UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

NAMEDIN PEREIRA TELES JÚNIOR

Êxodo Framework: Uma solução para modelagem do comportamento de ambientes para Internet das Coisas.

Manaus, Amazonas, Brasil
Fevereiro de 2018

NAMEDIN PEREIRA TELES JÚNIOR

Êxodo Framework: Uma solução para modelagem do comportamento de ambientes para Internet das Coisas

Tese submetida ao Programa de Pós-Graduação do Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para obtenção do título de Doutor em Informática.

Orientador: Prof. Dr. Eduardo James Pereira Souto

Manaus, Amazonas, Brasil
Fevereiro de 2018

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

T269ê

Teles Junior, Namedin Pereira
Êxodo Framework: Uma Solução para Modelagem do Comportamento de Ambientes para Internet das Coisas / Namedin Pereira Teles Junior. 2018
213 f.: il. color; 31 cm.

Orientador: Eduardo James Pereira Souto
Tese (Doutorado em Informática) - Universidade Federal do Amazonas.

1. Internet das Coisas. 2. Middleware. 3. Usabilidade. 4. Sistemas Distribuidos. I. Souto, Eduardo James Pereira II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Êxodo Framework: Uma solução para modelagem do comportamento de ambientes para Internet das Coisas."

NAMEDIN PEREIRA TELES JUNIOR

Tese de Doutorado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Eduardo James Pereira Souto - PRESIDENTE

Prof. Arilo Claudio Dias Neto - MEMBRO INTERNO

Prof. Vicente Ferreira de Lucena Junior - MEMBRO INTERNO

Prof. Rogério Patrício Chagas do Nascimento - MEMBRO EXTERNO

Prof. Rafael Roque Aschoff - MEMBRO EXTERNO

Manaus, 26 de Fevereiro de 2018

NAMEDIN PEREIRA TELES JÚNIOR

Êxodo Framework: Uma solução para modelagem dinâmica do comportamento de ambientes inteligentes para Internet das Coisas

Tese submetida ao Programa de Pós-Graduação do Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para obtenção do título de Doutor em Informática.

Banca Examinadora

Prof. Dr. Eduardo Pereira Souto (Orientador)
Universidade Federal do Amazonas (UFAM)

Prof. Dr. Arilo Cláudio Dias Neto
Universidade Federal do Amazonas (UFAM)

Prof. Dr. Vicente Ferreira de Lucena Junior
Universidade Federal do Amazonas (UFAM)

Rogério Patrício Chagas Nascimento
Universidade Federal de Sergipe (UFS)

Rafael Roque Aschoff
Instituto Federal de Pernambuco (IFPE)

Manaus, Amazonas, Brasil
Fevereiro de 2018

*Dedico essa tese a minha esposa **Nívia** e a minha mãe **Shirley** pela compreensão e paciência nos momentos que tive que me ausentar de tudo e dedicar todo tempo livre para elaboração desse trabalho.*

Agradecimentos

Agradeço primeiramente a Deus por ter me dado força e saúde para superar todos os obstáculos, me permitindo prosseguir e finalizar com êxito mais essa importante etapa da minha vida.

A minha esposa Nívia que sempre esteve ao meu lado me apoiando e incentivando a seguir em frente, sacrificando seu tempo livre em revisar a escrita dessa tese, apoiando e opinando sobre melhorias no projeto. Esteja certa de que essa tese carrega um pouco de você, muito obrigado.

A minha mãe que sempre me apoiou em todas as minhas decisões, me acompanhando e participando de cada momento, desde a graduação, mestrado e principalmente no doutorado, segurando a saudade nós momentos que tive que me ausentar e dedicar-me integralmente para conclusão dessa tese.

Ao meu Pai Tomé, meu maior exemplo de responsabilidade, perseverança e sabedoria, a pessoa que sempre acreditou e me aconselhou nos momentos que precisei.

Ao professor e orientador, Eduardo, por me orientar sabiamente, me pondo no caminho certo quanto ao desenvolvimento dessa tese. Pela confiança depositada quando precisei retornar ao mercado de trabalho e dividir meu tempo, antes dedicado integralmente a essa pesquisa, concordando em me orientar muitas vezes a distância, e por todo tempo a mim dedicado, muito obrigado ao meu orientador e amigo Eduardo Souto.

A minhas irmãs Nádia e Fernanda, meus irmãos Maxwell, Regnier e Guilherme, aos meus sobrinhos (as) que tanto amo Cesar, Yohan, Biel, Leonardo, beatriz, Julia e Laura.

Aos amigos que fiz enquanto na UFAM o qual não posso deixar de citar, Wesllen, Wesley, Dhanielly, que enquanto estive na UFAM foram meus amigos de toda hora, pessoas que sempre contei e acredito que sempre poderei contar, muito obrigado pelo apoio. E aos amigos do grupo ETSS.

À Universidade Federal do Amazonas, professores e funcionários que ajudam a manter a universidade sempre limpa e organizada para os alunos e pesquisadores.

A todos que fizeram dessa minha luta uma experiência de vida valiosa.

“Amo a liberdade, por isso deixo as coisas que amo livres. Se elas voltarem e porque as conquistei. Se não voltarem e porque nunca as possui. ”

JOHN LENNON (Pensamento)

Resumo

Internet das Coisas ou IoT (do Inglês, *Internet of Things*) é um paradigma da computação no qual dispositivos e objetos do mundo físico (e.g. televisões, geladeiras e condicionadores de ar) estão conectados por meio da Internet, provendo serviços e trocando informações entre si em uma escala global. Esse novo paradigma tem levado o mercado a apresentar soluções que cada vez mais aproxima a IoT do cotidiano das pessoas. Fabricantes como *Samsung*, *Philips*, *GE* e *Belkin* já dispõem de soluções de *hardware* (e.g. