history*. Em aspectos gerais, todos os termos pesquisados visaram abranger o maior número possível de trabalhos que tivessem uma ligação com o tema proposto neste trabalho.

Todos os resultados encontrados foram adicionados às pesquisas iniciais, e filtrados de acordo com o processo proveniente de um mapeamento sistemático

(PETERSEN; VAKKALANKA; KUZNIARZ, 2015). Entre as bases selecionadas, foram encontrados um total de 1302 trabalhos que correspondem à expressão escolhida, conforme a Tabela 1, e, a partir desses trabalhos, teve início o processo de filtragem.

Tabela 1: Seleção de trabalhos

	ScienceDirect	IEEE	ACM	TOTAL
Pesquisa Inicial	178	318	806	1302
Remoção de duplicados	178	317	805	1300
Remoção de impurezas	52	149	341	542
Filtro por título	8	28	57	93
Filtro por abstract	3	8	6	17
Filtro por Introdução/Conclusão	1	4	1	6
Filtro por Full Text	0	4	0	4

Fonte: Elaborado pelo autor.

O processo envolveu diversos filtros por etapas, sendo o primeiro a remoção de artigos duplicados, que resultou na eliminação de dois artigos. Em seguida a remoção por impurezas, que consiste em avaliar os critérios de inclusão e exclusão dos artigos. São eles:

Os critérios de inclusão foram os seguintes:

- Atender aos critérios de relevância frente aos tópicos de interesse do estudo;
- Pertencer ao período de publicação estabelecido.

Dentre os critérios de exclusão, encontram-se:

- Título e abstract fora do contexto investigado;
- Artigos que não se classificam como um full paper;
- Artigos que não se encontram em inglês;
- Artigos de áreas não relevantes.

Após concluída essa etapa de filtragem, restaram 542 trabalhos, que foram submetidos para a próxima etapa. O filtro por títulos avaliou os trabalhos e eliminou aqueles que se mostravam fora dos tópicos estabelecidos, restando assim 93 trabalhos. Em seguida foi realizada a filtragem por *abstract*, onde foi possível reduzir ainda mais os resultados finais, todos os trabalhos que não possuíam em seu abstract

relação com os temas almejados foram removidos, fazendo com que restassem apenas 17 artigos. A próxima etapa considerou apenas a introdução e conclusão dos artigos disponíveis, processo esse que reduziu o número de trabalhos para 6. A etapa final, por sua vez, constitui-se na leitura de todo o artigo, a fim de identificar sua relevância para com o intuito do mapeamento. Após essa análise, apenas 4 artigos foram selecionados. Posteriormente ao processo de filtragem dos artigos coletados, houve a adição de dois trabalhos relevantes que foram encontrados por pesquisas não relacionadas ao termo inicial de pesquisam totalizando todos os trabalhos relacionados. Em seguida, os mesmos foram submetidos a uma análise comparativa, que utilizou como critério os fatores expostos na seção 3.8. As seções seguintes apresentam os trabalhos relacionados encontrados.

3.2 Comprehensive Framework for Mobile Cyber-Physical Applications

Este *framework* captura diferentes tipos de contextos em dispositivos móveis, guarda-os em um repositório de contextos e realiza inferências sobre informações de contexto para gerar dados de predição acerca de um usuário como, por exemplo, seu possível comportamento futuro (LEE; CHEUN; KIM, 2011). A proposta dos autores é de torná-lo reutilizável entre diversas aplicações móveis que lidam com CPS. Para maior clareza, o *framework* será referenciado, de agora em diante, como CPA. O CPA foi dividido em três partes, com base em funcionalidades, apresentadas a seguir:

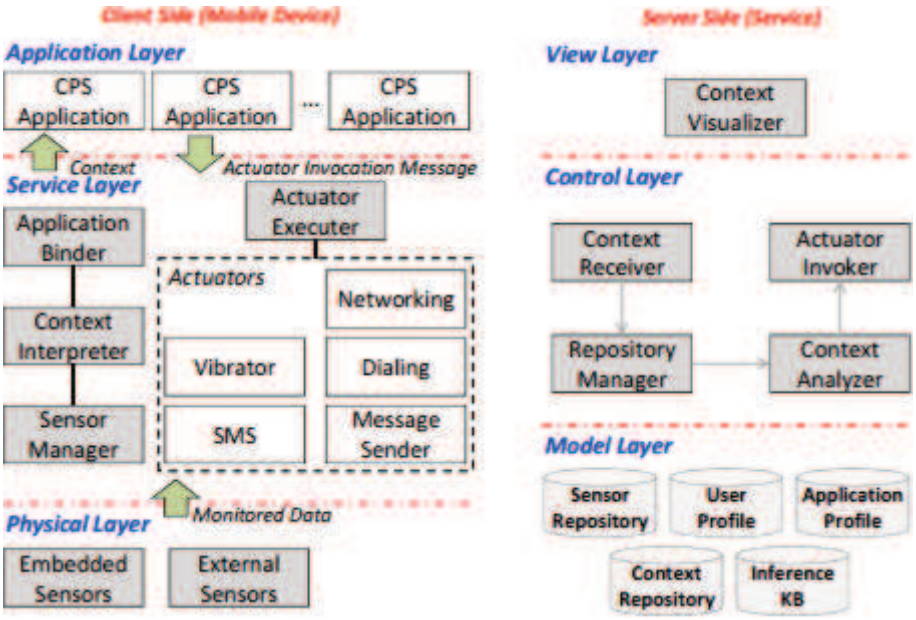
- *Acquisition of context*: Esse processo de aquisição de informações de contexto é realizado com a utilização de sensores;
- *Abstraction and understanding of context*: É o processamento do contexto, responsável por criar a relação entre o estímulo sensorial e atrela-lo a um contexto, processo este feito através de vários mecanismos de inteligência artificial, como *pattern matching;*
- *Reaction based on the recognized context*: Responsável por ativar uma ação com base no resultado do processamento de um *context*. Em sua forma mais básica, essa reação seria simplesmente notificar o resultado de um contexto processado.

As funcionalidades presentes no CPA são oferecidas na forma de serviços, e seus consumidores podem utiliza-las a qualquer momento e em qualquer local. Através da distribuição de funcionalidades complexas, diminui-se a carga nos clientes, e se torna possível substituir dinamicamente um determinado serviço, se necessário. A Figura 3 mostra a arquitetura do *framework*, dividida entre a parte do cliente e a parte do servidor. A parte do cliente possui três camadas distintas:

- *Physical Layer:* contém sensores internos dos dispositivos móveis, como acelerômetro, GPS, giroscópio, bem como sensores externos como, por exemplo, um monitorador de taxa cardíaca. Ainda conta com os protocolos de comunicação entre esses sensores e a aplicação cibernético-física;
- *Service Layer:* é responsável por converter os dados monitorados em contexto, interpretar os diferentes conjuntos de contexto, gerir a distribuição de dados de contexto para as aplicações CPS na *Application Layer*, e executar as ações nos atuadores;
- *Application Layer:* contém todas as aplicações CPS que utilizam de contexto e a situação do usuário a partir da *Service layer*.

Caso uma condição de contexto seja satisfeita, a aplicação CPS irá solicitar ao *Actuator Executor*, para que este utilize um atuador como *Vibrator, SMS, Networking, Dialing,* ou *Message Sender.* Consideremos um paciente cuja taxa cardíaca desce abaixo de 70 *bpm*, a partir desse momento, a aplicação CPS de saúde enviaria uma mensagem SMS para o médico daquele paciente informando a situação em que o mesmo se encontra.

Figura 3: Arquitetura geral do CPA



Fonte: (LEE; CHEUN; KIM, 2011).

No lado do servidor o conceito *Model-View-Control (MVC)* é aplicado. A *Model layer* contém cinco diferentes bases de dados.

- *Sensor Repository* guarda todas as informações de sensores;
- *User Profile* guarda todas as informações pessoais dos usuários, permitindo assim a personalização do serviço ao inferir o contexto do usuário;
- *Application Profile* guarda a informação de cada aplicação CPS;
- *Context Repository* serve para guardar as diferentes informações de contexto recebidas dos dispositivos móveis registrados;
- *Inference Knowledgebase* contém regras de inferência dos contextos, utilizados para interpretar o contexto que é armazenado no *Context Repository*.

A camada de controle (*Control Layer*) contém quatro componentes principais. *Context Receiver* é utilizado para receber a informação de contexto enviada por um dispositivo móvel, bem como enviá-la a dois outros componentes, o *Context Repository Manager* para armazenamento, e ao *Context Visualizer*, para que seu conteúdo seja visualizado. O *Context Repository Manager* armazena os dados de contexto recebidos na base de dados *Context Repository. Context Analyzer* é responsável por analisar os padrões do passado, e possíveis futuros através de inferência baseada nos dados de contexto. Os resultados serão armazenados no *Context Repository*. Por fim, a *View Layer* possui um *Context Visualizer*, que será

responsável por mostrar as informações de contexto recebidas com base em configurações do usuário.

Entre os métodos para analisar o contexto, o CPA divide o contexto em três partes:

- *Current Context Reasoning* em que a informação adquirida dos sensores diretamente é utilizada para gerar uma informação significativa de contexto, como no caso de informações de um sensor GPS;
- *Past Context Reasoning* é utilizado para adquirir os padrões de uso de um determinado usuário como, por exemplo, a coleção de locais visitados pelo usuário;
- *Future Context Reasoning* é utilizado para deduzir as possíveis ações dos usuários; um exemplo seria deduzir o local em que o usuário visitará, com base na análise de seus padrões anteriores.

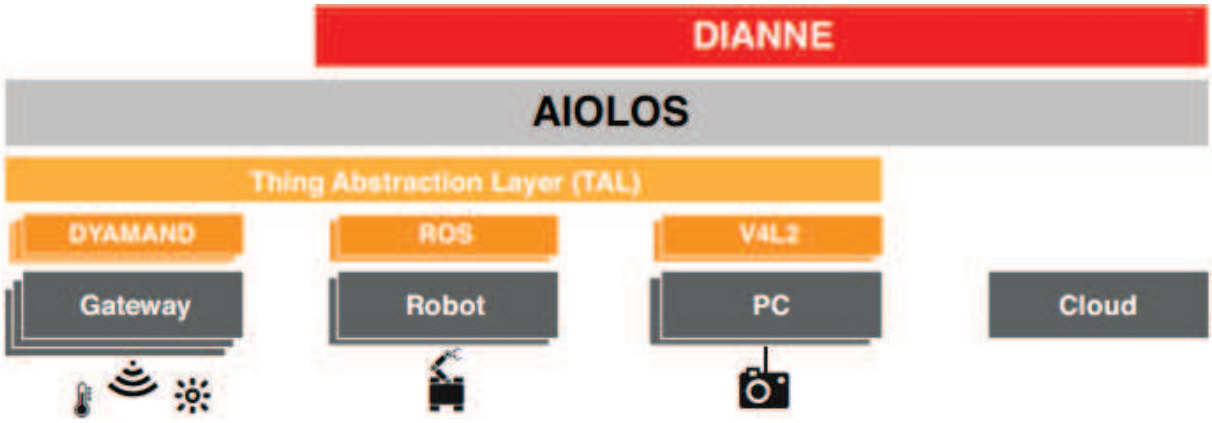
3.3 Middleware for Distributed Applications Incorporating Robots, Sensors and the Cloud

O trabalho propõe um *middleware* para sistemas CPS baseados em serviços, em um cenário utilizando-se robôs, atuadores, sensores em conjunto com a *nuvem*. O *middleware* ainda prove técnicas para desenvolver componentes com base em redes neurais, cuja popularidade cresceu recentemente na análise de dados sensoriais, bem como controle de robôs (CONINCK et. Al., 2016). Para maior clareza, o *middleware* será referenciado, de agora em diante, como MDA.

A Figura 4 ilustra as diferentes camadas presentes na arquitetura do MDA. A base do sistema é provida pelo AIOLOS, um *framework* de código aberto que permite a criação de aplicações baseadas em componentes, que podem ser distribuídos entre vários dispositivos sem que o desenvolvedor necessite gerenciar a comunicação entre eles. Os desenvolvedores podem criar componentes diretamente sobre o AIOLOS, ou utilizando de funcionalidades especificas providas pela camada DIANNE. DIANNE fornece serviços e abstrações utilizados para desenvolver componentes utilizando-se

de redes neurais. *Thing Abstraction Laye*r (TAL) é responsável por toda a comunicação entre os componentes da aplicação, sensores e robôs, sendo que sua implementação foi agregada ao *Robot Operating System* (ROS), cujo propósito se encontra no controle de robôs e ao DYAMAND, que é usado para o controle de sensores.

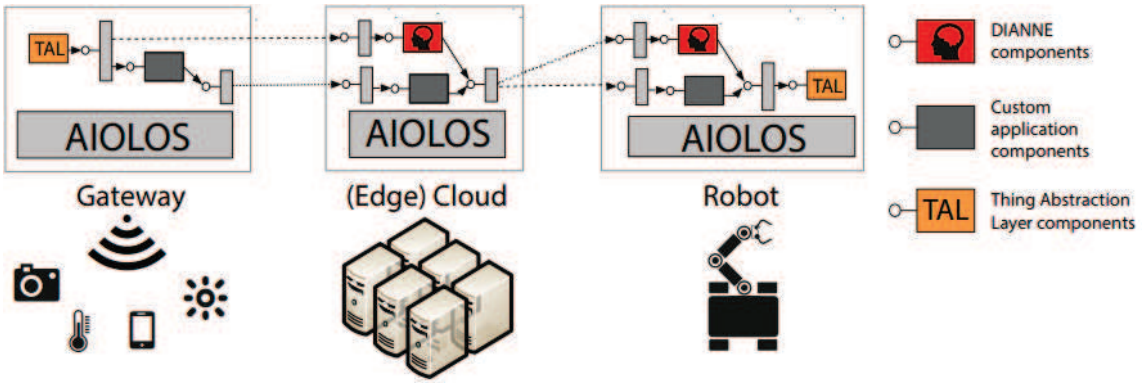
Figura 4: Arquitetura em camadas do MDA



Fonte: CONINCK et al. (2016).

Os componentes criados são distribuídos em pacotes *jar*, podendo ser executados em dispositivos que possuam uma AIOLOS *runtime*. Os pacotes podem ser inicializados e migrados entre *runtimes* a fim de criar um balanceamento de carga computacional, bem como permitir um ambiente dinâmico onde sensores e atuadores podem se conectar e desconectar a qualquer momento. Caso ocorra a desconexão de um serviço, o MDA é capaz de executar uma versão local de suas funcionalidades. O AIOLOS cria um proxy para todas as interfaces de serviço que se encontram em um mesmo nodo, bem como para aquelas que provem de outros nodos, conforme a Figura 5. As chamadas de métodos são enviadas pelo *proxy* para os componentes que implementam os serviços, sejam eles do mesmo nodo ou de nodos externos, o que permite que o AIOLOS acumule informações de monitoramento, como tempo de execução, valores de retorno, tamanho de argumentos, informações estas utilizadas no processo de decisão das políticas do proxy.

Figura 5: Interfaces de serviço no framework AIOLOS



Fonte: CONINCK et al. (2016).

TAL expõe os dispositivos físicos dentro de um serviço. Existem dois tipos de “Things”, ou coisas: sensores e atores. TAL oferece a funcionalidade de sensores como um serviço de um determinado tipo, que pode incluir temperatura, luz, câmera, entre outros. Já entre atuadores pode-se citar lâmpadas, fechaduras, entre outros. Cada tipo possui uma interface que pode ser customizada pelos desenvolvedores a fim de incorporar novas funcionalidades. DYAMAND é utilizado para fornecer a descoberta de serviços e protocolos de dispositivos com interfaces proprietárias, sendo que um *plugin* encapsula o DYAMAND como um provedor TAL, exportando assim todos dispositivos descobertos como serviços TAL. V4L2 (Video4Linux) foi incorporado ao TAL para prover acesso a sensores multimídia, como webcams, em sistemas Linux, e torna-los visíveis como coisas. Para atores robóticos foi utilizado o *Robot Operating System* (ROS), cujo proposito visa facilitar a criação de comportamentos complexos para robôs, sendo modular e utilizando uma API baseada em serviços com um sistema de envio de mensagens através do mecanismo *publish/subscribe.*

Para a demonstração de funcionamento desse sistema, os autores utilizaram um robô de fábrica (Kuka Youbot) e uma câmera externa. Foi utilizado um hardware Jetson TK1 equipado com GPU para servir como *cloud (edge*). Tanto o robô quanto a câmera são expostas como serviços pelo TAL. O vídeo da câmera é analisado por uma rede neural treinada para reconhecimento de objetos. Quando o robô se encontra conectado à cloud, esse pode transmitir as imagens captadas para a *cloud*, ou processá-las localmente utilizando um modelo com menor precisão, sendo que essa escolha é feita automaticamente baseada na avaliação da qualidade da conexão.

3.4 DTLC

O DTLC é um *framework* voltado à otimização do fluxo de trafego veicular em interseções, através da utilização de luzes dinâmicas em semáforos. Para isso os autores propõem duas formas de implementar uma solução (YOUNIS; MOAYERI, 2016). A primeira consiste em instalar ao longo das estradas um dispositivo computacional que prove conectividade aos veículos que transitam, chamada *road side unit* (RSU), esse dispositivo será responsável por coletar informações sobre as condições da rua. Ainda serão utilizados um controlador de luzes de trafego (TLC) em cada um dos semáforos a fim de controlar seu estado com algoritmos. Ao invés do RSU, podem ser utilizados sensores, desde que sejam capazes de se comunicar com o TLC. Essa implementação é chamada RITCO (*Road-based Infrastructure Traffic-light Control*).

A segunda solução é chamada VITCO (*Vehicle-bases Infrastructure Traffic-light Control*), em que se utiliza uma infraestrutura distribuída, com TLCs nos semáforos e com dispositivos de comunicação dentro dos veículos que se comunicam com a unidade de controle, fornecendo as chegadas de cada veículo e suas localizações nos segmentos de ruas. Essa solução reduz a necessidade da infraestrutura nas ruas, porém demanda que os veículos suportem o TLC (Younis; Moayeri, 2016).

A partir de uma análise dos protocolos, os autores concluíram que o RITCO apresenta as seguintes características:

- Quanto à comunicação: É independente dos veículos na estrada;
- Quanto ao processamento: A carga de trabalho é posta no RSU ou nos dispositivos ligados ao semáforo);
- Quanto ao custo: Deve ser pago pela cidade ou estado;
- Quanto a aplicação: Sua independência em relação a veículos evita problemas que possam surgir com defeitos de equipamentos, mas adiciona ao custo;
- Quanto à segurança: Mais seguro, pois não adquiri informações a partir dos veículos.

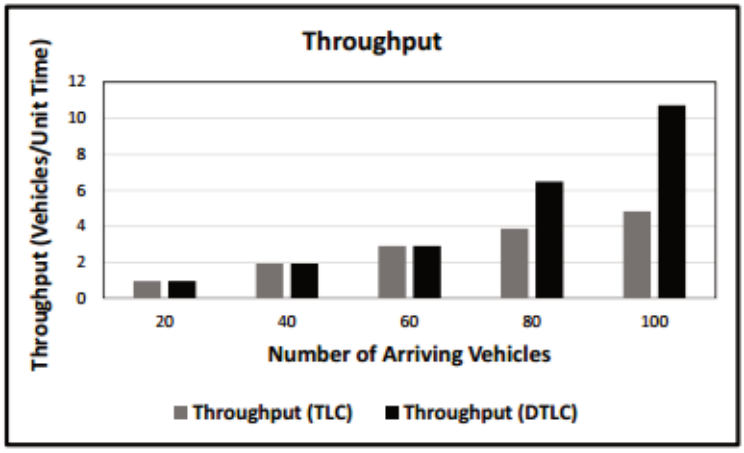
O VITCO, por sua vez, apresenta:

- Quanto à comunicação: Possui um maior custo de comunicação;
- Quanto ao processamento: A carga de trabalho é posta no veículo;
- Quanto ao custo: É compartilhado entre a cidade ou estado e os donos de veículos;
- Quanto à aplicação: Possui um período de transição até os veículos suportarem os equipamentos necessários ao sistema;
- Quanto à segurança: Decisões são realizadas pelos veículos, o que pode apresentar problemas com dispositivos que foram adulterados. A fim de proteger a privacidade, veículos poderiam utilizar de identificadores únicos.

Como condições para a troca do sinal, Verde->Vermelho e vice-versa, o *framework* leva em conta fatores síncronos, como o atual sistema de expiração por tempo determinado, bem como assíncrona, que considera certos eventos como *triggers*. Entre esses eventos pode-se citar: quando o número de veículos aguardando o sinal verde excede um valor, ou quando nenhum veículo cruzou a interseção com o sinal luz verde durante certo período de tempo, entre outros.

O *framework* foi validado considerando métricas como, tempo de espera, comprimento da linha de espera, e fluxo de trafego, sendo que os distintos protocolos não foram avaliados individualmente, visto que eles só diferem em como coletam a informação. A Figura 6 demonstra o fluxo de trafego em um semáforo com o *Dynamic Traffic Light Control* (DTLC) em funcionamento em comparação ao sistema normal. É possível observar que após o número de veículos ultrapassa o limite (definido em 75), o sistema entra em ação, ganhando vantagens em veículos por unidade de tempo.

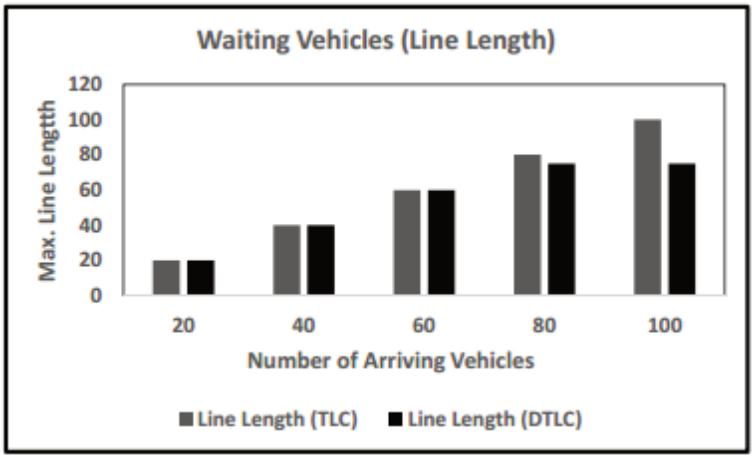
Figura 6: Fluxo de tráfego



Fonte: (YOUNIS; MOAYERI, 2016).

Na Figura 7 pode-se visualizar o número de veículos aguardando pelo sinal verde em um semáforo, novamente, quando um limite é atingido, o sistema DTLC passa a funcionar e ocorre a troca de sinal em período de tempo menor, conforme definido pelo *framework*. Esse comportamento auxilia na diminuição do tamanho da fila, garantindo que os congestionamentos serão tratados com antecedência.

Figura 7: Tamanho máximo da fila



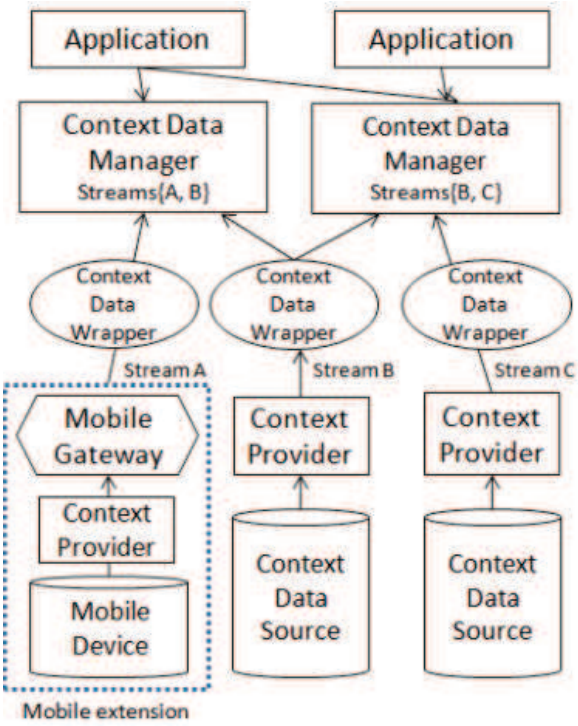
Fonte: (YOUNIS; MOAYERI, 2016).

3.5 ezContext

O trabalho propõe um *framework* para o gerenciamento automático do ciclo de vida de dados de contexto (MARTIN; LAMSFUS; ALZUA, 2010). O *framework*

apresenta três contribuições principais segundo os autores, sendo elas o gerenciamento automático do ciclo de vida de dados de contexto; uma extensão do *framework* para dispositivos móveis que visa habilitar a coleta de dados providos por eles; e a disponibilização de ferramentas visuais para auxiliar os desenvolvedores no estágio de prototipação e testes.

Figura 8: Arquitetura do ezContext



Fonte: MARTIN, LAMSFUS, ALZUA (2010).

Conforme pode ser visto na Figura 8, a arquitetura do *ezContext* é dividida em diferentes camadas, que serão explicadas a seguir:

- *Context Data Source*: Essa camada fornece dados de contexto aos níveis superiores da arquitetura. As *sources* podem ser distribuídas e fornecer dados heterogêneos, na forma de bases de dados, *web services*, ou sensores;

- *Context Provider*: Responsável por abstrair a coleta de contextos para as camadas superiores, fazendo com que qualquer mudança feita nos *Context sources* não afete o resto do sistema. Cada *Context Provider* envia a informação de contexto encapsulada por um objeto chamado *Context Data Wrapper* de forma assíncrona

para a próxima camada, denominada *Context Data Manager*. Cada provedor pode enviar informações referentes a um tipo de entidade, como uma pessoa, ou cidade, sendo que mais de um provedor pode enviar dados sobre o mesmo tipo de entidade de contexto. Os provedores são divididos em Proativos e Reativos. Proativos podem ser configurados para periodicamente acessar as informações de um *Context Data Source* e envia-las ao gerenciador, enquanto que os reativos irão aguardar até que um evento externo ocorra, como o pressionar de um botão ou a mudança da temperatura. Uma extensão mobile é disponibilizada para permitir a aquisição de contexto em ambientes móveis, na forma de um *Context Provider* no dispositivo móvel, que por padrão possui um provedor de localização, de movimento e um provedor social;

- *Mobile Gateway*: Existe na extensão mobile e é responsável por redirecionar as informações coletadas para o *Context Data Manager,* transformando os dados móveis em um *Context Data Wrapper Object;*

- *Context Data Manager:* Constitui a camada principal do *ezContext*, responsável por gerenciar o ciclo de vida de dados de contexto e realizando atividades como a conversão dos dados coletados no modelo de contexto, realizar a inferência sobre os dados, gerencia dos dados atuais de contexto, armazenamento de informações passadas de contexto e a disseminação das informações de contexto para a camada de aplicação.

- *Application*: Camada onde residem todas as aplicações, onde é permitido que requisições sejam enviadas ao *Context Data Manage*r para solicitar informações de contexto. O gerenciador é capaz de enviar notificações para as aplicações com base na situação do usuário, inferida a partir de informações de contexto coletadas.

Essa arquitetura pode ser configurada de forma distinta, utilizando-se de sua natureza *publish*/*subscribe* (EUGSTER et al., 2003), em que diferentes gerenciadores se encontram registrados a *Context Providers*, fazendo com que cada provedor possa publicar dados de contexto para diversos gerenciadores.

A fim de validar o *ezContext*, foi desenvolvimento um serviço de notificação que envia dados personalizados a dispositivos móveis do usuário, de acordo com a sua situação. O primeiro modelo envia notificações a usuários que se encontram aguardando pelo ônibus, e informava o tempo restante estimado para a chegada do mesmo. O segundo sugere restaurantes para usuários que se encontram caminhando próximos a centros de lazer em um horário próximo ao almoço. Foi constatado pelos autores que a utilização do *ezContext* causou grande redução na quantidade de código gerado pelo desenvolvedor, sendo essa uma das propostas do *framework*, uma vez que este oferece funcionalidades de alto nível para lidar com as tarefas complexas de baixo nível atreladas a esse tipo de desenvolvimento, ao mesmo tempo fornecendo a aplicações a capacidade de se adaptarem e reagirem a informações de contexto dos usuários (MARTIN; LAMSFUS; ALZUA, 2010).

3.6 Toward a Real-Time Framework in Cloudlet-Based Architecture

O trabalho propõe um *framework* com base em uma arquitetura de *cloudlets* para modelos de predição de requerimentos de usuários em tempo real em dispositivos móveis (KOTEVSKA; LBATH; BOUZEFRANE, 2016). Para maior clareza, o mesmo será referenciado, de agora em diante, como CLET. Sua proposta organiza o sistema em alguns atores, a serem definidos a seguir:

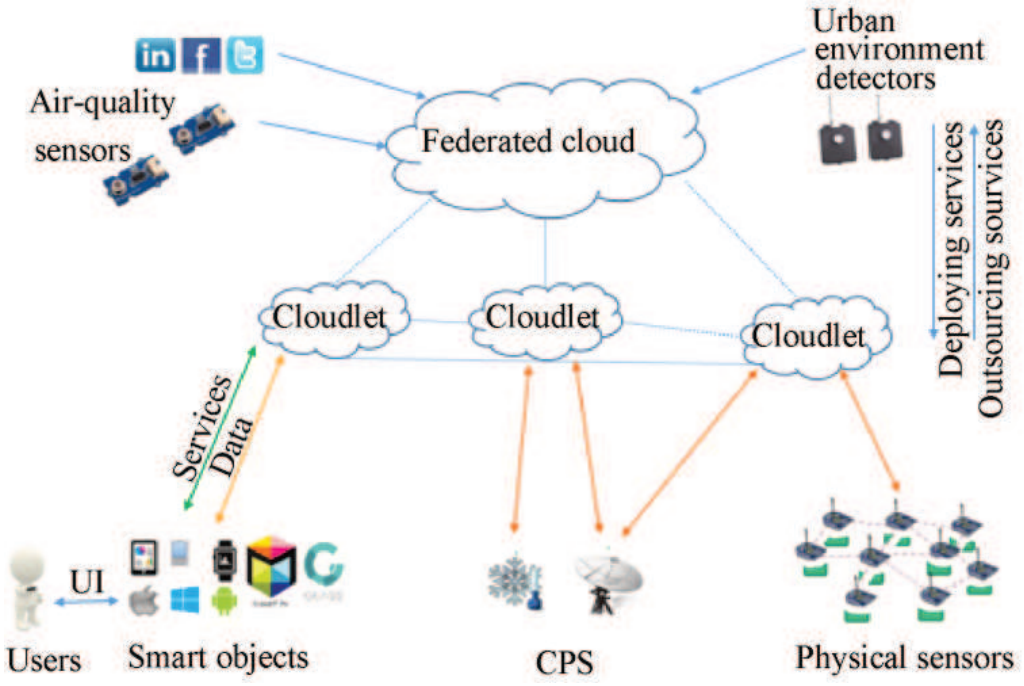
- ***Cloud***: define um nodo na camada mais alta, responsável por interagir com um número de *cloudlets*, possuindo uma melhor performance e com a possibilidade de ser rapidamente distribuído e integrado com pouco esforço;
- ***Cloudlets*** são definidos como uma hierarquia média do modelo, utilizando de menos memória, porém fornecendo menos performance que a *cloud*; sua função é de agir como uma *cloud* local para um determinado território, definido como sua área de influência, diminuindo assim o número de requisições a *cloud*;
- **Consumidores finais:** são considerados como consumidores finais os dispositivos que recebem informações das unidades de processamento como objetos inteligentes, descritos pelo autor como quaisquer tipos de interface do usuário que tenham a capacidade de receber comandos por

voz ou *touch pad,* retornando uma resposta ao sistema como, por exemplo, dispositivos móveis. É importante ressaltar que para os autores sistemas cibernéticos físicos são aqueles que possuem a capacidade de receber comandos, como sistema de alarmes;

- **Usuário:** pessoa que utiliza os objetos inteligentes em suas atividades.

O CLET considera um ambiente com diversos objetos inteligentes interconectados em uma cidade. A fim de atingir os requerimentos de tempo real nas respostas a partir de dispositivos móveis, com sua menor capacidade de processamento; optou-se por utilizar dos recursos dos *cloudlets* próximos. Caso nenhum desses *cloudlets* se encontre nas proximidades, o dispositivo em questão pode transitar para um estado em que requisita a *cloud*, ou no pior cenário, utilizar somente das informações e recursos próprios, provendo assim funcionalidades limitadas até a restauração de seu acesso aos recursos das camadas superiores (KOTEVSKA; LBATH; BOUZEFRANE, 2016). A Figura 9 apresenta a arquitetura do sistema, com a camada de baixo sendo composta de sensores, e usuários que utilizam de objetos inteligentes. Os usuários solicitam serviços e enviam dados para a camada intermediária, da *Cloudlet,* que por sua vez interage com a *cloud* (*federated cloud*), onde residem as unidades de processamento de maior desempenho, sistemas de gerenciamento de dados, entre outros serviços. A proposta dessa arquitetura traz consigo o aspecto de que o usuário final não precisa se preocupar a respeito dos diferentes ambientes a que ele se encontra conectado, para ele, só existe a concepção de serviço na nuvem; sendo somente relevante o fato de acessar os diversos serviços oferecidos sem perda de performance ou limitações de memória, e com um tempo de resposta satisfatório.

Figura 9: Arquitetura sistema através de tecnologia wireless



Fonte: (KOTEVSKA; LBATH; BOUZEFRANE, 2016).

Os modelos de predição do CLET lidam com informações provenientes de sensores físicos e sociais, onde usuários compartilham opiniões, experiências, eventos diariamente. As informações sociais do presente também são coletadas e analisadas em tempo real, a fim de detectar tendências significativas; dados de perfis de usuário, histórico de geolocalização, bem como aqueles gerados pelo módulo de descoberta de padrões são alguns dos dados coletados para definir os possíveis interesses do usuário. Dessa forma, quando um evento relevante ocorre dentro da área da cidade inteligente, os usuários serão notificados a respeito. A unidade de predição fica disponível na forma de serviços dentro das *cloudlets,* e utiliza dos dados mencionados anteriormente para predizer a próxima ação do usuário, exemplos seriam o caminho que ele utilizará em caso de engarrafamento, em que peça musical ele atenderá, entre outros.

Os autores propõem como estudo de caso o cenário de infraestrutura veicular dentro de uma cidade inteligente, em que veículos seriam equipados com sensores e atuadores capazes de coletar dados e atuar sobre componentes do veículo em tempo real, bem como ter acesso a *cloudlets* e a *cloud*. Os dados coletados forneceriam uma visão ampla da situação do trafego, podendo prover recomendações aos motoristas em relação ao tempo, rotas congestionadas, entre outros. O sistema de

cloud ainda seria capaz de unir informações de *cloudlets* distintas, de movimentação veicular e outros eventos, como acidentes e empregar métodos de aprendizagem, como reconhecimento de contextos (OTEBOLAKU; ANDRADE, 2016), para tornar o sistema mais adaptativo.

3.7 FrameTrail

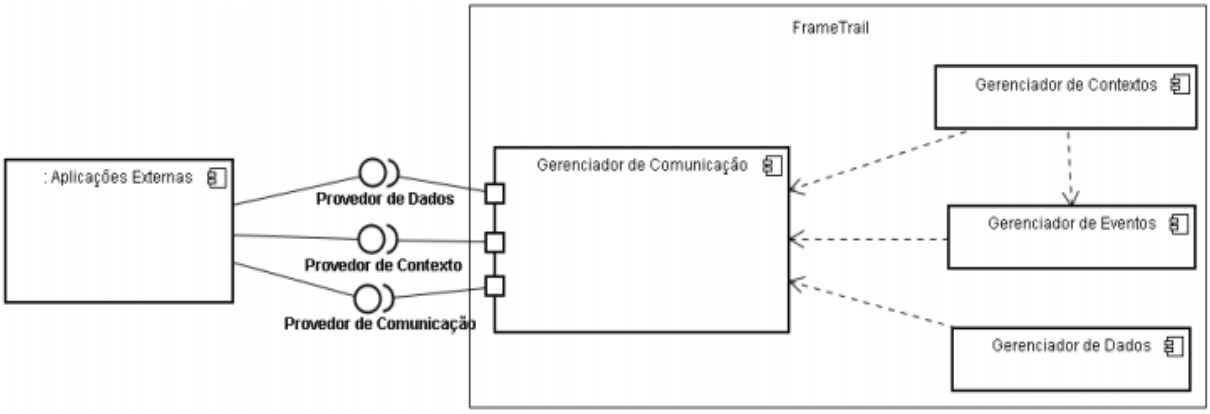
O trabalho propõe a criação de um *framework* que visa auxiliar no desenvolvimento de aplicações que empregam o conceito de trilhas (MARTINS, 2011). Para o autor, trilha é descrita como uma coleção de registros armazenados contendo informações de contextos visitados por uma entidade, bem como as atividades desenvolvidas por ela.

A proposta considera a manipulação de trilhas de forma genérica, lidando somente com os dados do ambiente, sendo que cada a cada entidade é atribuída uma trilha única. O autor teve como base para os modelos de trilha as ideias expostas pelo modelo UBITRAIL (SILVA, 2009), que se apresenta como um modelo para gerenciamento de trilhas em ambientes ubíquos.

O FrameTrail lida com o conceito de eventos, que são definidos pela aplicação tanto durante o desenvolvimento, quanto em tempo de execução, através da criação de *triggers*. Os mesmos podem ser consumidos por uma ou mais aplicações diferentes, no mesmo ou em distintos dispositivos. A cada evento são atreladas restrições, que são constantemente monitoradas pelo gerenciador de eventos, quando uma restrição é satisfeita, o gerenciador cria um novo evento e notifica os ouvintes registrados.

As aplicações registram os diversos sensores relevantes, que necessitam de um vínculo com algum tipo de hardware ou software. Através das leituras dos dados desses sensores o *framework* avalia as condições de restrições e gerencia os contextos.

Figura 10: Arquitetura do FrameTrail



Fonte: (MARTINS, 2011).

A Figura 10 revela a organização dos componentes do FrameTrail. Nessa arquitetura, as aplicações externas interagem através dos provedores externos, que são utilizados para se integrar com diversos tipos de dispositivos, em que os mesmos são responsáveis por traduzir os dados processos entre os dispositivos e o *framework*. Os provedores são subdivididos em três categorias:

- Provedores de contexto: permitem que as aplicações definam seus contextos conforme sua necessidade específica, configurando os dados de cada contexto, e ainda definindo provedores de dados e comunicação específicos a esses contextos.

- Provedores de dados: responsáveis por manter o repositório de dados, que armazena as informações sobre entidades e trilhas. Esses provedores por padrão são divididos em dois grupos, os que armazenam os dados em bases locais, no próprio dispositivo; e aqueles que as armazenam remotamente, necessitando de conectividade de rede para suas operações.

- Provedores de Comunicação: têm como objetivo integrar diferentes canais de comunicação, de forma independente a plataforma, realizando o intercâmbio de mensagens entre eles. Utilizando-se desse mecanismo, é possível interagir com aplicações que trabalham com tipos de mensagens distintas, já que essas são traduzidas pelo *framework* e posteriormente enviadas a seu destino no formato apropriado.

Além das funcionalidades dos provedores externos, o *framework* é dividido em quatro componentes internos. O Gerenciador de Eventos fica encarregado da criação de eventos previamente parametrizados. O Gerenciador de Contextos é responsável por detectar o contexto atual e fazer com que suas informações sejam acessíveis durante a execução da aplicação. O Gerenciador de Dados fica a cargo de manter os dados da trilha, executando a gravação e leitura de dados das trilhas através dos provedores externos de dados. O Gerenciador de Comunicação é responsável por integrar todos os componentes do *framework* e as aplicações desenvolvidas a partir dele.

3.8 Análise comparativa dos trabalhos selecionados

Nesta seção é feita uma comparação dos trabalhos relacionados em função das características consideradas relevantes dentro do escopo do trabalho. A Tabela 2 foi elaborada a fim de explicitar as diferentes contribuições de cada um dos trabalhos analisados frente a um conjunto de critérios. Abaixo são discutidos os critérios de comparação:

1. **Mapeia informações de contexto:** Este critério indica se o modelo mapeia informações de contexto sobre o cenário a qual afeta;

2. **Oferta funcionalidades de *framework*:** Define se o trabalho oferece uma forma de integração com aplicações já desenvolvidas, provendo um determinado conjunto de funcionalidades pré-estabelecidas, com o intuito de acelerar o desenvolvimento em pontos específicos do projeto;

3. **Mantém históricos de contextos:** Referente ao suporte a técnicas de armazenamento de informações de contexto, permitindo que seja mantido um histórico ao longo do tempo com as informações relevantes a determinadas entidades que compõe o projeto;

4. **Ambientes com entidades**: Caracteriza se o *framework* é capaz de ser utilizado em ambientes que agreguem características de entidades físicas e virtuais, e sua

capacidade de utilizar-se de dados provenientes de sensores e atuadores embutidos ou momentaneamente presentes no ambiente em dado momento, a fim de reagir automaticamente aos usuários presentes no mesmo recinto ou realizar atividades automáticas utilizando dos dados coletados através de aplicações externas que venham a exercer funções de gerencia de atividades específicas dentro do ambiente;

5. **Local para armazenamento dos dados:** Referente ao local onde são salvos os dados referentes aos contextos coletados pelo sistema. Esse critério identifica se os sistemas salvam dados em um servidor externo (remoto) ou se armazenam os dados em uma unidade local como, por exemplo, um dispositivo móvel;

6. **Estrutura de contextos com hierarquia:** Esse critério caracteriza se o *framework* permite que as informações de contexto coletadas sejam organizadas em um modelo hierárquico, que tenha como intuito trazer benefícios as aplicações que utilizarão do mesmo. Bem como se seus contextos apresentam características dinâmicas, ou seja, permita que o contexto para determinada entidade (sensores e atuadores) sejam modificados ao longo do tempo para assim melhor representar as condições reais de uso de ambientes inteligentes;

7. **Arquitetura do modelo:** Refere-se à arquitetura na qual os trabalhos analisados foram desenvolvidos, de acordo com as definições oferecidas por SCHOLLMEIER (2001) para os conceitos de *peer-to-peer.* Os que necessitam de uma estrutura mínima de servidores, tanto esses externos quanto servidores agindo em um dispositivo local foram designados cliente-servidor, e aqueles que não necessariamente necessitam de um servidor e que possam realizar suas atividades de comunicação apenas empregando uma rede de dispositivos que compartilham recursos entre si foram considerados *peer-to-peer*.

Tabela 2: Comparativo entre projetos selecionados

Critérios	CPA	MDA (Robos)	DTLC (Semáforo)	ezContext	CLET (*Cloudlet*)	FrameTrail
Mapeia informações de contexto	Sim	Sim	Sim	Sim	Sim	Sim
Oferta funcionalidades de *framework*	Sim	Não	Sim	Sim	Sim	Sim
Mantém histórico de contexto	Sim	Não	Não	Sim	Sim	Sim
Ambientes com entidades	Sim	Sim	Sim	Sim	Sim	Sim
Local para armazenamento de dados	Remoto	Não descrito	Não descrito	Não descrito	Remoto	Local/ Remoto
Estrutura de contextos com hierarquia	Não	Não	Não	Não	Não	Não
Arquitetura do modelo	Cliente-Servidor	Cliente-Servidor	*Peer-to-Peer*	Não descrita	Cliente-Servidor	Não descrita

Fonte: Elaborado pelo autor.

Quanto ao mapeamento de informações de contexto, todos os trabalhos analisados possuem algum tipo de mapeamento, visto que essa tarefa serve como base para o funcionamento das demais funcionalidades, porém poucos descrevem a forma específica como esse mapeamento é realizado. O CPA utiliza de contextos em três tipos distintos, contexto proveniente de dispositivos móveis, contexto de sensores externos, e o contexto de perfis de usuário. O MDA prove uma camada de abstração para lidar com dados vindos de sensores tanto com baixa quanto alta cadência de dados, e até mesmo robôs. Os sensores são divididos em tipos e expostos aos desenvolvedores na forma de serviços.

Quanto à oferta de funcionalidades de *framework*, somente o sistema MDA não oferta características atreladas a *frameworks*, visto que se apresenta como um *middleware* para facilitar o desenvolvimento de aplicações com componentes

distribuídos entre robôs, sensores e a nuvem. O trabalho DTLC, por sua vez, apresenta-se como um *framework* para otimização de fluxo de trafego em semáforos, e apresenta duas propostas distintas para atingir tal objetivo, testando a validade de seus modelos através de simulações.

Dentre os trabalhos selecionados, tanto o *middleware* MDA quanto o *framework* DTLC não oferecem mecanismos para registrar informações de contexto para uso posterior, portanto não proveem suporte a históricos de contextos.

Quanto ao quesito sobre ambientes com entidades reais e virtuais, todos os trabalhos podem ser utilizados dentro de um cenário com a presença dessas entidades. O MDA pode ser utilizado, por exemplo, na configuração de robôs e sensores para realizar tarefas específicas dentro de um ambiente inteligente, como a realocação de objetos. O DTLC seria o que possui a maior discrepância em relação a esse conceito, visto que sua aplicação abrange uma grande área (diversas interseções de uma região), mas considerando que fornece inteligência a ambientes, que reagem ao usuário, sua proposta ainda permanece válida.

A maioria dos trabalhos utiliza de métodos remotos para armazenar suas informações. O *middleware* MDA e o *framework* DTLC não mencionam formas de armazenamento remotas nem locais. Quanto ao MDA, o mesmo possui a capacidade de lidar com o processamento de informações operando em modo local, mas não informa se armazena qualquer tipo de dado sobre tal processamento. O *framework* ezContext cita que suas informações de contexto são armazenadas em memória, mas que a base de conhecimento é capaz de acessar qualquer tipo de base de dados externa. O FrameTrail não realiza uma limitação quanto ao tipo de armazenamento utilizado pela aplicação, podendo estes armazenarem os dados localmente ou em um servidor remoto.

Quanto à estrutura de contextos com hierarquia, o CPA não menciona nenhum tipo de modelo de hierarquia empregado na organização de seus contextos. Apenas utiliza os contextos para gerar inferências adicionais através de mecanismos como *pattern matching*, redes bayesianas e árvores de decisão. O MDA e o DTLC não demonstram uma forma específica que empregam para lidar com as informações de contexto coletadas de diferentes sensores, visto que não empregam de informações de histórico para realizar processamentos futuros em prol do sistema. O ezContext

utiliza-se de uma estrutura de pares de chave/valor para a estruturação de seus contextos, essa informação é encapsulada em conjunto com informações adicionais de configuração, como o tipo de entidade, o identificador do provedor, entre outras. Esses dados são utilizados para registrar um provedor e posteriormente processar os dados a partir da utilização de seu motor de regras. O *framework* não descreve nenhuma outra forma de organização a qual os contextos são submetidos, e não descreve os diferentes tipos de dados suportados dentro de sua estruturação. O CLET não menciona nenhum tipo de estrutura hierárquica que utiliza para seus contextos, apenas descreve que seus sensores sociais captam as informações e as analisam em tempo real para detectar tendências que considera significativas entre seus usuários. O FrameTrail não descreve nenhuma estrutura hierárquica para seus contextos, porém descreve que cada aplicação deve registrar e definir quais informações serão utilizadas como contexto.

Na questão de arquiteturas, a maioria dos trabalhos utiliza uma estrutura cliente-servidor para suas operações, centralizando dados e provendo uma forma de armazenar informações. Porém, o *framework* DTLC não específica a necessidade ou utilização de uma estrutura cliente servidor para suas aplicações, apenas define que sua utilização viria da comunicação entre uma infraestrutura distribuída entre os sensores e seus controladores, compartilhando recursos para atingir os requisitos do sistema; portanto se assemelhando a uma estrutura *peer-to-peer*. Possivelmente uma proposta de larga escala desse sistema implicaria na utilização de uma estrutura com um enfoque em um ou mais servidores, a fim de prover uma métrica de comunicação e sincronia entre diferentes pontos de intersecções em uma grande região de abrangência. O ezContext não específica o tipo de arquitetura utilizado em seu desenvolvimento ou se os aplicativos que o utilizam necessitam de uma arquitetura específica para usufruírem de suas funcionalidades. O FrameTrail não limita o seu utilizador acerca de um tipo de arquitetura específica, permitindo tanto a construção de uma estrutura centralizada quanto uma estrutura de comunicação entre as entidades.

Ao analisar as soluções propostas, nota-se que os trabalhos não apresentam mecanismos que busquem representar os contextos através de um modelo de hierarquia, para melhor refletir a dinamicidade dos ambientes mutáveis. As entidades hierárquicas e dinâmicas do CMFrame visam fornecer esses mecanismos, de forma

que permitir que entidades modifiquem sua hierarquia durante o ciclo de vida do sistema, bem como oferecer uma forma de armazenar informações de contexto de forma dinâmica sempre que necessário.

Através de seu uso, o *framework* tem como meta prover o acesso aos contextos de forma a diminuir o esforço e o número de etapas realizadas pelo desenvolvedor de aplicativos, bem como armazenar contextos genéricos de uma forma mais eficiente, através de suas entidades hierárquicas e dinâmicas, do ponto de vista de múltiplos sensores e atuadores.

3.9 Considerações sobre o capítulo

O capítulo apresentou uma breve descrição acerca dos trabalhos existentes que continham relação com o *framework* proposto, tendo como objetivo identificar aspectos relacionados entre os trabalhos pesquisados, bem como tendências de implementação empregadas no desenvolvimento dos mesmos. O próximo capítulo apresenta o *framework* CMFrame, demonstrando sua arquitetura e principais componentes.

4. FRAMEWORK CMFRAME

Este capítulo descreve o CMFrame, um *framework* que fornece suporte ao gerenciamento de históricos de contextos em ambientes compostos por sensores e atuadores, através do uso de entidades hierárquicas e dinâmicas. A seção tem início com uma visão geral acerca do funcionamento do *framework*. Na segunda seção são apresentados os requisitos. A terceira seção apresenta a arquitetura do *framework*. Por fim, a última seção apresenta as considerações finais sobre o capítulo.

4.1 Visão Geral

A área de atuação do CMFrame considera seu uso em ambientes compostos por sensores e atuadores, entre outros objetos que compõem o repertório IoT e que possam ser representados através de entidades reais e virtuais. As entidades hierárquicas e dinâmicas que o *framework* apresenta podem ser empregadas em conjunto com o conceito de entidades reais e virtuais, como uma forma de organizar os elementos de um ambiente em uma hierarquia, definindo a relação que cada entidade tem com as demais. Todos esses objetos encontram-se interligados em uma rede local, coletando dados constantemente e trocando informações entre si. Dessa forma, torna-se possível aplicar o CMFrame em casos que contemplem diversas configurações de ambientes inteligentes, bem como os aplicados a sistemas CPS, porém, neste último com algumas ressalvas, como a falta de mecanismos de tolerância a falhas e que garantam a alta confiabilidade do sistema frente a situações adversas, como as encontradas nesse tipo de sistema. Outro fator é a ausência de encriptação na transferência das mensagens entre os dispositivos que utilizam o CMFrame, dificultando sua utilização para determinadas aplicações que necessitem de maior segurança.

A seguir serão apresentados conceitos utilizados pelo *framework*, visando o melhor entendimento da forma como se relacionam com o mesmo, bem como aspectos relacionados à base de dados escolhida.

4.1.1 Sensores

Sensores são definidos como dispositivos que convertem um parâmetro físico, químico ou biológico em um sinal elétrico, e posteriormente entregue a um sistema de observação na forma de um dado (BERMUDEZ et al., 2009). Sua função é a de coletar medidas de um ambiente em que se encontra inserido, portanto monitorado. Um sensor genérico pode ser equipado com diferentes capacidades, permitindo que coletem diferentes atributos físicos, entre eles: luz, temperatura, umidade, pressão, velocidade, aceleração, aspectos acústicos, campos magnéticos, condutividade, radiação solar, entre outros. Os valores coletados por sensores são variados, abrangendo tanto indicadores de ligado e desligado, quanto valores em intervalos numéricos, como no caso da temperatura, e alcançando até mesmo tipos de dados como impressões digitais, som, fotos e vídeos (AUGUSTO et al, 2013).

4.1.2 Atuadores

Os atuadores são dispositivos, normalmente mecânicos, responsáveis por gerar um movimento sobre um determinado mecanismo ou sistema (TARAZÓN, 2015). Da mesma forma que os sensores, necessitam residir fisicamente no ambiente em que irão exercer sua função, geralmente próximos ao mecanismo que desejam atuar sobre. Os atuadores exercem sua função a partir de um sinal de controle, normalmente na forma de um sinal elétrico, de pressão, ou até mesmo surgir a partir da interação da força humana diretamente sobre sua estrutura (SEKHAR; UWIZEYE, 2012). Os atuadores podem ser divididos em atuadores pneumáticos, hidráulicos, elétricos, térmicos ou mecânicos. A necessidade de lidar com materiais frágeis, como a manipulação de órgãos por parte de robôs, trouxe a necessidade da criação de materiais macios e leves, o que trouxe à tona um novo tipo de atuadores, chamados de atuadores macios, capazes de produzir movimentos flexíveis a partir da deformação de seus materiais.

4.1.3 Contexto

Contexto para ambientes inteligentes, portanto, pertinente ao escopo deste trabalho, refere-se a toda informação passível de captação por um objeto inteligente (PETROULAKIS et al, 2013) dentro de um ambiente físico que seja considerada relevante para que um determinado aplicativo possa desempenhar suas funções. Dentro desse ambiente, o objeto inteligente pode representar tanto um objeto móvel, como no caso de um telefone celular de um usuário, quanto um objeto estático como no caso de um sensor que mede o nível de água presente em um reservatório. Em um cenário real, um ambiente contendo sensores e atuadores distribuídos em suas dependências, pode ser afetado simultaneamente por um ou mais aplicações que desejam realizar suas atividades com base nas informações coletadas. A captura dessas informações permite que aplicações analisem os diversos dados coletados a fim de captar a situação em que o ambiente se encontra, e assim reagirem de acordo com as necessidades do ambiente ou com a proposta do aplicativo.

No CMFrame, contextos encontram-se atrelados a uma entidade, e essa por sua vez obedece a uma hierarquia. Cada entidade é hierárquica e dinâmica, enquanto que os contextos atrelados a cada uma delas são dinâmicos, embora por razões distintas. Os contextos são considerados dinâmicos, pois no tempo X, podem ser completamente distintos (tanto na quantidade de valores, quanto a respeito de informações que contêm) daqueles enviados no tempo Y.

Um exemplo dessa dinamicidade pode ser visto na Tabela 3, que representa os dados vindos do sensor de umidade com um identificador fornecido pelo desenvolvedor, definido aqui como ID 2, para uma determinada entidade. Em dado momento, o microcontrolador (entidade) atrelado ao sensor de umidade pode realizar inferências adicionais e assim prover informações extras a base de dados, nesse caso foi calculado que o valor 320, para esse sensor, corresponde a 68% de umidade no ambiente. Essa adição de informações não traz detrimento algum aos valores anteriormente inseridos pela mesma entidade, e não determina que no futuro esse valor adicional se torne obrigatório.

Tabela 3: Exemplo de contextos dinâmicos

ID	Info	Porcentagem	Data	AmbienteAtual
2	320		2017-12-20T20:44:52	
2	320	68%	2017-12-20T20:45:52	
			2017-12-20T20:46:52	Sala

Fonte: Elaborado pelo autor

Além de informações de contexto adicionais, o uso desse fator dinâmico permite que a aplicação armazene dados completamente distintos uns dos outros, para a mesma entidade, sem a necessidade de uma reestruturação do *framework*. Só é necessário que exista um identificador conhecido pela aplicação para que essa informação seja acessível posteriormente, como pode ser visto no caso do campo *AmbienteAtual*, que descreve o ambiente na qual aquela entidade móvel se encontra fisicamente. Outra vantagem do fator dinâmico está na possibilidade de empregar sensores e atuadores que ainda não foram desenvolvidos no momento que este *framework* foi proposto, visto que ele não impõe uma limitação acerca de quais informações podem ser salvas.

Figura 11: Contextos dinâmicos em JSON

```
/* 1 */
{
   "ID" : 2,
   "Info" : 320,
   "Data" : "2017-12-20T20:44:52"
}

/* 2 */
{
   "ID" : 2,
   "Info" : 320,
   "Porcentagem" : "68%",
   "Data" : "2017-12-20T20:45:53"
}

/* 3 */
{
   "Data" : "2017-12-20T20:46:52",
   "AmbienteAtual" : "Sala"
}
```

Fonte: Elaborado pelo autor

Internamente, o *framework* lida com os contextos na forma de um arquivo JSON, permitindo assim a variação dos elementos utilizados a qualquer momento. A

Figura 11 mostra essa representação. Através dessa métrica, os valores adicionais, no caso desse exemplo, *Porcentagem e AmbienteAtual*, podem estar presentes em qualquer quantidade desejada, podem possuir qualquer nomenclatura, desde que não utilizada pelas demais funções do *framework* (parâmetros essenciais), e podem utilizar qualquer tipo de valores que possam ser codificados como um tipo numérico, ponto flutuante, *string*, date, entre outros. Se tratando de informações de contexto, o identificador (ID) exemplificado representa a qual entidade os dados pertencem, e por esse motivo pode se repetir ao longo do tempo.

4.1.4 Nodo

Dentre os tipos de dispositivos que agrupam dados de sensores, existem aqueles que trabalham como *Gateways*. Os Gateways são responsáveis por agregar os dados coletados a partir dos sensores genéricos e os transmitir a um determinado local que agregue esses dados, como uma base de dados em um servidor remoto. *Gateways* normalmente possuem uma maior capacidade de processamento, maior nível de bateria e maior capacidade de alcance de transmissão (YICK; MUKHERJEE; GHOSAL, 2008).

Dentro do escopo deste trabalho, um nodo (*No*) é definido como um *Gateway*, ou seja, um agregador de sensores em uma única plataforma, responsável por transmitir em determinado intervalo de tempo as diversas informações de seus sensores. Os atuadores presentes no ambiente também estarão conectados a um nodo. É importante ressaltar que em outros sistemas que utilizem do *framework,* esse *No* pode ser representado de outra forma, ou até mesmo encontrar-se subdividido entre outros componentes e sua implementação pode ou não possuir uma inteligência artificial para refinar o seu comportamento.

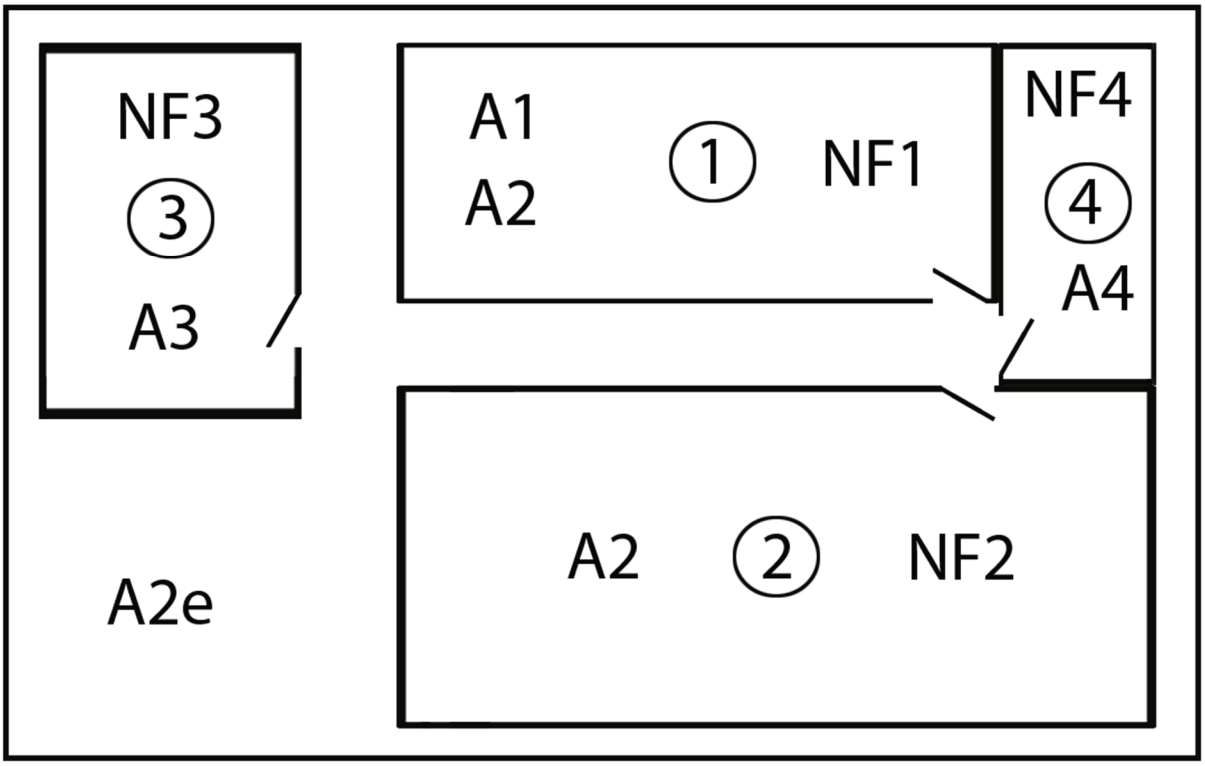
Os nodos descritos neste trabalho são divididos entre nodos físicos (*NoF*) e nodos virtuais (*NoV*). Os nodos físicos representam os gateways mencionados, e são unidades controladoras que agregam os sensores e atuadores de todo, ou de uma parte, do ambiente. Um dos exemplos de nodo físico é o Arduino uno[1] ou um

[1] https://www.arduino.cc/en/main/arduinoBoardUno

Raspberry Pi[2]. Os nodos físicos também são responsáveis por enviar os dados referentes aos sensores e atuadores conectados a ele, para que sejam armazenados na base de dados de contextos.

Quanto aos nodos virtuais, esses são representações em nível de software, portanto só existindo em um nível virtual, sem uma equivalência no ambiente físico. Um dos exemplos de seu funcionamento seria um software que fosse executado em uma máquina pertencente à mesma rede dos ambientes inteligentes monitorados ou em um dispositivo móvel e que realizasse o monitoramento de condições pertencentes a mais de um ambiente.

Figura 12: Divisão de nodos



Fonte: Elaborado pelo autor

A Figura 12 mostra um exemplo de nodo virtual, com uma possível subdivisão das diferentes salas do Mobilab, o Laboratório de Pesquisa em Computação Móvel da Universidade do Vale do Rio dos Sinos (UNISINOS), e seus nodos. Cada elemento AX representa um aplicativo em execução em sua respectiva sala, em conjunto com um nodo físico (NFX). Portanto, a sala 1 contém o aplicativo A1 e A2 em execução,

[2] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

ambos utilizando o nodo físico NF1. O A2e, por sua vez, seria um nodo virtual, externo aos ambientes no sentido de que não necessita permanecer fisicamente nos mesmos ambientes que controla, é somente uma representação virtual. Nesse exemplo, o A2e seria um aplicativo semelhante ao A2, porém com um nível de influência mais alta, visto que afeta todos os ambientes que possuem um aplicativo A2. A definição de nodos para este trabalho é apenas conceitual, os nodos, sejam físicos (*NoF*) quanto virtuais (*NoV*), são tratados internamente pelo *framework* como entidades, pois geram ou possuem a capacidade de gerar dados de contexto.

4.1.5 Entidades hierárquicas e dinâmicas

Dentro da estrutura do CMFrame, uma entidade representa um elemento que, com algum significado relevante ao sistema, pode ser tanto físico quanto de caráter completamente virtual, que possua uma função maior que apenas um sensor ou atuador individualmente posicionado em um ambiente. É obrigatório que a mesma gere dados a partir de um sensor ou conjunto de sensores, ou que modifique o sistema através de seus atuadores, tanto fisicamente no caso de um *NoF*, quanto virtualmente no caso de um *NoV.*

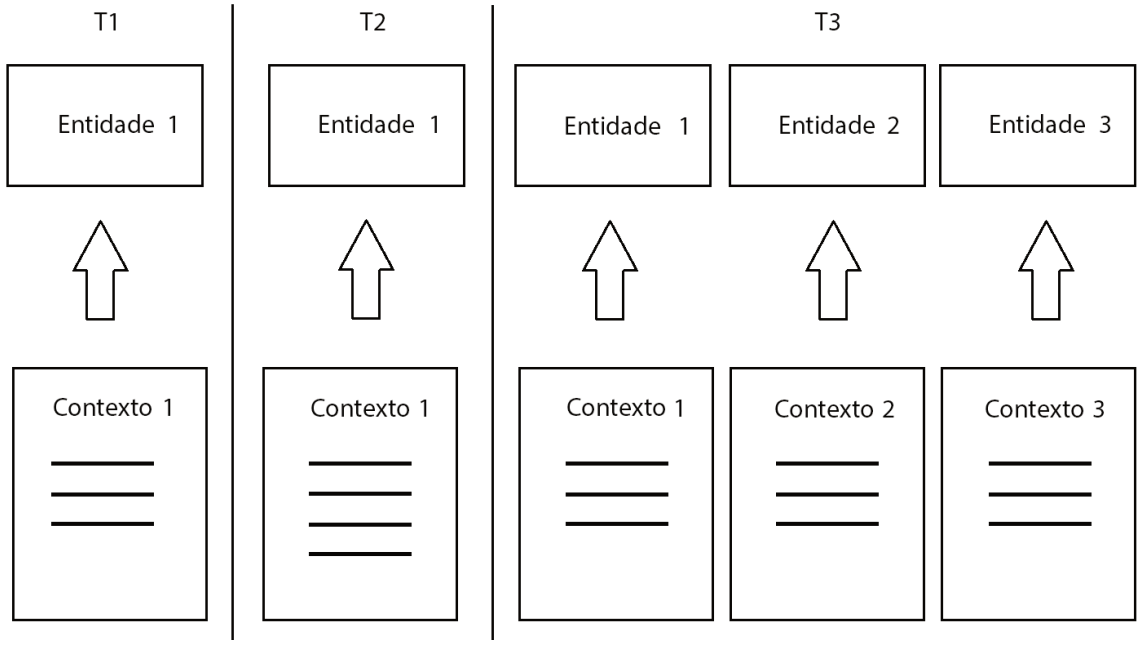
Uma entidade é capaz de representar diversos elementos de um ambiente, e até mesmo o próprio ambiente. Dentre alguns desses elementos encontram-se:

- Pessoa, representada através de seu dispositivo móvel;
- Um robô, móvel ou estático;
- Uma unidade processante e agregadora de sensores (*NoF*).
- Uma sala com diversos componentes inteligentes em suas dependências, com seus sensores e atuadores;
- Um andar, contendo diferentes salas (ambientes)
- Um prédio, que pode ser uma coleção de diversas salas, cada uma organizada em andares.

A cada entidade é atrelado um conjunto de informações de contexto. Para o funcionamento do CMFrame, é necessário que o tempo em que tais informações

foram geradas componha o contexto, visto que tal tempo é utilizado como referência para formar um histórico, conforme visto na Figura 13. O *framework* emprega como mecanismo de indexação de contexto as informações referentes ao tempo em que esses dados foram gerados e a que entidade eles pertencem. Na Figura 13, os diferentes tempos presentes para esse histórico são definidos como T1, T2 e T3. O contexto 1 encontra-se atrelado à entidade 1, e existe uma versão desse contexto para cada tempo em que ele é armazenado, podendo ou não conter informações distintas entre cada um dos tempos, portanto, a versão presente no T1 pode ser diferente do T2 e essa pode ou não ser distinta da encontrada em T3.

Figura 13: Organização de históricos



Fonte: Elaborado pelo autor

Para o CMFrame, as entidades são consideradas tanto entidades hierárquicas, quanto dinâmicas. Seu fator hierárquico designa que cada entidade se encontra presente em um nível diferente dentro de uma hierarquia definida, permitindo que as entidades sejam consideradas como individuais dentro dessa hierarquia (entidades base) ou que possuem outras entidades dentro de si (entidades *container*), assim gerando um agrupamento de entidades.

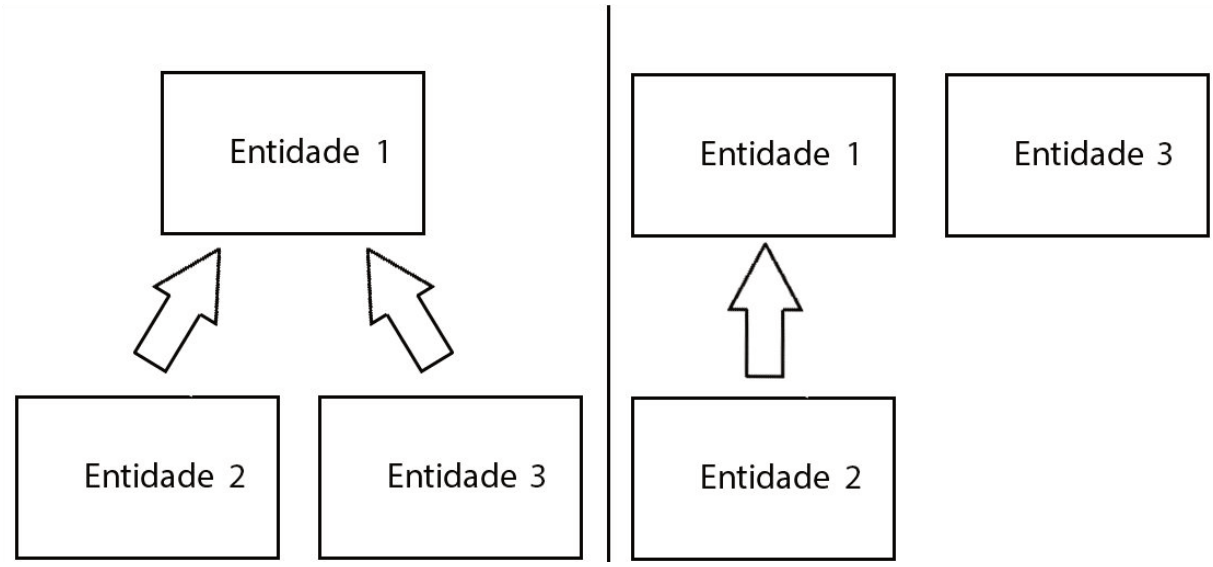
O fator dinâmico define que, durante a execução das aplicações que venham a usufruir do *framework*, a hierarquia pode sofrer modificações, mais especificamente,

as entidades podem mudar de nível hierárquico, caso fossem anteriormente individuais, podem aderir a um grupo de entidades, na forma de nodos filhos; podem se tornar entidades individuais caso fizessem parte de um grupo; ou ainda podem simplesmente mudar seu grupo. As trocas hierárquicas do CMFrame são realizadas a partir da solicitação da aplicação que o emprega, no momento que esta considera como oportuno, sendo definida para entidades individuais dentro do sistema, sejam elas *containers* ou *base*. O *framework* requer que sejam informados dois parâmetros apenas, o identificador único que representa a entidade cuja hierarquia será modificada e o identificador da entidade de destino, criando assim um vínculo em que a primeira entidade será designada como nodo filho da segunda. A hierarquia sempre é definida designando-se qual entidade (base ou *container*) de identificador X, pertencerá a qual outra entidade (base ou *container*), de identificador Y. O *framework* lida com a conversão entre entidades *bases* e *containers* automaticamente conforme as trocas hierárquicas são realizadas. Em geral, caso uma entidade *container* seja atribuída como nodo filho de outra entidade, ela permanece como *container*, e mantém a hierarquia de nodos filhos que possui. A relação de hierarquia só é removida em alguns casos específicos, como nos casos de referência circular (em que um nodo filho de uma entidade se torna seu nodo pai).

Considerando que os próprios contextos são dinâmicos, é possível que o Contexto 1 do tempo T1 guarde três informações, sendo elas X,Y e Z, enquanto que no Contexto 1 do tempo T2 sejam armazenadas quatro informações, os valores anteriores e mais o valor W, sendo que ambas refletem contextos provenientes da entidade 1, a qual se encontram atrelados, como visto na Figura 13.

A Figura 14 apresenta a distribuição de três entidades distintas dentro do CMFrame, em que a Entidade 1 representa uma sala física, a Entidade 2 um microcontrolador para um jardim inteligente (jardim que possui um controlador inteligente, que capta dados de sensores e pode ser acessado remotamente a fim de gerar atuações) e a Entidade 3 um robô móvel, com seus sensores e atuadores. Na parte esquerda da figura, é possível notar que a Entidade 2 e a Entidade 3 encontram-se atreladas à Entidade 1 dentro da hierarquia naquele momento do tempo. Na parte direita da figura, ocorre uma mudança hierárquica, em que a Entidade 3 (robô), anteriormente pertencente a Entidade 1 (sala), sai da sala, e, portanto, se desvincula da mesma, se transformando em uma entidade base.

Figura 14: Entidades Hierárquicas



Fonte: Elaborado pelo autor

Considerando que contextos estão atrelados a uma entidade, torna-se possível, a partir dessa distribuição hierárquica, que a sala inteligente (Entidade 1) contenha dentro de si os contextos pertencentes a outra entidade (Entidade 2), visto que a Entidade 2 é uma entidade base que se encontra contida dentro da entidade container (Entidade 1). Essa proposta de modificações hierárquicas auxilia na melhor representação de ambientes físicos que sofrem alterações em seus elementos ao longo do tempo, como no caso de fábricas altamente automatizadas. Essa estrutura hierárquica permite que aplicações que operam com uma quantidade de entidades fixa, possam navegar na hierarquia para buscar os contextos que se encontram atrelados em cadeia, a qualquer momento.

A utilização em larga escala das entidades hierárquicas e dinâmicas pode ser observada em um sistema CPS que lide com uma fábrica de manufatura, em que uma linha de montagem pode ser composta de *N* ambientes físicos, com diversas entidades presentes em cada um. Nesse cenário, existem as seguintes entidades:

- Três entidades que correspondem a salas, sendo que todas são consideradas entidades container, numeradas de 1 a 3. A sala 1 e 2 possuem linhas de montagem e são igualmente configuradas, enquanto que a sala 3 corresponde ao departamento de estoque;
- Uma entidade móvel base, representando um operador (pessoa);

- Quatro entidades presentes na sala 1, que representam robôs móveis de uma linha de montagem, e são classificados como entidades base;
- Quatro entidades robôs móveis presentes na sala 3, que se encontram desligadas.

Nesse caso, o operador é capaz de se mover livremente entre os três ambientes, e influir (atuar) sobre eles conforme necessário. Quando o operador se move da sala 2 para a sala 1, ocorre uma mudança de hierarquia, que é armazenada e que modifica a relação da entidade (operador) para que fique atrelado a nova entidade (sala). Essa mudança hierárquica do operador deve ser sempre requisitada por parte do aplicativo que utilize das funcionalidades do CMFrame, a critério do mesmo. Em seguida o operador decide modificar o modo de operação da linha de montagem, passando a produzir um novo tipo de produto. A partir desse momento, sua atuação (contexto) sobre a linha de montagem é armazenada, e sabe-se em que ambiente o mesmo atuou em determinado momento. Ao mesmo tempo, as quatro entidades robôs presentes na sala 1 são substituídos pelas quatro entidades (robôs) previamente armazenadas no estoque, visto que os primeiros não são capazes de produzir o novo tipo de produto, e assim são geradas oito mudanças de hierarquia, correspondendo as oito entidades robôs que mudaram para outra sala.

Caso o operador deseje consultar quais os robôs de determinado tipo se encontram em operação na fábrica, seriam realizadas mudanças hierárquicas que agrupariam os todos ambientes sob a mesma entidade (fábrica), permitindo que os contextos atrelados a cada uma fossem consultados.

4.1.6 Quanto à base de dados

O *framework* considera aspectos voltados ao contexto de IoT e seus objetos interconectados, trazendo à tona preocupações relacionadas com a geração de um grande conjunto de dados, bem como de sua grande variedade. Dentro da literatura existe um consenso em que Big Data é caracterizada como tendo três características primárias, denominadas de três "V'"; Volume, referente a quantidade de dados que são gerados por uma aplicação neste domínio; variedade: que diz respeito a grande variedade de formatos que esses dados apresentam e velocidade, em que os dados são gerados em intervalos de tempo muito curtos, dando uma ideia de continuidade

(BABICEANU; SEKER, 2016). Os requerimentos referentes a escalabilidade e disponibilidade em sistemas que armazenam dados de sensores resultaram em uma busca por bases de dados *NoSQL*, que são capazes de adicionar dinamicamente novos atributos aos seus registros, além de distribuir eficientemente os dados entre múltiplos servidores (CATTELL, 2011).

Considerando o grande volume de dados provenientes dos objetos de IoT*,* tem se como uma das alternativas para sua disponibilização o uso de um sistema de armazenamento de dados *NoSQL,* orientado a documentos, como é o caso do *MongoDB*, Cassandra[3], CouchDB, entre outros (FAZIO et al., 2015). Os autores (VEEN; WAAIJ; MEIJER, 2012) avaliaram as diferenças entre bancos de dados NoSQL e obtiveram a conclusão de que embora a base de dados Cassandra seja a melhor escolha para grandes aplicações críticas de sensores, o MongoDB torna-se a melhor escolha para pequenas ou médias aplicações de sensores em um contexto não-crítico, especialmente em casos com uma grande quantidade de escritas no banco de dados onde a performance de escrita torna-se importante, características estas que melhor representam um escopo de IoT para ambientes inteligentes no contexto deste trabalho.

MongoDB é uma base de dados *OpenSource* no modelo NoSQL que existe desde 2007, representando uma base de dados escalável e orientada a documentos. Nessa proposta as linhas tradicionais de uma tabela são substituídas por documentos, uma estrutura capaz de representar relações hierárquicas complexas ao permitir o armazenamento de *arrays* e até mesmo outros documentos embutidos em sua hierarquia (CHODOROW, 2013). Dessa forma, documentos armazenados em MongoDB podem ser mapeados para tipos de dados de linguagens de programação. Outra vantagem é que não existe uma predefinição quanto ao formato de um documento em suas chaves ou valores, permitindo que sejam adicionados campos conforme necessário. MongoDB armazena seus dados em um formato binário de um documento JSON, chamado BSON, que suporta booleanos, inteiros, valores de ponto flutuante, datas, *strings* e tipos binários. É um formato de dados leve e eficiente, podendo ser codificado e decodificado de forma rápida graças a sua utilização de tipos de dados em C.

[3] http://cassandra.apache.org/

Pelos motivos listados, o MongoDB foi a base de dados escolhida na implementação da porção do *framework* que lida com o armazenamento de dados. Tem-se como intuito criá-lo inicialmente de forma local, demonstrando o seu funcionamento como uma base de dados para propostas orientadas à internet das coisas. É importante ressaltar que o *framework* permite que sejam armazenadas informações diversas de contexto dos ambientes e, portanto, quaisquer ações feitas sobre um ambiente, como o processo de tomada de decisão, dependem exclusivamente de como a aplicação emprega os conjuntos de dados que seus sensores fornecem.

4.2 Requisitos do framework

Com o intuito de atender as especificações do *framework* proposto, são definidas as seguintes funcionalidades:

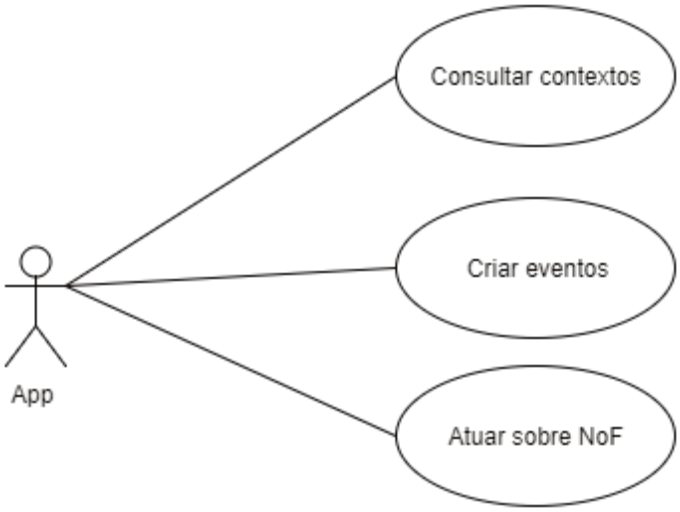
- Permitir a comunicação do *framework* com um banco de dados *NoSQL*, mais especificamente o banco de dados orientado a documentos, *MongoDB*;
- Armazenar informações de contextos provenientes de diversos sensores e atuadores, distribuídos entre ambientes físicos de configurações variáveis;
- Permitir que aplicações realizem a consulta por diferentes informações de contexto de um ambiente, organizadas na forma de históricos, na base de dados;
- Fornecer um mecanismo para a mudança de hierarquia de um determinado ambiente a partir de um aplicativo, em tempo de execução;
- Fornecer mecanismos de comunicação para os diferentes tipos de nodos presentes durante a execução do *framework* por parte de diferentes dispositivos;
- Realizar filtragens dos dados enviados pelos nodos, a fim de otimizar o método de escrita no banco e diminuir o volume de dados provenientes das entidades;
- Permitir o monitoramento constante de informações de contexto que sejam de interesse do aplicativo, através de um mecanismo de controle de limites;
- Permitir a consulta de informações pertinentes aos estados válidos possíveis para um determinado ambiente, a partir do instante de tempo solicitado pelo aplicativo;

- Realizar a comunicação de múltiplas aplicações simultaneamente para com uma instância do *framework* a fim de que possam operar sob o mesmo conjunto de dados.

4.2.1 Casos de Uso

A Figura 15 mostra o caso de uso de um aplicativo que incorpora as funcionalidades do CMFrame. Podem existir diversas aplicações presentes no mesmo ambiente, cada qual afetando um determinado número de sensores e atuadores através de seus respectivos nodos físicos.

Figura 15: Caso de Uso do Ator Aplicativo

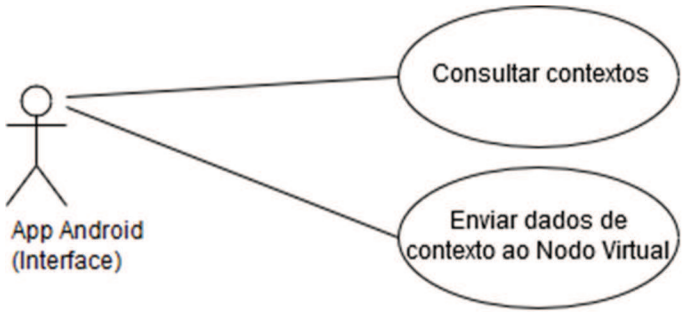


Fonte: Elaborado pelo autor

Dentro desse funcionamento, um aplicativo que utilize do CMFrame pode consultar os contextos armazenados previamente na base de dados, para que sejam utilizados pelo aplicativo em seu julgamento acerca das condições atuais e passadas de determinado ambiente. O *framework* ainda pode solicitar que sejam criados eventos a fim de monitorar algum contexto desejado. Outra funcionalidade se refere à comunicação por parte do *framework* para com um *NoF*, designando ordens de atuação com base nos atuadores que se encontram ligados a um determinado *NoF*.

A Figura 16 apresenta o caso de uso de um aplicativo *Android*[4] utilizando a interface desenvolvida para interagir com os demais elementos de um ambiente. Essa interface *Android* será capaz de consultar diferentes informações de contexto a partir da base principal, permitindo assim realizar o monitoramento acerca do ambiente a partir de um dispositivo móvel. Outra funcionalidade prevista é o envio de informações dos sensores presentes no dispositivo *Android* para um *NoV*, permitindo assim que o *NoV* utilize desses dados como uma camada extra de informações de contexto que pode ser acrescida a sua tomada de decisão acerca de um ambiente.

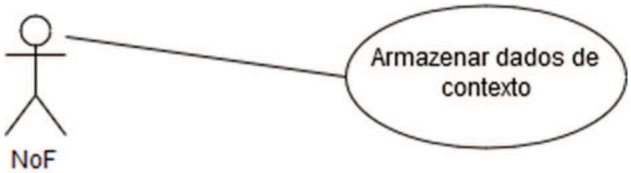
Figura 16: Caso de Uso do Ator Aplicativo Android



Fonte: Elaborado pelo autor

A Figura 17 mostra o caso de uso de um *NoF* inserido em um ambiente. Um ambiente pode ser composto por tantos *NoFs* quanto forem necessários para agrupar todos os sensores e atuadores presentes. Nesse caso o *NoF* seria responsável por agrupar todas as informações dos equipamentos conectados a ele e registrar esses dados de forma estruturada na base de dados de contextos.

Figura 17: Caso de Uso do Ator *NoF*



Fonte: Elaborado pelo autor

[4] https://www.android.com/

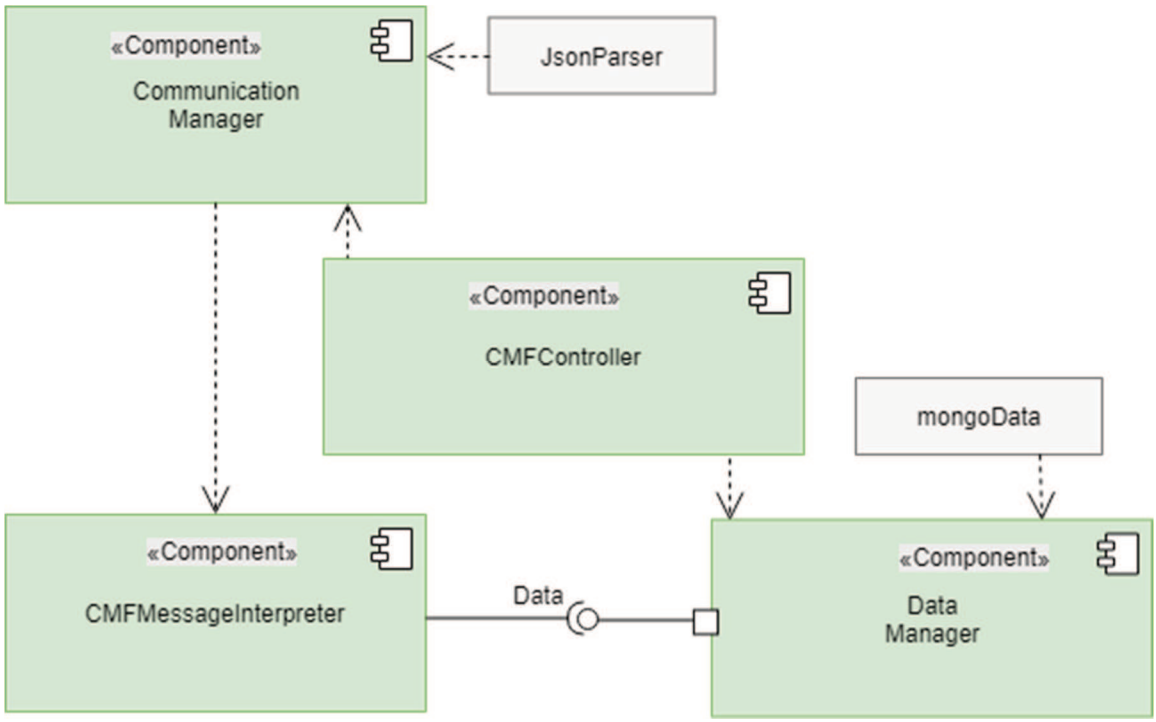
4.3 Arquitetura do framework

Essa seção descreve a arquitetura do CMFrame, em que os diferentes elementos do *framework* são representados na forma de componentes interligados através de interfaces. Optou-se pela separação da arquitetura do *framework* em duas partes, devido a questões técnicas e de forma a facilitar sua implementação.

Dessa forma, o CMFrame consiste em um conjunto de componentes que representam dois módulos. Para melhor representá-los, a parcela do *framework* que representa a conexão com a base de dados é denominada CMFDatabase, e o módulo que foi incorporado a todos os dispositivos que desejam se comunicar com a base de dados é chamado de módulo CMFrame, apesar que ambos fazem parte do *framework* e serem necessários para o seu funcionamento.

A Figura 18 mostra a arquitetura do módulo CMFDatabase. Dentre os principais componentes, destacam-se o *CFMController*, *DataManager*, *CommunicationManager,* e o *MessageInterpreter*. O *CFMController* é o responsável por integrar alguns dos comportamentos internos do *framework*, necessários ao seu funcionamento, bem como prover os métodos de acesso às funcionalidades presentes no mesmo. Nesse módulo, o *DataManager* é o componente responsável por estruturar os dados recebidos e realizar a integração com o banco de dados escolhido, além de traduzir os dados vindos do banco para um conjunto de informações que possa trafegar pela rede e ser interpretado pelo aplicativo utilizando o CMFrame.

Figura 18: Arquitetura do CMFDatabase



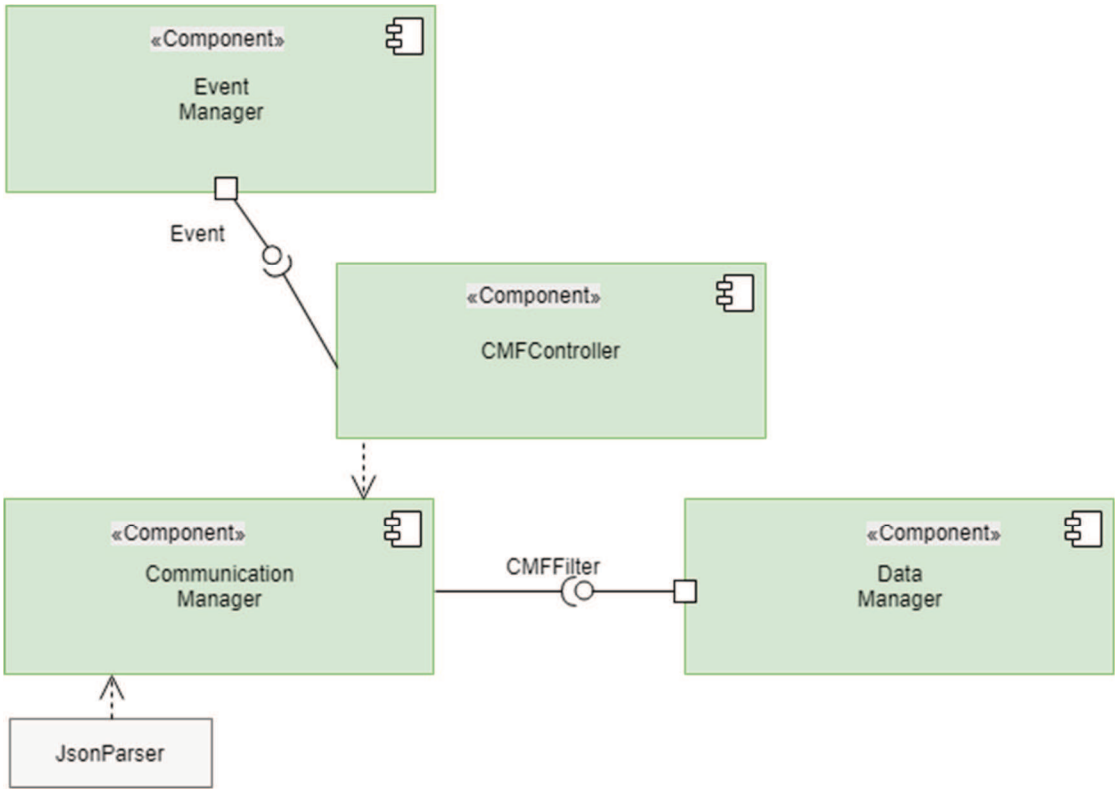
Fonte: Elaborado pelo autor.

O CMF*MessageInterpreter* realiza o tratamento de mensagens que são enviadas a esse módulo, contém, portanto, todo o comportamento sobre as ações que devem ser realizadas para cada tipo de mensagem recebida com base no protocolo do *framework*. Sua expansão é possível e necessária para a criação de novas mensagens, em padrões novos e que incluam outros comportamentos não originalmente previstos na implementação inicial do *framework*. O *CommunicationManager* realiza a comunicação externa ao *framework* como, por exemplo, a comunicação com o servidor de armazenamento, que poderia ser realizada através de métodos *REST (Representational State Transfer)*, que definem uma maneira uniforme de acesso a recursos de *web services*, através do protocolo HTTP. Esse componente também poderia ser implementado pelo usuário utilizando uma tecnologia de conexão qualquer, desde que respeite o protocolo de comunicação definido internamente no *framework*.

O principal reuso desse módulo pode ser encontrado no *MessageInterpreter*, em que o usuário pode modificar seu comportamento para prover novas funcionalidades em relação ao tratamento de mensagens, inclusive retornando informações para um propósito específico, diminuindo a quantidade de bytes transmitida pela rede. O *DataManager* também pode ser modificado a fim de prover uma implementação completamente diferente em relação ao banco de dados utilizado. O *CommunicationManager* também pode sofrer modificações a fim de suprir a implementação de um protocolo de comunicação distinto entre as instâncias do *framework*.

O outro módulo pode ser observado na Figura 19, que mostra a arquitetura do CMFrame, que é inserido junto ao código do aplicativo que faz uso de suas funcionalidades. Dentre os componentes presentes, pode-se mencionar o *CFMController*, *DataManager*, *CommunicationManager e o EventManager*. É possível notar que alguns dos componentes possuem nomenclaturas comuns entre os dois módulos do *framework*, porém internamente conservam características distintas atreladas ao funcionamento particular de cada uma.

Figura 19: Arquitetura do CMFrame



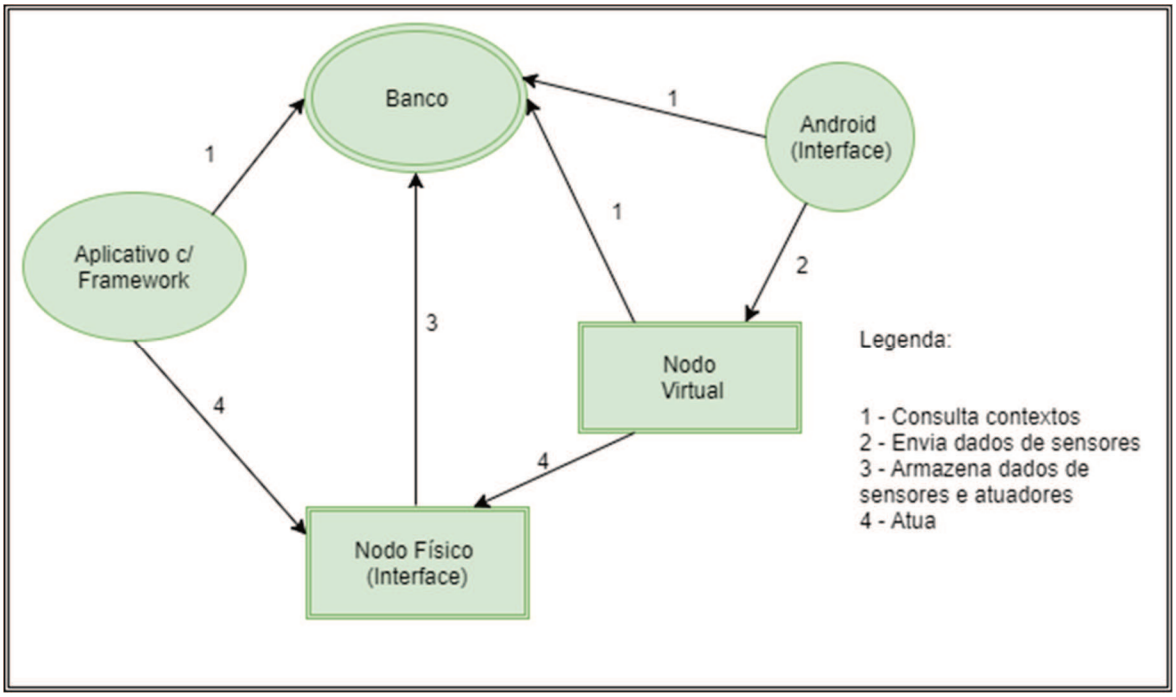
Fonte: Elaborado pelo autor.

O CMFController e o *CommunicationManager* desempenham a mesmas funções gerais dentro do módulo CMFrame do que as vistas no CMFDatabase, porém o *CommunicationManager* é mais desenvolvido nesse módulo, a fim de incorporar as diversas funcionalidades necessárias para a comunicação com o módulo da base de dados e inversão de fluxo no aplicativo. O *DataManager* por sua vez não possui a integração direta com qualquer tipo de banco de dados, mas desempenha outras funções de conversão de estruturas e criação de filtros para o *framework*. O *EventManager*, por sua vez, é responsável pelas atividades relacionadas aos eventos suportados pelo *framework,* como os mecanismos de monitoramento de variáveis de contextos relevantes para uma determinada aplicação e aqueles relacionados ao envio de mensagens do sistema que retornem informações ao aplicativo.

Dentro do módulo CMFrame, torna-se do interesse do aplicativo criar modificações dentro do *EventManager,* provendo novos eventos fora do escopo inicial desse trabalho. A expansão do *DataManager* permite a criação e modificação dos filtros básicos para buscas na base de dados. Em relação ao *CommunicationManager* é possível a criação de novas estruturas de envio de mensagens do sistema, expandindo a funcionalidade do *framework*, bem como a implementação de outros protocolos de transmissão de dados.

A Figura 20 mostra uma distribuição de nodos físicos e nodos virtuais, a fim de ilustrar a interação dos diferentes elementos que podem compor um ambiente físico estruturado através do CMFrame. Em um conjunto de salas, por exemplo, pode-se ter várias aplicações que utilizam do CMFrame para gerenciar seus contextos, todos consultando dados de contexto armazenados como histórico, a partir do banco de dados. Dessa forma, torna-se possível atuar sobre os atuadores conectados a um determinado nodo físico, através da comunicação com a interface presente no *NoF*. O *NoF* é responsável por agrupar e enviar para armazenamento no banco de dados as informações de contexto dos sensores e atuadores conectados a ele, bem como o instante de tempo que essa informação foi coletada.

Figura 20: Interação de nodos em ambientes



Fonte: Elaborado pelo autor.

O *NoV,* por sua vez, seria capaz de consultar dados de contextos do banco de dados e, baseado na lógica do aplicativo que o incorpora, realizar inferências para assim atuar sobre os *NoFs* de que tem acesso. A interface para dispositivos Android definirá um padrão de comunicação, que entre outras coisas permitirá que os dados de seus sensores sejam enviados ao *NoV* em um padrão que o mesmo possa interpretar, assim agregando dados para um possível refinamento na lógica de inferência por parte do *NoV*. Essa interface ainda será capaz de se conectar a base de dados para buscar informações de contextos dos ambientes a que tem acesso.

O CMFrame armazena as informações de contexto tanto dos sensores quanto dos atuadores, visto que a informação sobre o estado dos atuadores em um ambiente, em certo período no tempo, é de suma importância para aplicações que desejam descobrir o último estado válido para um determinado ambiente, e pode ser aplicada a sistemas CPS, onde sempre é necessário ter uma configuração válida para que o sistema possa retornar.

O processo de armazenamento tem início a partir de um nodo físico. A partir de um processo inicial de configuração, como a classificação do *NoF* como representando de uma entidade estática ou móvel, bem como sua informação de

contexto de localização. Essa localização pode ser fornecida a partir de um dispositivo GPS ou inserida manualmente com base em um valor de interesse do usuário. Uma entidade estática classificaria aquele *NoF* como elemento que permanecerá sempre no mesmo ambiente, enquanto que móvel definiria que ele pode ser um controlador atrelado a um robô, por exemplo, caso este tenha sensores e atuadores que contribuam para o ambiente. Caso o dispositivo seja móvel, os valores de sua localização atual devem ser enviados em conjunto com as demais informações de contexto, caso contrário serão omitidos a fim de economizar na transmissão de dados. Posterior a essa etapa, o utilizador pode fazer envios de dados de contexto na cadência que for apropriada.

Os dados mínimos de contexto podem ser vistos no exemplo abaixo, onde há um Identificador único para o dono daquela informação em particular, o campo Info que descreve o dado do sensor ou atuador e as informações referentes ao momento que aquele dado foi coletado, para auxiliar no histórico.

```
{
        [{ ID: 1, Info: 55, DateCollected: "2017-12-10T12:21:48.0862191-02:00"}]
}
```

Internamente, o *framework* lida com as informações de contexto como um JSON[5], o que permite maior flexibilidade de formatos de dados. Cada sensor ou atuador pode fornecer qualquer número de parâmetros adicionais, e não é necessário seguir o mesmo padrão entre envios. É possível, por exemplo, enviar um dado adicional a um sensor de temperatura em dado momento, e no próximo envio omiti-lo completamente, tomando proveito então do fator dinâmico de contextos. Esse fator torna o armazenamento de dados de contexto muito interessante, visto que se desejado, o *NoF* pode ser configurado de forma a prover um conjunto adicional de informações a sensores e atuadores, e ao mesmo tempo não limita o utilizador quanto ao conjunto inicial de informações necessário ao funcionamento do *framework*. Dessa forma é possível que sejam adicionados quaisquer tipos de sensores e atuadores desenvolvidos no futuro, mediante a tradução dessa informação dentro do *NoF*

[5] https://www.json.org/

previamente ao envio dos dados, sendo limitado apenas pela capacidade de envio da rede a qual o *framework* se encontra em operação.

O banco de dados *NoSQL* utiliza de coleções no lugar de uma tabela de um banco de dados relacional, portanto, é criado uma coleção de dados para cada *NoF* existente em um ambiente, onde a informação de contexto total de um ambiente é a representação de todas as coleções de *NoFs*, *NoVs*, e demais entidades que compõe um ambiente. Uma coleção diferente fica responsável por armazenar quais entidades se encontram inseridas em qual ambiente, sendo que entidades móveis podem pertencer a mais de um tipo de ambiente simultaneamente.

4.4 Considerações sobre o capítulo

Neste capítulo foram apresentados os diferentes elementos que compõem os módulos presentes no *framework* CMFrame, bem como seus requisitos e sua arquitetura. Uma versão inicial do *framework* foi desenvolvida a fim de validar a arquitetura proposta, e assim determinar se o mesmo atende às necessidades para qual foi projetado. No próximo capítulo serão apresentados os aspectos de implementação da versão inicial, bem como o cenário proposto para a aplicação do mesmo.

5. ASPECTOS DE IMPLEMENTAÇÃO E AVALIAÇÃO

Este capítulo discute os aspectos de implementação do CMFrame, bem como os aspectos de avaliação que foram aplicados. Na primeira seção são definidos os aspectos pertinentes ao desenvolvimento do *framework*, que foi submetido à avaliação. A segunda seção descreve a metodologia que foi empregada na avaliação, apresentando o cenário a qual ele foi submetido a fim de revelar informações sobre sua funcionalidade quando aplicado a atividades de gerenciamento de históricos de contextos utilizando de suas entidades hierárquicas e dinâmicas. Para o processo de avaliação, foram criadas duas aplicações, AmbientMonitor e EnergyMonitor, que são descritos na seção três e quatro. A quinta seção descreve a avaliação do *framework* através das aplicações. Por fim, a última seção apresenta as considerações finais sobre o capítulo.

5.1 Implementação do framework

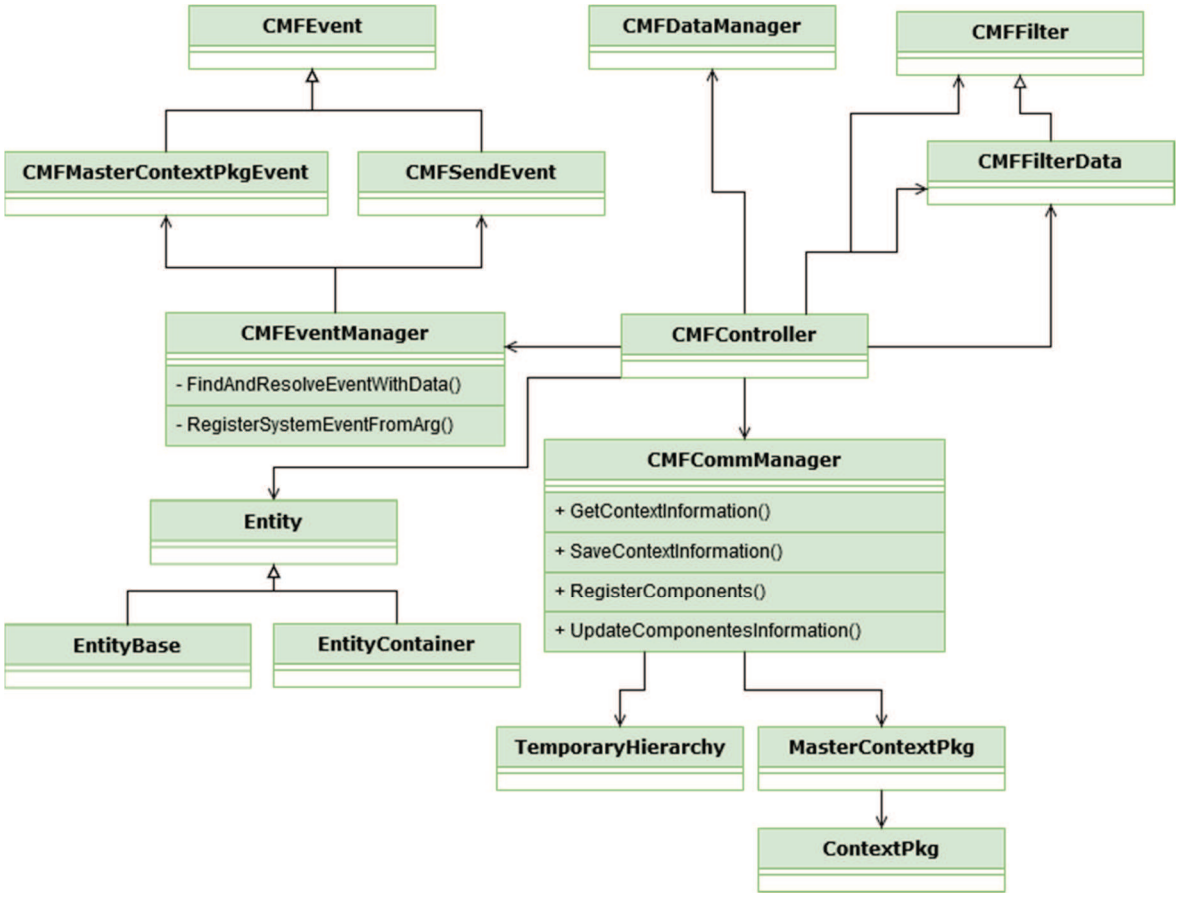
O CMFrame foi desenvolvimento na linguagem C#[6]. A partir do desenvolvimento do *framework*, espera-se prover um conjunto de funcionalidades que permitam a manipulação de contextos registrados em uma base de dados *NoSQL,* por parte de aplicações diversas que venham a influenciar um determinado ambiente ou conjunto de ambientes, levando em consideração na sua construção as características presentes em ambientes compostos por múltiplos objetos de Internet das coisas. Ao término de sua implementação, o *framework* foi submetido a um cenário de testes a fim de validar seu funcionamento sob o ponto de vista do gerenciamento de históricos de contextos.

A implementação inicial levou em consideração a distribuição do *framework* em dois módulos, denominados CMFrame e CMFDatabase. O CMFrame é o módulo responsável por realizar a comunicação entre o aplicativo desenvolvido e o módulo CMFDatabase. Todos os mecanismos de acesso aos dados de contexto, bem como

[6] https://docs.microsoft.com/en-us/dotnet/csharp/csharp

as funcionalidades de contextos hierárquicos e dinâmicos são acessíveis através desse módulo.

Figura 21: Diagrama de classe do CMFrame



A Figura 21 apresenta o diagrama de classe do módulo CMFrame. Toda a comunicação entre os módulos é realizada através do componente de comunicação, denominado *CommunicationManager*, que possui uma implementação do protocolo MQTT[7]. A classe *CMFDataManager* contém validações dos tipos recebidos do módulo CMFrame. *CMFEventManager* fica a cargo de gerenciar o ciclo de vida dos eventos criados pela instância do *framework,* trabalhando em conjunto com os diferentes tipos de eventos existentes, como *CMFEvent,* a qual todos os eventos são derivados, *CMFMasterContextPkgEvent*, para a tradução das mensagens de contexto

[7] MQTT (Message Queue Telemetry Transport) é um protocolo de troca de mensagens baseado no método *publish*/*subscribe*, concebido com o objetivo de fornecer baixa latência, com baixo tráfego de rede para situações onde a disponibilidade de largura de banda é baixa ou limitada, como no caso da comunicação de sensores.

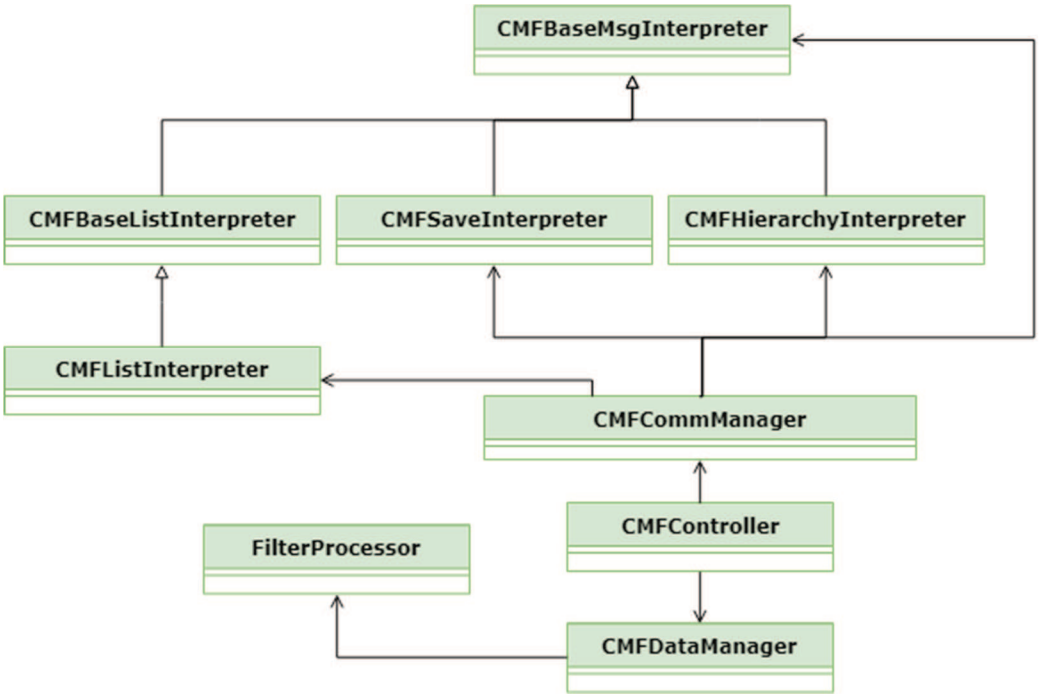
hierárquico recebidas do módulo CMFDatabase, e *CMFSendEvent,* que define o evento para as requisições que necessitam de um retorno ao CMFrame.

As classes *CMFFilter* e *CMFFilterData* correspondem a filtros utilizados nas requisições por contextos, configurando seu comportamento conforme as necessidades do usuário. As requisições por dados de contexto podem ser feitas para uma entidade, especificando um ID, um filtro que define quais dados da entidade se deseja adquirir, e uma função a ser retornada com os valores de contexto coletados. A busca por contextos em hierarquia, por sua vez, também segue a mesma parametrização, porém nesse caso toda a hierarquia é percorrida a partir da entidade desejada, agregando os dados pertinentes com base no filtro definido pela aplicação. Também é possível utilizar de suas funcionalidades de armazenamento de contexto sem a utilização do CMFrame, desde que o protocolo de comunicação do *framework* seja obedecido e se faça uso de uma implementação do protocolo MQTT.

O desenvolvedor utiliza classes que representam as *EntityBase* e *EntityContainer* para modelar o comportamento que suas entidades terão na aplicação. Cada uma das entidades é inicializada e configurada com informações básicas, que dizem respeito aos sensores e atuadores acoplados a si, dentro do ambiente. Durante sua execução, o *framework* percorre cada uma das entidades previamente registradas, e executa o método responsável pelo comportamento daquela entidade. Esse comportamento é fornecido pelo desenvolvedor, e diz respeito as atividades que a aplicação realiza com base nos dados contexto recebidos para a entidade em questão. De preferência, essas classes devem ser especializadas, provendo assim um comportamento customizado de acordo com as necessidades do desenvolvedor para aquela entidade específica, e permitindo o reuso de código ao desenvolver entidades com características semelhantes.

O CMFDatabase, por sua vez, fica responsável pelas conexões com o banco de dados escolhido, disponibilizando suas funcionalidades as demais instâncias do *framework* que executam o módulo CMFrame. A Figura 22 mostra o diagrama de classe do CMFDatabase.

Figura 22: Diagrama de classe do CMFDatabase



Fonte: Elaborado pelo autor

As classes responsáveis por interpretar as mensagens vindas para o CMFDatabase estão relacionadas ao *CMFBaseMsgInterpreter*, portanto, primordialmente, seria necessário ao utilizador do *framework* expandir suas funcionalidades a fim de prover um novo comportamento, caso desejado. Suas derivações padrões incluem mecanismos que interpretam diferentes tipos de mensagens internas para o *framework*, como aqueles que lidam com pedidos de armazenamento de dados, mudança de hierarquia, busca por contextos, entre outros. Caso a modificação das mensagens incluam novos padrões não definidos internamente, torna-se necessário a expansão do *CommunicationManager* presente no módulo CMFrame, refletindo as novas condições desejadas. Toda a comunicação com o banco de dados do *framework* é realizada a partir da classe *CMFDataManager.* A classe *FilterProcessor* governa a geração do filtro de pesquisa utilizado para todas as consultas a base de dados, dentro do *DataManager,* a partir de uma requisição recebida através da classe de comunicação *CMFCommManager*. O envio das mensagens de retorno é iniciado pelos interpretadores de mensagem, e é entregue ao *CMFCommManager* para concluir seu envio até a instância do CMFrame apropriada.

Internamente o CMFrame emprega um sistema de mensagens que converte a estruturação interna das classes para o equivalente em uma *string*, posteriormente codificando-as em um array de bytes para a transmissão pela rede. O *framework* utiliza *flags* em texto dentro da *string* de transmissão, cada *flag* (conforme Apêndice A) tem um propósito específico e designa um tipo de solicitação distinta entre os módulos do *framework*.

A fim de prover as funcionalidades condizentes a sua capacidade como um gerenciador de históricos de contextos, o *framework* possui uma implementação básica tanto do protocolo de comunicação, quanto da interação com o banco de dados, bem como um conjunto de métodos e funções distribuídas entre seus dois módulos, que garantem seu funcionamento para ambientes inteligentes mais simples, como aqueles encontrados em casas inteligentes, ou salas inteligentes, sem a necessidade de efetivamente expandir suas funcionalidades e classes em um primeiro momento caso o comportamento padrão do mesmo seja satisfatório ao propósito almejado. Isso torna o CMFrame uma solução completa no que diz respeito as funcionalidades iniciais que já se encontram incluídas em seu código, algumas delas sendo: armazenamento de contextos com base nas entidades, organização de contextos conforme a estrutura das entidades hierárquicas e dinâmicas, consulta das informações de contexto de entidades, tanto para uma entidade individual quanto em cadeia hierárquica, bem como os mecanismos necessários para a mudança hierárquica mediante solicitação do utilizador. Considerando que o código do *framework* encontra-se incluído na aplicação, no caso do módulo CMFrame, não existe a necessidade de expansão somente através de subclasses, podendo ser realizada diretamente no código-fonte caso desejado.

5.2 Metodologia de avaliação

A etapa de avaliação tem como intuito verificar as contribuições e limitações do desenvolvimento de um *framework* que suporte a implementação de aplicações que se utilizem de históricos de contextos para a realização de inferências, e que empreguem o conceito de entidades hierárquicas e dinâmicas conforme apresentados no CMFrame. Para tanto, foi realizado o desenvolvimento de um cenário englobando

diferentes ambientes, em uma representação de uma casa inteligente. Posteriormente foi submetido à avaliação através do desenvolvimento de duas aplicações distintas que fazem uso das funcionalidades providas pelo *framework* no quesito gerenciamento de históricos de contextos, empregando suas características particulares de entidades hierárquicas e dinâmicas, e averiguando as contribuições e limitações de seu uso em sistemas sensíveis a contexto. Na seção seguinte será descrito o cenário proposto para a execução da avaliação.

5.2.1 Cenário de uma Casa Inteligente

O cenário proposto visa representar uma casa inteligente, que engloba três ambientes distintos, representando a sala, a cozinha e o quarto de uma casa inteligente. Esse cenário possui dois usuários, X e Y**,** que podem se mover livremente entre qualquer um dos ambientes. A sala contém dois computadores, bem como um jardim inteligente. A cozinha conta com o monitoramento dos produtos de sua geladeira inteligente. O quarto possui um controle de som ambiente, alarme inteligente e um controle para a regulagem das persianas. Todos os ambientes possuem um sistema de iluminação inteligente, bem como sensores para medir o consumo energético dos dispositivos em uso.

Cada um dos usuários possui um cartão inteligente virtual, que se encontra em um aplicativo em seu smartphone, e possui um identificador único para o usuário. Vinculado a esse identificador se encontram informações de contexto armazenadas na base de dados do *framework,* representando um perfil próprio para aquele usuário. O usuário poderá aproximar seu smartphone dos leitores próximos aos computadores para regular suas configurações com as predefinições do perfil. Esse contexto contém informações referentes ao brilho da tela ideal, tamanho da fonte (para computadores), volume das caixas de som, se o mesmo necessita da leitura em voz alta dos elementos que compõem a tela, o nível de luminosidade ideal para o ambiente e a temperatura preferencial do ar condicionado. Esse aplicativo de celular é distinto das duas aplicações utilizadas para o processo de avaliação

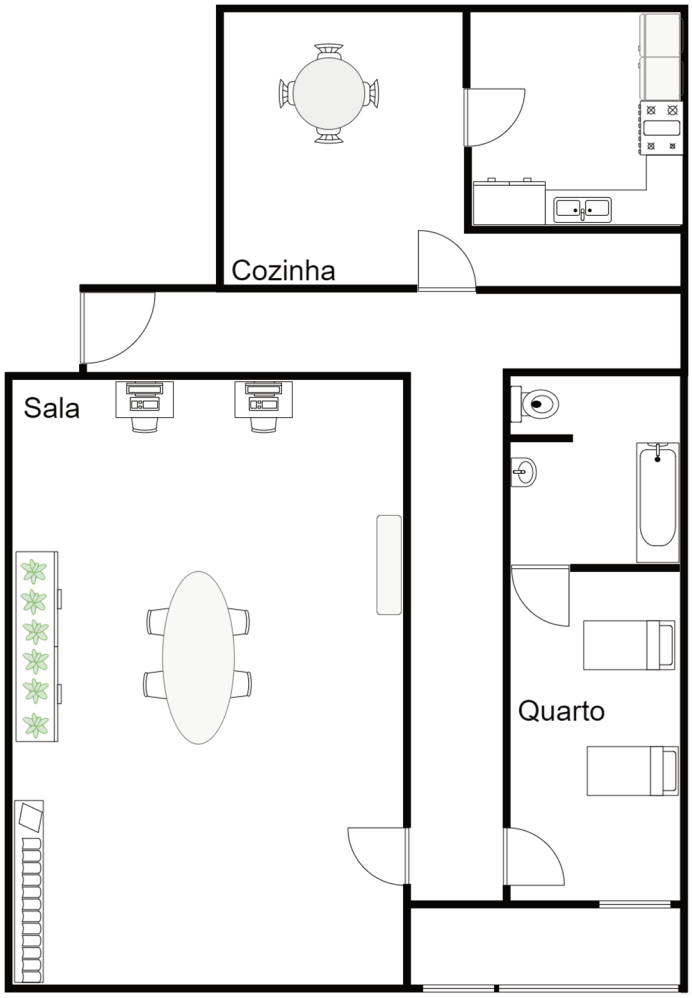
Como um dos elementos utilizados para avaliar a funcionalidade de entidades hierárquicas e dinâmicas do CMFrame, será empregado o uso de entidades móveis,

representadas pelos usuários X e Y, e por um robô de limpeza, que reside no corredor, e pode se mover livremente entre qualquer um dos três ambientes para realizar a limpeza. Periodicamente, o mesmo irá mudar de ambiente para desempenhar sua função.

A partir das definições desse cenário, são dadas as seguintes interações: o usuário X, munido de seu smartphone, entra na casa inteligente. Chegando à sala, abre o aplicativo do cartão virtual. O aplicativo se comunica e registra o usuário como presente na sala, com base no sistema de localização. As lâmpadas presentes na sala e o ar condicionado são acionados automaticamente, conforme a configuração no perfil do usuário. X se dirige a um dos computadores disponíveis, ligando-o, e aproxima seu smartphone do leitor ao lado do computador, para assim configura-lo de acordo com as preferências armazenadas no perfil do usuário. O usuário Y chega uma hora depois e se dirige a cozinha, abre seu aplicativo, que registra sua presença nesse ambiente e aciona as lâmpadas automaticamente, com base nas definições do perfil. Y se dirige a geladeira e retira alguns produtos para preparar um jantar. Enquanto isso o robô de limpeza se dirige à sala. O controlador responsável pelo jardim inteligente da sala detecta a necessidade de irrigação com base na análise das condições do solo, acionando assim o motor apropriado.

O usuário X interrompe o uso do computador da sala, o deixa ligado e se dirige a cozinha para jantar, ao término, se dirige ao quarto. Ao término do jantar, Y se dirige à sala, liga o ar condicionado (que foi desligado) e se conecta ao outro computador, aproximando seu smartphone do leitor. Quando o usuário X entra no quarto, o aplicativo de seu celular registra sua presença, e com base nas predefinições de seu perfil, existe a configuração de iluminação, som ambiente, o momento em que o alarme será acionado de manhã, bem como a definição sobre a abertura automática do sistema de persianas. Nesse momento o robô de limpeza vai à cozinha para limpá-la. O usuário X opta por dormir, enquanto Y sai da sala e decide lavar a louça da cozinha. Posteriormente, Y se dirige ao quarto, o aplicativo de celular registra sua presença, e com base nas definições do perfil, não interfere nas configurações do ambiente visto que já foram definidas pelo usuário anterior. Por fim, Y se dirige a sua cama para dormir.

Figura 23: Cenário Casa Inteligente



Fonte: Elaborado pelo autor

A Figura 23 apresenta a disposição de cada ambiente para o cenário proposto, e os identifica como referência para a alocação de recursos utilizados nesse cenário.

Tendo em vista o cenário proposto, apresenta-se a estruturação de sensores e atuadores disponíveis nos três ambientes, considerando sua organização em entidades do CMFrame.

Entidades Móveis

- Usuários (Representados através de seus cartões inteligentes);
  - X (Entidade ID – 4);
  - Y (Entidade ID – 5).
- Robô de limpeza (Entidade ID – 6).

Sala (Entidade ID 1)

- Entidade (ID 7) – Microcontrolador;
    - Sensores
        - Temperatura – (ID 1);
        - Umidade – (ID 2);
        - Luminosidade – (ID 3);
        - Medidor de consumo energético – (ID 4).
        - Detector de Presença – (ID 5).
    - Atuadores
        - Controle para regular o ar condicionado – (ID 6).
- Entidade (ID 8) – Lâmpada
    - Sensores
        - Medidor de consumo energético – (ID 1);
        - Luminosidade – (ID 2).
    - Atuadores
        - Controle de intensidade da iluminação e de estado da lâmpada – (ID 3).
- Entidade (ID 9) – Microcontrolador do jardim Inteligente
    - Sensores
        - Umidade – (ID 1);
        - Luminosidade – (ID 2);
        - Temperatura – (ID 3).
    - Atuadores
        - Motor para uma bomba de irrigação – (ID 4), ligado a entidade ID 9.
- Entidade (ID 10) – Microcontrolador
    - Sensores
        - Medidor de consumo energético (Computador 1) – (ID 1);
        - Medidor de consumo energético (Computador 2) – (ID 2);
        - Detector de Presença (Computador 1) – (ID 3);
        - Detector de Presença (Computador 2) – (ID 4);
    - Atuadores
        - Sistema de desligamento automático de estações de trabalho – (ID 5).

        - Sistema de desligamento automático de estações de trabalho – (ID 6).
- Outros dispositivos
    - Dois computadores (estações de trabalho), conectados a entidade ID 10, cada qual com seu próprio sensor de consumo energético e sensor de presença;
    - Ar Condicionado, conectado a entidade ID 7, com seu sensor de consumo energético.

Cozinha (Entidade ID 2)

- Entidade (ID 11) – Lâmpada
    - Sensores
        - Medidor de consumo energético (Lâmpada) – (ID 1);
        - Luminosidade – (ID 2);
        - Medidor de consumo energético (Geladeira) – (ID 3);
        - Sensor de ingredientes (Geladeira) – (ID 4).
    - Atuadores
        - Controle de intensidade da iluminação e de estado da lâmpada – (ID 5).

Quarto (Entidade ID 3)

- Entidade (ID 12) – Microcontrolador
    - Atuadores
        - Controle do volume das caixas de som – (ID 1);
        - Controle da abertura das persianas – (ID 2);
        - Alarme – (ID 3).
- Entidade (ID 13) – Lâmpada
    - Sensores
        - Medidor de consumo energético – (ID 1);
        - Luminosidade – (ID 2);
    - Atuadores
        - Controle de intensidade da iluminação e de estado da lâmpada – (ID 3).

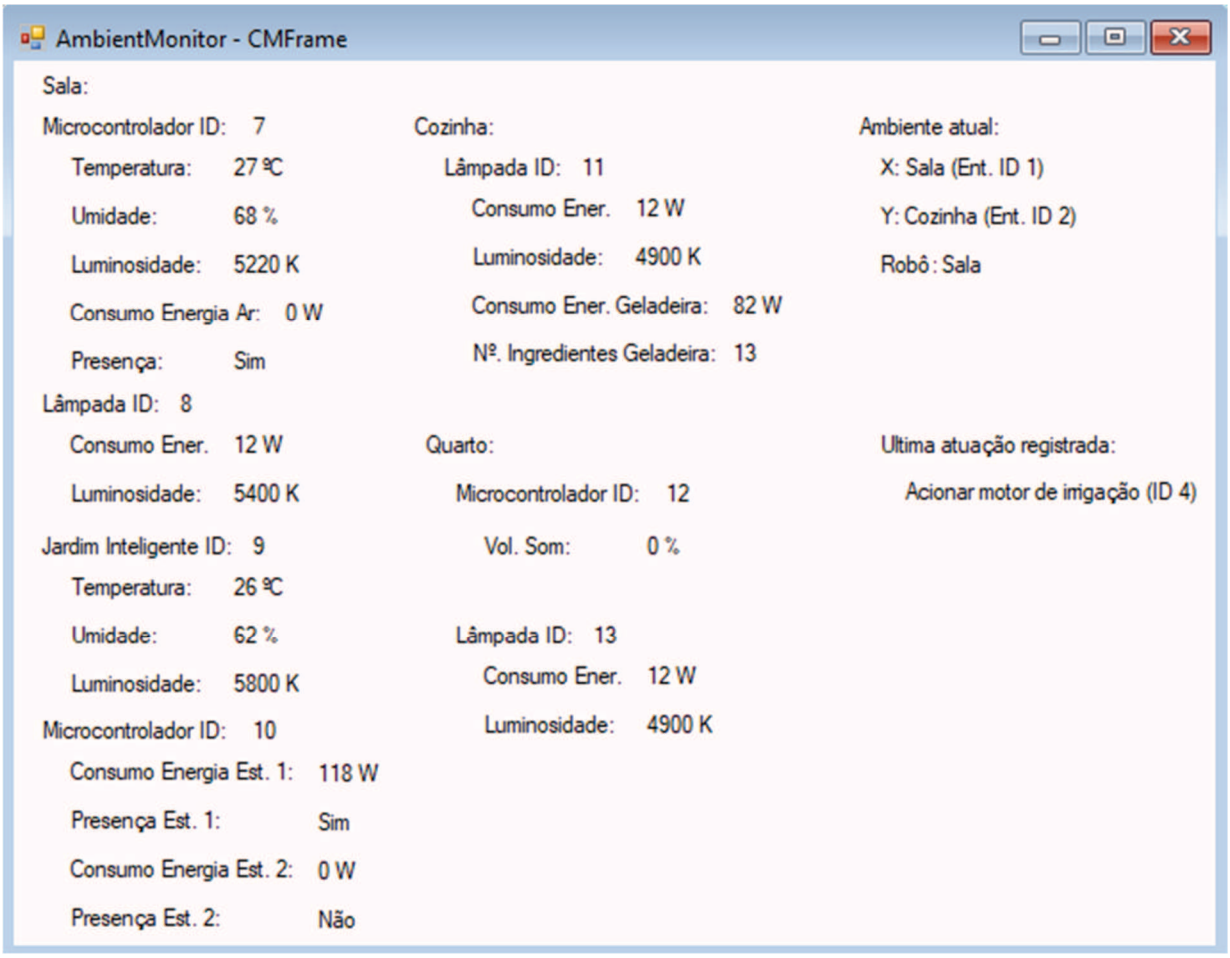
Neste cenário, considera-se que cada entidade, controlada por um microcontrolador, seja capaz de se conectar a mesma rede a qual se encontra uma instância do CMFDatabase, e assim se comunicar via MQTT através do protocolo de comunicação definido pelo CMFrame. A partir dessa comunicação, os dados captados pelos sensores, bem como informações pertinentes aos atuadores, são transmitidos ao CMFDatabase para armazenamento na forma de contexto, gerando assim um histórico cronológico para cada entidade presente neste cenário. Sendo assim, as informações de contexto pertinentes ao ambiente não são geradas pelas aplicações que nele influem, e sim pelos objetos de IoT.

5.3 AmbientMonitor

O AmbientMonitor tem como objetivo monitorar o estado atual de cada um dos ambientes da casa inteligente, bem como atuar sobre eles no momento em que é identificada uma mudança nos dados de contexto que requer uma atuação. As atuações modificam o comportamento de um ou mais elementos do ambiente, através da mudança nos parâmetros de um atuador. O CMFrame, no quesito atuações, só é responsável por fornecer as informações de contexto solicitadas pelo aplicativo, alertando-o através de um método de retorno. Os dados recebidos devem ser analisados pela aplicação a fim de determinar qual tipo de ação deve ser realizada, o que torna as atuações de completa responsabilidade das aplicações que venham a utilizar o *CMFrame*.

No AmbientMonitor, as informações de contexto são coletadas da base de dados de históricos a cada cinco segundos, atualizando a tela principal da aplicação com os últimos dados de contexto que foram armazenados pelos sensores e atuadores do ambiente. Esses dados por sua vez são analisados para verificar a necessidade de atuações por parte do aplicativo.

Figura 24: Tela do AmbientMonitor



Fonte: Elaborado pelo autor

A Figura 24 ilustra a tela do aplicativo, que exibe cada um dos sensores relevantes ao propósito da aplicação, com base no ambiente em que se encontram. No canto esquerdo da tela se encontram as entidades pertencentes à Sala, onde o microcontrolador de ID 7 possui os sensores gerais à Sala, como temperatura, umidade, luminosidade, bem como o sensor de presença e um indicador de consumo de energia para o ar condicionado. A entidade 8 fornece os dados acerca do consumo da lâmpada desse ambiente e sua luminosidade. O controlador do jardim inteligente (ID 9) ficará responsável por enviar os dados dos sensores e atuar sobre o motor de irrigação quando obter a confirmação por parte do aplicativo. A entidade ID 10 fornece dados acerca do consumo de cada uma das estações de trabalho e se existe um usuário presente utilizando-as.

Os dados disponibilizados no aplicativo refletem as últimas informações de contexto presentes no histórico das entidades, e, portanto, correspondem as últimas informações coletadas no cenário. Ao centro da tela encontram-se alguns dados pertinentes às entidades da Cozinha, com os sensores atrelados à lâmpada e a geladeira. Para a geladeira, são informados dados acerca de quais ingredientes se encontram em seu interior, bem como o número de ingredientes restante. O aplicativo também mostra os dados do volume atual do som definido para o Quarto, a cada variação com base nos dados de contexto. No canto direito da tela o aplicativo informa sobre a localização dos usuários dentro da casa inteligente, bem como em que cômodo o robô de limpeza se encontra. Essa informação é obtida através da verificação dos contextos hierárquicos recebidos para cada ambiente, em que é observado se no retorno existe a menção dessas entidades. Caso exista, significa que a entidade se encontra presente no ambiente. Por fim, o aplicativo também disponibiliza na tela a última atuação realizada.

Abaixo são descritas as principais etapas implementadas dentro da classe AmbientMonitor, relatando o fluxo realizado para atender ao cenário proposto, bem como os acontecimentos conforme os usuários (X e Y) interagem com o ambiente.

1. X (Entidade ID 4) entra na Sala (Entidade ID 1). O sensor de presença da sala o detecta, e salva o novo valor de presença na forma de contexto;
2. X abre o aplicativo do smartphone, que se comunica e registra a presença do usuário como contexto, que é verificado pelo aplicativo. O AmbientMonitor percebe essa mudança e solicita ao CMFrame uma mudança hierárquica, informando que a Entidade ID 4 passa a fazer parte da hierarquia da Entidade ID 1. Em seguida atua modificando os parâmetros de luminosidade definidos para a lâmpada da sala (Entidade ID 8), com base nos dados de contexto armazenados;
3. X aciona o ar condicionado manualmente. Sensor de consumo e temperatura passam a armazenar dados de contexto refletindo a nova condição.
4. O usuário X liga um computador e aproxima o smartphone do leitor ao seu lado. Ocorre uma atuação que modifica os valores do computador (tamanho fonte, volume das caixas de som, etc.) com base no contexto armazenado na entidade do cartão virtual com ID 4. Simultaneamente o sensor de presença atrelado ao computador passa a fornecer dados adicionais, uma vez que foi acionado. Esses dados de presença fazem uso dos contextos dinâmicos;

5. O usuário Y se dirige a cozinha, o sensor de presença armazena novos dados de contexto, que são consultados pelo aplicativo. O aplicativo solicita ao CMFrame uma mudança de hierarquia da entidade ID 5 para dentro da hierarquia da entidade ID 2 (cozinha);
6. Y retira os produtos da geladeira, fazendo com que seus sensores armazenem dados de contexto distintos sobre a quantidade de produtos disponíveis na geladeira, dados esses que são captados pelo aplicativo, atualizando a lista de ingredientes disponíveis;
7. O robô de limpeza se dirige à sala, são geradas duas trocas de hierarquia por parte do aplicativo, uma quando o robô sai do quarto para o corredor, desvinculando-se assim de qualquer entidade superior na hierarquia. A outra troca de hierarquia ocorre no momento em que o aplicativo detecta novos valores de contexto vindos do sensor GPS do robô, e que indiquem a sua nova localização na sala;
8. O aplicativo, ao fazer a leitura dos contextos do jardim inteligente, detecta necessidade de irrigação, acionando a atuação do motor presente dentro da entidade ID 9 (Microcontrolador do jardim);
9. X para de utilizar o computador da sala, o deixa ligado e se dirige a cozinha. Essa ação gera duas trocas hierárquicas para o usuário X, uma em que o desloca para o corredor (sem entidade) e a outra para a cozinha (entidade 2);
10. X se dirige ao quarto, gerando mais duas mudanças de hierarquia (corredor e quarto);
11. Y se dirige à sala, gerando novamente duas mudanças de hierarquia, liga novamente o ar condicionado, e se conecta a outro computador com o uso de seu smartphone, gerando uma atuação;
12. O aplicativo no smartphone de X registra sua presença no quarto, e então é definido o som ambiente do quarto, o horário para o acionamento do alarme e a abertura das persianas de manhã, tudo com base nos dados de contexto previamente armazenados na base de dados vinculada a seu cartão inteligente;
13. O robô de limpeza se dirige à cozinha para limpá-la, realizando novamente duas trocas de hierarquia;
14. Y sai da sala, deixando o computador ligado, e se dirige à cozinha para lavar a louça, gerando duas mudanças hierárquicas;
15. Y sai da cozinha e se dirige ao quarto, fazendo com que o aplicativo ajuste as condições para a nova hierarquia.

A Tabela 4 ilustra um trecho dos dados de contexto obtidos pela aplicação, a partir da leitura da base de dados, separados pelo instante de tempo correspondente aos itens de fluxo acima listados.

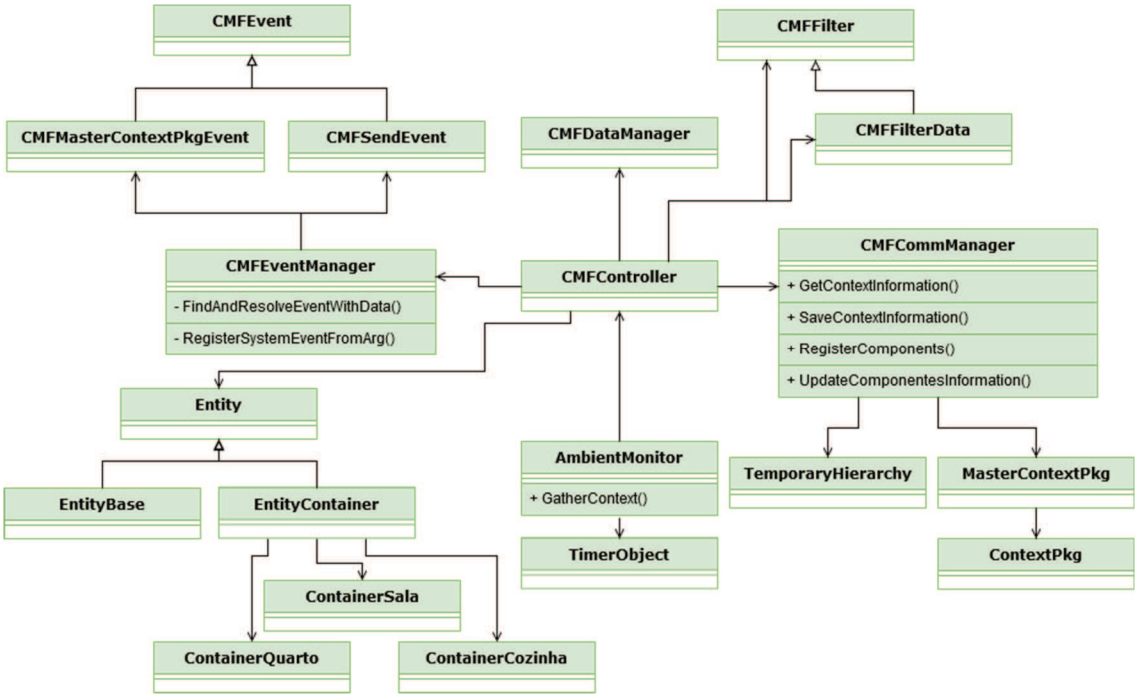
Tabela 4: Contextos do AmbientMonitor

Instante: 1	**Instante: 2 à 4**	**Instante: 5**	**Instante: 6 à 7**
Sala:	**Sala:**	**Sala:**	**Sala:**
Sem usuários na sala; Lâmpadas desativadas; Ar condicionado desativado; Computadores desativados.	Usuário X na sala; Lâmpadas ativas; Ar condicionado ligado; Computador 1 ativo.	Usuário X na sala; Lâmpadas ativas; Ar condicionado ligado; Computador 1 ativo.	Usuário X na sala; Lâmpadas ativas; Ar condicionado ligado; Computador 1 ativo; Robô presente na sala.
		Cozinha:	**Cozinha:**
		Usuário Y presente; 22 Ingredientes na geladeira.	Usuário Y presente; 16 Ingredientes na geladeira.

Fonte: Elaborado pelo autor

A Figura 25 apresenta o diagrama de classes do AmbientMonitor, com as classes disponibilizadas pelo módulo CMFrame, em conjunto com a classe principal do aplicativo.

Figura 25: Diagrama de classe do AmbientMonitor



Fonte: Elaborado pelo autor.

Somente foram mencionados os métodos principais dentro de cada classe. Como o principal meio de comunicação do aplicativo para com as demais classes do CMFrame, foi criada a classe AmbientMonitor, que engloba toda a lógica de funcionamento do aplicativo. Suas atribuições incluem todas as chamadas por informações de contexto, através do uso do *CommunicationManager* para a requisição e transmissão de mensagens entre os módulos. Não houve a necessidade de expandir a classe de comunicação, visto que os métodos de contexto fornecidos já suprem as necessidades do AmbientMonitor, através da coleta de contextos em hierarquia. Também foi desenvolvida uma classe de temporizadores para regular a cadência com que as informações de contexto são requisitadas ao *framework*. A classe de entidades *container* foi expandida em três derivações, para fornecer o comportamento específico para cada ambiente analisado pela aplicação, com base nos dados de contexto.

As mudanças de hierarquia são sempre solicitadas pelo aplicativo, a seu critério. No AmbientMonitor, esse critério é a análise dos dados de contexto vindos dos sensores de presença, sempre mantendo o local onde a entidade se encontra através da hierarquia. O aplicativo armazena cada uma das trocas de hierarquia dos

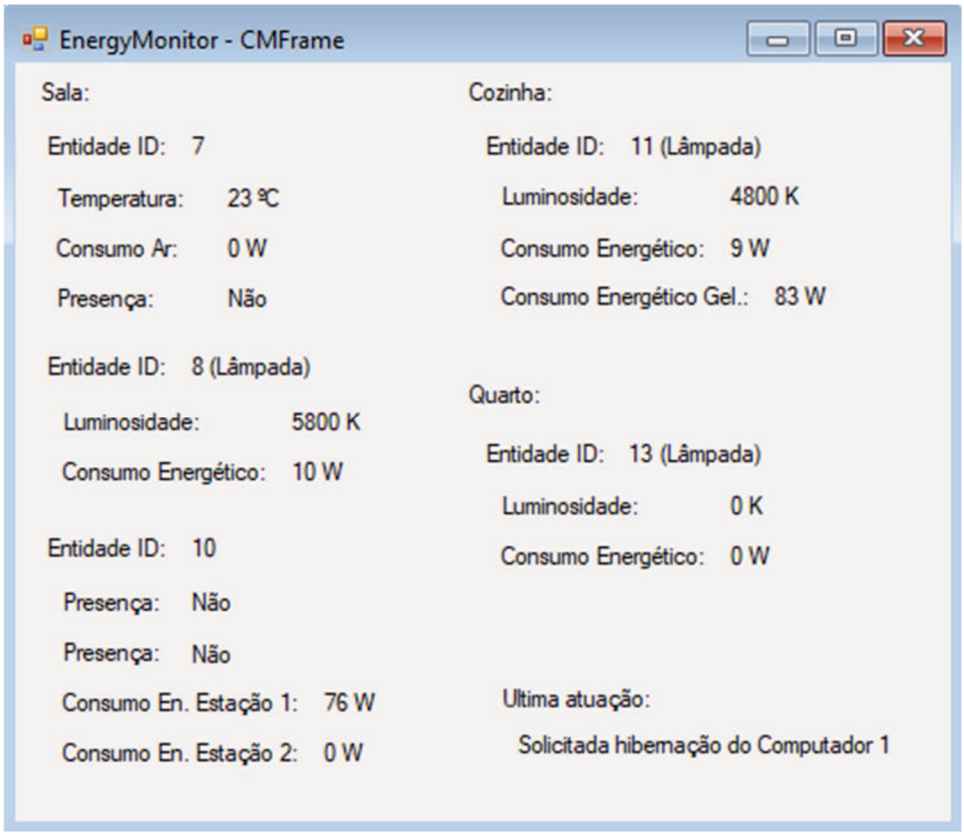
usuários em seus respectivos históricos, através do uso de contextos dinâmicos fornecido pelo *framework.* Os sensores de presença utilizam da mesma característica dinâmica, visto que seus dados somente são enviados, através do microcontrolador, quando esse sensor é ativado. Com isso a quantidade de informações armazenada naquele instante para a entidade (microcontrolador) difere dos demais. Através da hierarquia, o AmbientMonitor consegue detectar em qual ambiente o robô se encontra, bem como se está limpando o ambiente, qual a posição onde a limpeza ocorre, o número de limpezas diárias, o momento em que começa a limpeza, e o instante em que termina, através da verificação de contextos de um ambiente específico.

5.4 EnergyMonitor

O EnergyMonitor é um aplicativo para o controle do consumo de energia para os três ambientes apresentados no cenário da casa inteligente, a sala, a cozinha e o quarto, atuando nos mesmos conforme a necessidade, com base nos dados de contexto previamente armazenados. Para isso, tem como intuito utilizar os mecanismos de hierarquia, portanto entidades hierárquicas e dinâmicas, em uma configuração que englobe os dados de todos os sensores presentes nestes ambientes, de forma simultânea. A partir disso, será possível que os dados sejam consultados através de uma única chamada de método, retirando a necessidade de utilizar-se dos meios convencionais em que se verifica individualmente os dados de contexto de cada entidade. Essa hierarquia de entidades pode ser realizada a qualquer momento para refletir diferentes configurações dos ambientes presentes na base de dados.

A cada cinco segundos o EnergyMonitor consulta os dados de contexto armazenados na base de dados, verificando os sensores de presença e os sensores de consumo a fim de determinar suas atuações. O EnergyMonitor consulta o mesmo conjunto de dados que o AmbientMonitor utiliza para suas inferências, fazendo com que ambas aplicações afetem o ambiente simultaneamente, com base em seus objetivos.

Figura 26: EnergyMonitor



Fonte: Elaborado pelo autor.

A Figura 26 mostra a tela do aplicativo, exibindo os dados considerados relevantes ao propósito da aplicação (gerenciamento de energia) dentre todos os sensores que se encontram presentes nos três ambientes. No canto esquerdo da tela estão dispostos os dados referentes ao ambiente Sala, onde são monitorados os dados acerca do consumo do ar condicionado, bem como se existem usuários presentes naquele instante. No lado direito da tela, o aplicativo mostra as informações do sistema de iluminação presente na cozinha, bem como o consumo energético da geladeira conectada a entidade 11. Conforme a quantidade de itens presentes na geladeira, sua temperatura interna será regulada, mantendo-os refrigerados, sem excessos, para economizar energia. Os dados de contexto provenientes do quarto também são captados, provendo uma leitura sobre o consumo energético de seus dispositivos. Como último elemento da tela, o aplicativo informa sobre a última atuação realizada.

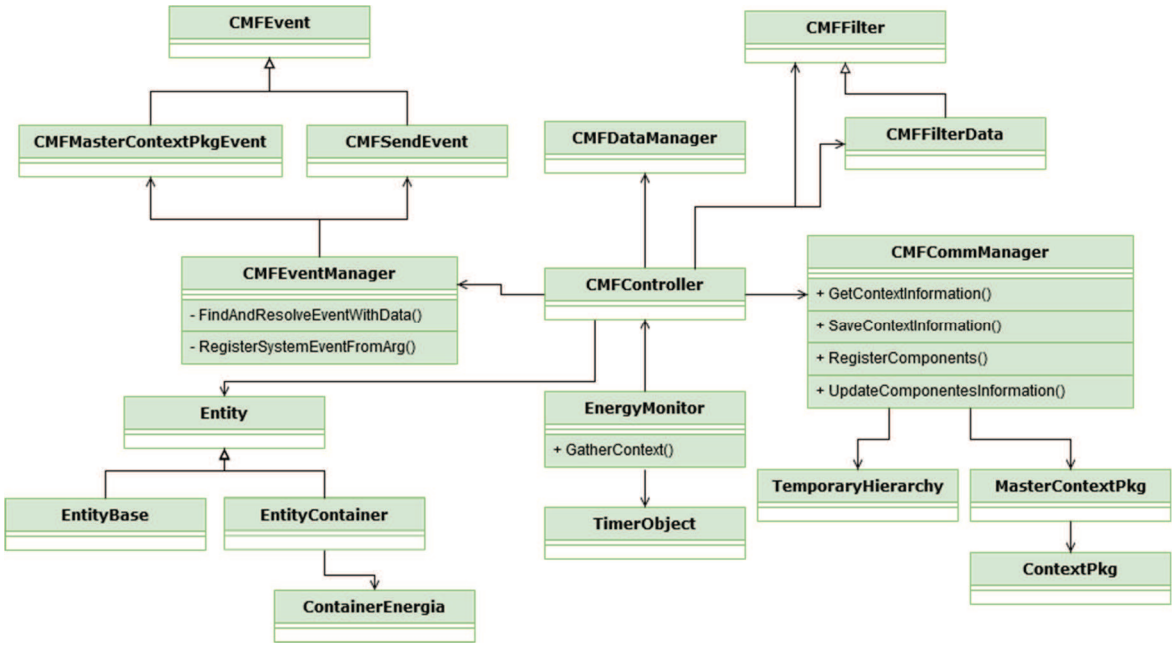
Abaixo são descritas as principais etapas implementadas dentro da classe EnergyMonitor, relatando o fluxo realizado pelo aplicativo em conjunto com as ações que ocorreram no cenário, bem como seus parâmetros mais relevantes.

1. O aplicativo solicita ao CMFrame a criação de uma hierarquia que englobe todos os ambientes, sala, cozinha e quarto, e a partir dessa estrutura, pode solicitar e receber os contextos de forma hierárquica;
2. Conforme o fluxo definido pelo cenário, o usuário X entra na sala e liga seu aplicativo, a lâmpada é acionada automaticamente. Então o sensor de consumo da lâmpada armazena as novas leituras como contexto, que são captadas posteriormente pelo aplicativo. Simultaneamente o ar condicionado é acionado de forma automática, com base no perfil do usuário, modificando o sensor de consumo atrelado a esse equipamento;
3. X liga um computador, modificando o sensor de consumo e ativando o sensor de presença para aquela máquina. Essa informação será monitorada pelo aplicativo;
4. O usuário Y entra na cozinha e aciona as lâmpadas, modificando a leitura dos sensores;
5. O usuário X para de utilizar o computador da sala e se dirige à cozinha. O sensor de presença acusa sua ausência na sala. Essa informação será monitorada pelo aplicativo;
6. O aplicativo detecta que X saiu da sala, e inicia uma contagem para uma atuação a fim de reduzir a operação do ar condicionado do ambiente;
7. Após cinco minutos, o aplicativo atua para hibernar o computador utilizado por X, para economizar energia, visto que o sensor de presença não o acusa na máquina e nem na sala. O ar condicionado da sala também é desligado;
8. X entra no quarto, as lâmpadas são acionadas automaticamente, com base no perfil do usuário, assim modificando os valores monitorados pelo EnergyMonitor.
9. Y entra na sala e aciona o ar condicionado manualmente, os sensores atrelados a ele têm seus valores atualizados;
10. Y liga o outro computador presente na sala, o sensor de consumo reflete os novos valores;
11. Posteriormente, Y sai da sala em direção à cozinha. O aplicativo inicia uma contagem para desligar tanto as lâmpadas da sala quanto o computador. O ar condicionado também será desligado a fim de economizar energia;

12. Y sai da cozinha e se dirige ao quarto. O aplicativo detecta sua saída e inicia a contagem para desligar o sistema de iluminação da cozinha.

Os dados de contexto em hierarquia foram recebidos através da classe *MasterContextPkg* e *ContextPkg*, que determinam a forma como os contextos hierárquicos serão recebidos por uma instância do CMFrame, fornecendo uma estrutura definida para organizar esse volume de dados. A classe *ContextPkg* foi desenvolvida tendo em vista a natureza dinâmica dos contextos, onde a quantidade de informações presentes em um contexto não é conhecida previamente pelo *framework.* Dessa forma, mesmo que o usuário do CMFrame não derive a classe para customizar seu comportamento, os dados variáveis de contexto existentes serão automaticamente incorporados e disponibilizados cada vez que o método de retorno cadastrado para os contextos hierárquicos for acionado.

Figura 27: Diagrama de classe do EnergyMonitor



Fonte: Elaborado pelo autor.

A Figura 27 mostra o diagrama de classes do EnergyMonitor, com as classes presentes no CMFrame, em conjunto com a classe principal do aplicativo. O EnergyMonitor emprega a mesma classe de temporizadores desenvolvida para o AmbientMonitor, e contém sua lógica na classe *EnergyMonitor*, utilizando os métodos providos pelo CMFrame para coletar simultaneamente as informações de contexto

dos três ambientes presentes no cenário, e assim fazer o julgamento acerca das atuações necessárias.

As modificações do AmbientMonitor se encontram na lógica do aplicativo, realizando atuações distintas com base no seu propósito, bem como na estrutura hierárquica que foi solicitada. No EnergyMonitor, o aplicativo cria uma única hierarquia, no início de sua execução, responsável por englobar todos os sensores e atuadores presentes em cada um dos ambientes, assim permitindo que colete os contextos hierarquicamente a cada cinco segundos. A classe de entidades *container* foi expandida para tratar os dados de contexto que dizem respeito ao consumo de energia.

5.5 Avaliação do Framework

Para avaliar o *framework* foram desenvolvidas duas aplicações distintas, denominadas AmbientMonitor e EnergyMonitor. Ambas aplicações fazem uso do cenário proposto na seção 5.2.1, utilizando os mesmos ambientes e, portanto, empregando o mesmo conjunto de dados. Através do desenvolvimento das aplicações, busca-se avaliar as características oferecidas pelo CMFrame, procurando responder questões sobre quais contribuições e limitações existem na utilização de entidades hierárquicas e dinâmicas para sistemas sensíveis a contexto. É importante salientar que ambas aplicações se encontram em execução simultaneamente, compartilhando o mesmo conjunto de dados e afetam os mesmos ambientes com suas atuações.

Os dados armazenados dentro da base de dados do CMFrame foram simulados, tendo em vista valores condizentes com os sensores propostos. A simulação foi realizada através de um aplicativo para realizar inserções na base de dados. Em um primeiro momento, o AmbientMonitor e o EnergyMonitor foram desenvolvidos com todas as regras necessárias para a execução dos passos pertinentes a cada um, definidos nos itens 5.3 e 5.4. Em seguida, o cenário foi montado com os dados iniciais necessários, e ambas aplicações foram inicializadas e conectadas a uma instância do CMFDatabase. Após essa etapa, os dados necessários para simular as situações ocorridas no cenário foram inseridos um a um,

diretamente na base de dados com o uso do aplicativo. Dessa forma é possível replicar o comportamento de um ambiente real, que recebe dados cronológicos de contexto, formando um histórico ao longo do tempo. Considerando que as aplicações checam pelos últimos dados de contexto presentes, e que não existem ações no fluxo do cenário próximas entre si que sejam dependentes umas das outras, torna-se possível empregar o método de inserção de dados pelas etapas do cenário.

A avaliação realizada sobre o aplicativo AmbientMonitor visa averiguar os mecanismos de troca hierárquica, e se os mesmos apresentam contribuições para seus utilizadores, na forma de pontos positivos e pontos negativos identificados através de sua utilização.

O EnergyMonitor, por sua vez, tem sua avaliação mais direcionada ao gerenciamento de um aspecto específico (gerenciamento de energia), mas que diz respeito a sensores e atuadores que se encontram presentes em mais de um ambiente. Dessa forma, o EnergyMonitor gerencia um aspecto particular de um conjunto de ambientes, enquanto existe outro aplicativo atuando sobre o mesmo conjunto de dados, mas provendo uma funcionalidade distinta. Em sua implementação, a hierarquia é realizada uma vez, no início da aplicação, e posteriormente é utilizada para consultar os contextos de toda a sua estrutura, filtrando em nível de aplicação os dados relevantes. Assim torna-se possível que o EnergyMonitor e o AmbientMonitor tenham estruturas hierárquicas distintas, cada uma voltada à necessidade específica de sua aplicação.

5.5.1 Avaliação AmbientMonitor

Nesta seção será descrito a análise sobre o aplicativo AmbientMonitor, quanto aos quesitos oferecidos pelo CMFrame, relatando as percepções e conclusões obtidas através de seu desenvolvimento. Serão considerados os aspectos referentes ao uso de suas entidades hierárquicas e dinâmicas, trocas de hierarquia, utilização de contextos dinâmicos, bem como as limitações encontradas ao utilizar essas funções.

Dentro do cenário desenvolvido, o aplicativo do smartphone de cada usuário, que os representa como entidades de ID 4 e ID 5, é responsável por informar quando

os mesmos trocam de ambiente, e armazenar no perfil. A partir dessa comunicação, o AmbientMonitor verifica os dados de contexto apropriados para detectar este deslocamento do usuário, e solicita ao seu módulo do CMFrame pela troca de hierarquia e pelo armazenando do ambiente em que ela se encontra, no histórico da entidade. Dessa forma, o AmbientMonitor centraliza ambas funções e permite a avaliação sob o ponto de vista de aplicativo único.

- **Quanto às entidades hierárquicas e dinâmicas**

O AmbientMonitor faz uso das entidades hierárquicas e dinâmicas para manter a representação do ambiente constantemente atualizada, refletindo as condições físicas dos ambientes em sua representação virtual, através das entidades. Isso significa que conforme um elemento do mundo físico troca de ambiente, o aplicativo realiza uma ação correspondente de troca de hierarquia, informando o *framework* sobre o novo local (entidade) onde aquela entidade móvel se encontra.

As entidades móveis presentes no cenário são seus dois usuários, X e Y, e o Robô de limpeza. Seu livre trânsito por diferentes ambientes reflete condições reais vistas em muitos ambientes inteligentes, onde existem entidades móveis que possuem uma representação física e se deslocam para desempenhar suas funções, possivelmente alterando o funcionamento de múltiplos elementos físicos ou virtuais durante esse processo.

A partir dessa hierarquia atualizada, o aplicativo pode solicitar os dados de uma entidade específica, que irá fielmente representar as condições reais do ambiente e proverá dados de contexto armazenados no histórico para todas as entidades presentes no ambiente. No caso do AmbientMonitor, são realizadas solicitações por contextos das entidades “sala”, “cozinha” e “quarto”, com uma cadência constante. Como a hierarquia se encontra atualizada, os dados recebidos corresponderão a todas as entidades presentes nesses ambientes. A partir dos últimos valores de contexto recebidos para cada uma das entidades é verificado se existe a necessidade de atuar sobre o ambiente.

Considerando a implementação do AmbientMonitor, é possível afirmar que a utilização das consultas hierárquicas traz benefícios quando comparado ao método

de consulta convencional. Sua principal vantagem está na redução de solicitações por contexto ao *framework* trafegando pela rede, bem como na organização e manutenção futura do código. O cenário proposto é composto por 10 entidades estáticas e três móveis. A cada cinco segundos é solicitado o contexto hierárquico das entidades de ID 1,2 e 3, transmitindo também a qual aplicativo aquela hierarquia pertence, bem como um identificador para designar a estrutura hierárquica em questão, para quando houver mais de uma hierarquia. Dessa forma, é somente realizada uma chamada por contextos, que será transmitida via rede até o módulo CMFDatabase e processada, retornando ao aplicativo todos os contextos solicitados através da hierarquia definida, em um único processo.

Outra vantagem de empregar as entidades hierárquicas e dinâmicas em conjunto com as trocas hierárquicas é que, quando a hierarquia sofre modificações, o método de consulta passa a refletir a nova condição da hierarquia imediatamente. Sem esse mecanismo em operação, seria necessário que a aplicação sempre tivesse conhecimento sobre todos os elementos que compõe um determinado ambiente, a todo o momento. A partir disso seria necessário realizar uma chamada de contexto para cada uma destas entidades, o que geraria, nesse caso, dez chamadas distintas por contextos, e dez verificações distintas por contextos nos métodos de retorno.

- **Quanto ao uso dos contextos dinâmicos**

O uso de contextos dinâmicos se refere à capacidade de modificar a quantidade de elementos armazenados como contexto para uma entidade em tempo de execução. No AmbientMonitor, essa funcionalidade foi empregada em duas situações distintas, primeiro para os sensores de presença distribuídos no ambiente, e posteriormente ao armazenar as hierarquias anteriores de cada entidade.

No caso dos dados vindos do sensor de presença, existe uma configuração no microcontrolador (entidade 7) e no microcontrolador da entidade 10, que somente envia os dados específicos de presença quando existe algum indivíduo presente no ambiente. No caso da entidade 10, o sistema funciona da mesma forma, enviando dados de presença quando algum usuário se encontra utilizando uma estação de trabalho. O armazenamento de informações dessa maneira possibilita a existência de

diferentes conjuntos de informações de contexto em instantes de tempo distintos, sendo armazenados no histórico da mesma entidade.

O armazenamento de hierarquias para uma entidade móvel, por sua vez, designa que, a partir da solicitação do aplicativo, será salvo o momento em que esta entidade realizou uma troca hierárquica, guardando a sua nova posição dentro da hierarquia. Como o armazenamento dessas informações é feito na base de dados específica da entidade, emprega-se diretamente o uso dos contextos dinâmicos, visto que a base agora armazena simultaneamente dados normais de contexto, com seus valores específicos, e também dados relativos à hierarquia a que a entidade pertence.

Ao permitir que os tipos de informações armazenadas mudem livremente durante o tempo de execução, e sem definir previamente qual serão os formatos dos dados a serem armazenados, torna-se possível afirmar, que para o AmbientMonitor, a possibilidade de armazenar contextos com tamanha dinamicidade traz maior praticidade e conveniência ao uso do *framework*, tanto em seu processo de armazenamento interno, quanto para o desenvolvedor de aplicações.

- **Limitações**

As solicitações por informações de contexto requerem que o utilizador do *framework* informe um parâmetro pela qual as pesquisas se baseiam. Esse parâmetro é definido através da utilização das classes derivadas do CMFFilter e, a menos que seja explicitamente solicitado, emprega por padrão um filtro nulo (que retorna todos os resultados). Quando se considera a natureza do armazenamento de dados de contexto na forma de histórico, nota-se que essas informações chegarão a quantidades proibitivas para o transporte pela rede, e mesmo que as conexões de rede em uso permitam que a transmissão ocorra, torna-se difícil lidar com esse volume de dados. Para remediar essa situação, é possível definir filtros mais específicos para o propósito do utilizador, limitando quais dados de contexto serão retornados, e até mesmo a quantidade máxima de elementos retornados. Para situações em que as informações de contexto são mais complexas ou muito concatenadas, recomenda-se que seu retorno seja feito com a utilização de estruturas de dados customizadas pelo desenvolvedor, que desmembrem os elementos a fim de facilitar a utilização dos contextos no aplicativo.

Os filtros implementados no CMFrame são restritos a apenas um filtro aplicado a toda a hierarquia, o que não é ideal. Uma solução mais plausível seria permitir que o usuário limitasse a busca designando quais informações deseja de cada entidade na hierarquia, e utilize o filtro padrão nas entidades não especificadas. As limitações encontradas no desenvolvimento do aplicativo não estão atreladas diretamente ao conceito das entidades hierárquicas e dinâmicas, estando voltadas a questões pontuais de implementação do *framework.* Algumas dessas questões, bem como outras que foram descobertas durante o desenvolvimento das aplicações, e que não eram passiveis de reparo imediato foram sugeridas na seção de trabalhos futuros.

5.5.2 Avaliação EnergyMonitor

Essa seção descreve a análise do aplicativo EnergyMonitor, quanto as funcionalidades do CMFrame por ele empregadas, relatando as percepções e conclusões obtidas no seu desenvolvimento.

O EnergyMonitor foi executado simultaneamente ao AmbientMonitor, enquanto que ambos realizam todas as suas requisições a mesma instância do módulo CMFDatabase, que também se encontra conectado na mesma rede, e se comunica com as demais instâncias do CMFrame através do protocolo MQTT. Essa execução conjunta demonstra a capacidade de executar múltiplas aplicações que compartilham a mesma visão sobre o ambiente, definido como um dos requisitos para o *framework*.

A seguir serão descritas as percepções relacionadas a entidades hierárquicas e dinâmicas, bem como contextos dinâmicos a partir do desenvolvimento do EnergyMonitor.

- **Quanto às entidades hierárquicas e dinâmicas**

Para o aplicativo EnergyMonitor foi criada uma única hierarquia, utilizada ao longo de todo seu ciclo de vida, definindo uma agregação de todos os elementos presentes no cenário, que são automaticamente organizadas entre entidades

containers e entidades bases atreladas como nodos filhos. Essa estrutura hierárquica serve como base para a coleta de todas as informações de contexto através de uma chamada de método única, em que os resultados são filtrados no método de retorno. Tendo em vista que os contextos vindos das entidades móveis presentes nesse cenário específico não são relevantes para o propósito do gerenciamento de energia, torna-se possível empregar a mesma hierarquia durante toda a execução do aplicativo, sem excluir nenhuma informação de contexto importante. Ao estabelecer a comunicação com o módulo CMFDatabase, o aplicativo coletou as informações de contexto hierarquicamente, efetuando a análise desses dados para a execução de suas atuações conforme descrito pelo fluxo do aplicativo na seção 5.4.

- **Quanto ao uso dos contextos dinâmicos**

Os aspectos referentes ao uso dos contextos dinâmicos por parte do EnergyMonitor foram os mesmos que os observados para o AmbientMonitor, visto que é uma configuração fornecida pelos microcontroladores (entidade 7 e 10) presentes no ambiente, e que utilizam do módulo CMFrame para regular o armazenamento de contextos com fatores dinâmicos. Seu uso também beneficia o EnergyMonitor automaticamente, visto que o mesmo pode coletar os dados de presença para as estações de trabalho a fim de regular a hibernação das estações quando as mesmas não se encontram em uso. Tendo em vista que não foram realizadas trocas hierárquicas no EnergyMonitor, os dados referentes a nova posição na hierarquia de uma entidade não foram armazenados por esse aplicativo, como realizado no AmbientMonitor.

- **Limitações**

As limitações encontradas no EnergyMonitor dizem respeito aos mesmos quesitos encontrados no aplicativo AmbientMonitor, visto que as dificuldades não estão atreladas diretamente aos conceitos de entidades hierárquicas e dinâmicas, mas sim a questões particulares da implementação realizada no *framework*.

A fim de detectar possíveis problemas, também houve a verificação dos logs da aplicação, para averiguar se a troca de mensagens entre as instâncias era efetuada com sucesso, bem como se as informações de contexto coletadas condiziam com os dados armazenados na base de dados para os filtros de busca definidos pela

aplicação. Quanto a esses parâmetros, não foram encontradas irregularidades de comunicação considerando a comunicação simultânea entre as três instâncias durante o período de testes.

5.6 Considerações sobre o capítulo

Este capítulo descreveu os aspectos de implementação para ambos os módulos que compõe o CMFrame. Foi descrita a metodologia de avaliação, bem como o cenário proposto. A avaliação do CMFrame foi realizada com base no desenvolvimento de duas aplicações, *AmbientMonitor* e *EnergyMonitor*, permitindo a avaliação do conceito de entidades hierárquicas e dinâmicas que o *framework* oferta. No próximo capítulo serão apresentadas as considerações finais acerca do trabalho, dando destaque as suas principais contribuições e elencando os trabalhos futuros.

6. CONSIDERAÇÕES FINAIS

A dissertação abordou temas relacionados a *frameworks*, objetos de internet das coisas, sistemas cibernéticos físicos, bem como ambientes físicos que adquirem certo nível de autonomia ao permitir que os mesmos reajam aos usuários presentes. Foi conduzida uma pesquisa a fim de identificar *frameworks* que realizassem o

gerenciamento de históricos de contextos, e averiguar acerca da forma como conduzem a estruturação de seus contextos para facilitar o tratamento posterior.

A especificação de um *framework,* denominado CMFrame, foi proposta e desenvolvida. Seu propósito é oferecer uma outra forma de lidar com as informações de contexto armazenadas a fim de gerar um histórico, empregando um conceito de entidades hierárquicas e dinâmicas para melhor representar os ambientes inteligentes atuais, e prover suporte a informações de contexto variadas. O *framework* oferece métodos para que dispositivos controladores possam interagir com uma base de dados, e assim armazenar os dados de contexto adquiridos em seu ambiente, bem como permitir que aplicações externas possam empregar desses dados de contexto para tomar decisões acerca do ambiente e reagir aos usuários presentes.

Com base no desenvolvimento de duas aplicações e sua implementação no cenário de uma casa inteligente aplicado as instalações do Mobilab, verificou-se que o *framework* atende as necessidades como um gerenciador de históricos de contextos, para situações que utilizem ambientes físicos com elementos inteligentes e aplicações que fazem a gerência do ambiente através da atuação sobre os mesmos, como nos casos avaliados. Tendo como mecanismo complementar a esse objetivo, o emprego de entidades hierárquicas e dinâmicas. Seu uso visa facilitar o processo organizacional dos elementos de um ambiente passíveis de representação através de entidades, sejam elas reais ou virtuais, permitindo que os contextos de maior relevância para a aplicação sejam disponibilizados no momento desejado através do seu relacionamento entre as entidades.

Dessa forma, o CMFrame pode ser utilizado em situações onde existam ambientes físicos que necessitam de características inteligentes, armazenando e consultando informações sobre si, a fim de refinar seu comportamento através da análise de suas condições. Para tanto, torna-se necessário que os elementos presentes nestes ambientes sejam capazes de representação através do conceito de entidades. Essas condições o tornam uma opção válida para, porém não restrito a, aplicações que realizam tarefas gerencias sob um ambiente, visto que os mesmos se beneficiam do monitoramento e análise constante de suas condições internas para a melhor tomada de decisão acerca das ações seguintes a serem tomadas.

6.1 Contribuições

A principal contribuição desse trabalho, bem como um fator essencial para o mesmo, diz respeito à forma como organiza suas estruturas internamente, especificamente a concepção das entidades hierárquicas e dinâmicas.

Esse conceito oferece uma nova forma de organizar os contextos, através de seu vínculo com as entidades, permitindo também a modificação do relacionamento existente entre cada uma dessas entidades, através de uma hierarquia. Durante o ciclo da vida de uma aplicação, é permitido que a mesma altere a ordem entre entidades e assim seu relacionamento, com o intuito de melhor refletir o ambiente dinâmico a que essas entidades representam, além de agrupa-las, determinando quais delas pertencem a quem. Esse agrupamento torna-se útil quando se deseja avaliar os contextos de uma determinada entidade que se encontra em um nível hierárquico mais elevado, como por exemplo, uma entidade que represente todo um prédio, e que dentro de si possua uma vasta quantidade de entidades menores representando todos os elementos que compõe esse prédio e que possam ser representados através de entidades. Porém, não há nenhuma limitação técnica que impeça o *framework* de ser utilizado para armazenar contextos em ambientes que não se favoreçam da modificação hierárquica fornecida por suas entidades hierárquicas e dinâmicas, caso o desenvolvedor não deseje fazer uso dessa funcionalidade. O *framework* disponibiliza uma forma pela qual desenvolvedores possam usufruir de informações de contexto geradas por ambientes físicos, através da configuração do comportamento que cada uma das entidades exibirám, sendo que cada aplicação pode definir seu próprio comportamento para uma determinada entidade.

Outra contribuição reside no fato de que os contextos armazenados pelo CMFrame são dinâmicos em sua natureza, e podem armazenar quantidades e tipos de dados diferentes ao longo do tempo, para a mesma entidade. Isso permite efetivamente que o utilizador do *framework* não fique limitado quanto às informações de contexto que necessita armazenar a qualquer momento, além de permitir que o mesmo utilize sensores e atuadores que não foram desenvolvidos no período em que esse trabalho foi proposto.

Essas funções têm como objetivo final melhor refletir os ambientes inteligentes que empregam modificações nos elementos que os compõe ao longo do tempo, como por exemplo, uma linha de montagem que substitua as estações de produção a fim de modificar o produto que se encontra em manufatura.

6.2 Trabalhos Futuros

A partir do desenvolvimento do *framework* proposto neste trabalho, em conjunto com as aplicações que o empregam para gerar inferências sobre um cenário existente no mundo real, torna-se possível elencar alguns pontos do trabalho que podem se beneficiar de futuros refinamentos. A maioria desses pontos gira em torno de aspectos de implementação do *framework*, soluções distintas para abordar um problema específico já existente, ou mesmo para suprir uma nova necessidade que não se encontra presente no escopo original desse trabalho. Tendo em vista a natureza do *framework*, e suas possíveis aplicações em sistemas reais, torna-se necessário observar as necessidades que surgem quando utilizado em ambientes de maior escala e, portanto, fornecendo um maior nível de confiança quanto a sua robustez para o utilizador final.

Abaixo são mencionadas algumas das possibilidades de trabalhos futuros aplicados a esse *framework.* A lista não é exaustiva e representa apenas os pontos principais:

- Expansão da parcela do *framework* responsável por gerenciar as interações com o banco de dados, permitindo que exista mais de um processo utilizando do mesmo conjunto de dados, facilitando a redundância e aumentando a disponibilidade;
- Expansão do componente de eventos, incorporando funcionalidades que permitam a criação de eventos com tarefas predefinidas em intervalos de tempo constante;
- Armazenamento de dados de forma local, para aqueles dispositivos móveis que possam em determinado momento sair da área de cobertura do sistema, mas cujas informações permanecem relevantes para uso futuro;

- Melhoria do sincronismo interno entre diferentes tipos de mensagens enviadas ao módulo de banco de dados, visto que certas mensagens necessitam de prioridade em sua execução a fim de garantir a consistência dos dados;
- Simplificar a forma como o framework lida com as trocas de hierarquia, a fim de torna-las mais simples e permitindo que seja definida sem múltiplas invocações de métodos;
- Desenvolver uma forma mais simplificada de representar uma quantidade maior de informações de contexto em entidades hierárquicas;
- Desenvolvimento de métricas que permitam que o *framework*, a partir da análise de seus dados, retorne o estado válido de uma determinada entidade em certo período de tempo;
- Expandir a complexidade dos filtros de pesquisa implementados pelo *framework*, a fim de fornecer uma maior gama de resultados ao usuário final;
- Otimizações para melhora do desempenho das conversões de estrutura realizadas internamente para o sistema de contextos dinâmicos;
- Desenvolvimento das interfaces de fácil utilização para os dispositivos *Android* e Microcontroladores como *Arduino* e *Raspberry-Pi*.

REFERÊNCIAS

ANDERT, G. Object frameworks in the Taligent OS. Compcon Spring ’94, Digest of Papers. p.112–121, 1994.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A survey. Computer Networks, v. 54, n. 15, p. 2787–2805, 2010.

AUGUSTO, J. C.; CALLAGHAN, V.; COOK, D.; KAMEAS, A.; SATOH, I. “Intelligent Environments: a manifesto”. Human-centric Computing and Information Sciences, v. 3, n. 1, p. 1-12, 2013.

BABICEANU, R. F.; SEKER, R. Big Data and virtualization for manufacturing cyber-physical systems: A survey of the current status and future outlook. Computers in Industry, v. 81, p. 128–137, 2016.

BERMUDEZ, L.; DELORY, E.; O’REILLY, T.; RIO FERNANDEZ, J. DEL. Ocean observing systems demystified. OCEANS 2009. p.1–7, 2009.

BUSCHMANN, F.; MEUNIER, R. A System of Patterns, Proceedings of the First Conference on Pattern Languages and Programming, Addison-Wesley, 1994.

CAMBRUZZI, W.; RIGO, S. J.; BARBOSA, J. L. V. Dropout Prediction and Reduction in Distance Education Courses with the Learning Analytics Multitrail Approach. Journal of Universal Computer Science, v. 21, n. 1, p. 23–47, 2015.

CATTELL, R. Scalable SQL and NoSQL data stores. ACM SIGMOD Record, v. 39, n. 4, p. 12-27, 2011.

CHAOUCHI, H.: The Internet of Things – Connecting Objects to the Web. J. Wiley Publ., 2010.

CHEN, C. L. P.; ZHANG, C.-Y. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. Information Sciences, v. 275, p. 314–347, 2014.

CHEN, M; WAN, J; LI, F. Machine-to-Machine Communications: Architectures, Standards and Applications. KSII Transactions on Internet and Information Systems, vol. 6,n. 2, p. 480-497, 2012.

CHODOROW, K. MongoDB: The Definitive Guide, 2nd Edition. O’Reilly Media, 2013.

CIARAMELLA, A.; CIMINO, M. G. C. A.; LAZZERINI, B.; MARCELLONI, F. Using context history to personalize a resource recommender via a genetic algorithm. 2010 10th International Conference on Intelligent Systems Design and Applications. p.965–970, 2010.

CONINCK, E. D.; BOHEZ, S.; LEROUX, S.; et al. Middleware Platform for Distributed Applications Incorporating Robots, Sensors and the Cloud. 2016 5th IEEE International Conference on Cloud Networking (Cloudnet). p.218–223, 2016.

COOK, D; DAS, S, K. Smart Environments: Technology, Protocols, and Applications. Wiley Inter-Science , 2004

CRESPO, S. Composição em WebFrameworks. Tese de doutourado em Ciencias em Informática, PUC-Rio, 2000.

DEY, A. K. Understanding and Using Context. Personal and Ubiquitous Computing, v. 5, n. 1, p. 4–7, 2001.

DRIVER, C.; CLARKE, S. An application framework for mobile, context-aware trails. Pervasive and Mobile Computing, v. 4, n. 5, p. 719–736, 2008.

EUGSTER, P. T.; FELBER, P. A.; GUERRAOUI, R.; KERMARREC, A.-M. The Many Faces of Publish/Subscribe. ACM Comput. Surv., v. 35, n. 2, p. 114–131, 2003.

FAYAD, M. E., SCHMIDT, D. C., and JOHNSON, R. E. Building application frameworks: object-oriented foundations of framework design. New York: John Wiley & Sons, 1999.

FAZIO, M.; CELESTI, A.; PULIAFITO, A.; VILLARI, M. Big Data Storage in the Cloud for Smart Environment Monitoring. Procedia Computer Science, v. 52, p. 500–506, 2015.

FORTINO, G.; TRUNFIO, P. (EDS.). Internet of Things Based on Smart Objects. Cham: Springer International Publishing, 2014.

GEISBERGER, E; BROY, M. Living in a networked world. Integrated research agenda cyber-physical systems (agendaCPS). Munich: Herbert Utz Verlag, 2015.

HONG, J.; SUH, E.-H.; KIM, J.; KIM, S. Context-aware system for proactive personalized service based on context history. Expert Systems with Applications, v. 36, n. 4, p. 7448–7457, 2009.

JESCHKE, S. Everything 4.0? - Drivers and Challenges of Cyber Physical Systems, Dec. 2013. Disponível em: <http://www.ima-zlw-ifu.rwth-

aachen.de/fileadmin/user_upload/INSTITUTSCLUSTER/Publikation_Medien/Vortraege/download//Forschungsdialog4Dez2013.pdf>. Acesso em: 17 fev. 2017

JOHNSON, R. E., FOOTE, B. Designing Reusable Classes. Journal of Object-Oriented Programming, Volume 1, Number 2. Department of Computer Science University of Illinois at Urbana-Champaign. pp. 22–35, 1988.

KOTEVSKA, O.; LBATH, A.; BOUZEFRANE, S. Toward a real-time framework in cloudlet-based architecture. Tsinghua Science and Technology, v. 21, n. 1, p. 80–88, 2016.

KOUBÂA, A.; ANDERSSON B. A vision of cyber-physical Internet, in Proc. Workshop RTN, pp. 1–6, Jul. 2009.

LEE, E. A. Cyber Physical Systems: Design Challenges. 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC). p.363–369, 2008.

LEE, E. A. The Past, Present and Future of Cyber-Physical Systems: A Focus on Models. Sensors, v. 15, n. 3, p. 4837–4869, 2015.

LEE, J. Y.; CHEUN, D. W.; KIM, S. D. A comprehensive framework for mobile cyber-physical applications. 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA). p.1–6, 2011.

MARTIN, D.; LAMSFUS, C.; ALZUA, A. Automatic context data life cycle management framework. 5th International Conference on Pervasive Computing and Applications. p.330–335, 2010.

MARTINS, M. V. L. FrameTrail: Um Framework para o Desenvolvimento de Aplicações Orientadas a Trilhas. Unisinos, São Leopoldo - RS. Dissertação, maio 2011.

MATTSSON, M. Object-Oriented Frameworks - A survey of methodological issues. 1996.

MÖLLER, D. P. F. Guide to Computing Fundamentals in Cyber-Physical Systems. Cham: Springer International Publishing, 2016.

NICOPOLITIDIS, P.; POMPORTSIS, A. S.; PAPADIMITRIOU, G. I.; OBAIDAT, M. S. Wireless Networks. New York, NY, USA: John Wiley & Sons, Inc., 2003.

NUNES, D. S.; ZHANG, P.; SILVA, J. S. A Survey on Human-in-the-Loop Applications Towards an Internet of All. IEEE Communications Surveys & Tutorials, v. 17, n. 2, p. 944–965, 2015.

OTEBOLAKU, A. M.; ANDRADE, M. T. User context recognition using smartphone sensors and classification models. Journal of Network and Computer Applications, v. 66, p. 33–51, 2016.

PASCOE, M. J. Adding Generic Contextual Capabilities to Wearable Computers. Proceedings of the 2Nd IEEE International Symposium on Wearable Computers. ISWC '98. p.92, 1998.

PATEL, K. K.; PATEL, S. M. Internet of Things - IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges. International Journal of Engineering Science and Computing, v. 6, n. 5, 2016.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology, v. 64, p. 1–18, 2015.

PETROULAKIS, N. E.; TRAGOS, E. Z.; FRAGKIADAKIS, A. G.; SPANOUDAKIS, G. A lightweight framework for secure life-logging in smart environments. Information Security Technical Report, v. 17, n. 3, p. 58–70, 2013.

POSTCAPES. Internet of Things (IoT) History. 2016. Disponível em: <https://www.postscapes.com/internet-of-things-history/>. Acesso em: 16 fev. 2017

ROSA, J. H.; BARBOSA, J. L. V; KICH, M.; BRITO, L. A Multi-Temporal Context-aware System for Competences Management. International Journal of Artificial Intelligence in Education, v. 25, n. 4, p. 455–492, 2015.

SATYANARAYANAN, M. Pervasive Computing: vision and challenges. IEEE Personal Communications, v. 8, p. 10-17, 2001.

SCHILIT, B. N.; THEIMER, M. M. Disseminating Active Map Information to Mobile Hosts. Netwrk. Mag. of Global Internetworking ., v. 8, n. 5, p. 22–32, 1994.

SCHILIT, B.; ADAMS, N.; WANT, R. Context-Aware Computing Applications. Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications. WMCSA ’94. p. 85–90, 1994.

SCHMID, H. A. Creating the Architecture of a Manufacturing Framework by Design Patterns. Proceedings of the Tenth Annual Conference on Object-oriented Programming Systems, Languages, and Applications. OOPSLA ’95. p.370–384, 1995.

SCHOLLMEIER, R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. Proceedings First International Conference on Peer-to-Peer Computing. p.101–102, 2001.

SEKHAR, P. K.; UWIZEYE, V. 2 - Review of sensor and actuator mechanisms for bioMEMS. In: BHANSALI, Shekhar; VASUDEV, Abhay (Eds.). {MEMS} for Biomedical Applications. Woodhead Publishing Series in Biomaterials, Woodhead Publishing. p. 46–77, 2012.

SILVA, J. M. UbiTrail: Um Modelo para gerenciamento de Trilhas Orientado a Ambientes Ubíquos. Unisinos Sao Leopoldo - RS. Dissertação, junho 2009.

SIMMON, E.; SOWE, S. K.; ZETTSU, K. Designing a Cyber-Physical Cloud Computing Architecture. IT Professional, v. 17, n. 3, p. 40–45, 2015.

STOJMENOVIC, I. Machine-to-Machine Communications With In-Network Data Aggregation, Processing, and Actuation for Large-Scale Cyber-Physical Systems. IEEE Internet of Things Journal, v. 1, n. 2, p. 122–128, 2014.

TALIGENT Inc., Building Object-Oriented Frameworks, A Taligent White Paper, 1994.

TARAZÓN, Rafa López. Chapter 28 - Robotics in Micro-manufacturing and Micro-robotics. In: QIN, Yi (Ed.). Micromanufacturing Engineering and Technology (Second Edition). Micro and Nano TechnologiesSecond Edition ed. Boston: William Andrew Publishing. p. 661–674, 2015.

TORUNSKI, E.; OTHMAN, R.; OROZCO, M.; SADDIK, A. EL. A Review of Smart Environments for Energy Savings. Procedia Computer Science, v. 10, p. 205–214, 2012.

TÖRNGREN, M.; ASPLUND, F.; BENSALEM, S.; et al. Chapter 1 - Characterization, Analysis, and Recommendations for Exploiting the Opportunities of Cyber-Physical

Systems. In: H. Song; D. B. Rawat; S. Jeschke; C. Brecher (Eds.); Cyber-Physical Systems, Intelligent Data-Centric Systems. p.3–14, 2017.

VEEN, J. S. VAN DER; WAAIJ, B. VAN DER; MEIJER, R. J. Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual. 2012 IEEE Fifth International Conference on Cloud Computing. p.431–438, 2012.

VERMESAN. O.; FRIESS P. Internet of Things—From Research and Innovation to Market Deployment. Aalborg, Denmark: River Publishers, 2014.

WEINAND, A.; WEIN, A.; GAMMA, E.; INC, T. ET++ - a Portable, Homogenous Class Library and Application Framework. Proceedings of the UBILAB ’94 Conference. 1994.

YICK, J.; MUKHERJEE, B.; GHOSAL, D. Wireless sensor network survey. Computer Networks, v. 52, n. 12, p. 2292–2330, 2008.

YOUNIS, O.; MOAYERI, N. Cyber-physical systems: A framework for dynamic traffic light control at road intersections. 2016 IEEE Wireless Communications and Networking Conference. p.1–6, 2016.

APÊNDICE A – FLAGS INTERNAS DO FRAMEWORK

Flags	Descrição
_EID	Designa o ID do evento interno
_SG	GUID do destinatário
_DT	Envio genérico de valores
_CN	Nome da coleção - Opcional
_OM	Modo de operação
_OMO	Opções do modo de operação

_EX	Parâmetro exclude, utilizado para exclusão de campos não desejados
_IN	Parâmetro Include, preserva o identificador interno
_ID	ID da entidade
_TK	Define o tipo de tarefa de consulta na base de dados
_UP	Campos e valores para atualização
_DF	Campos de retorno
_FL	Designa a lista de filtros utilizada
_CR	Utilizada no mecanismo de busca, listando os componentes (sensores e atuadores) a serem retornados.

Flags	**Tipos de filtro – Utilizados na lista de filtros (_FL)**
_f0	Filtro de equalidade
_f1	Filtro para elemento maior que outro
_f2	Filtro para elemento maior que outro, ou igual
_f3	Filtro para elemento menor que outro (*less than*)
_f4	Filtro para elemento menor que outro, ou igual (*less than or equal*)
_f5	Filtro para designar o tempo do contexto desejado
_f6	Define o operador OR dentro do filtro desejado

Modo de operação (Flags)	**Descrição**
Dt	Transmissão de dados comum
Sc	Designa um filtro de busca
SV	Define a função de armazenamento de contextos
GID	Solicitação pelo ID da entidade
AEP	Altera as propriedades da entidade
RC	Registrar componentes (sensores e atuadores)
CF	Cria um novo filtro
CGE	Troca a hierarquia de uma determinada entidade
GEC	Coleta o contexto da sentidades em hierarquia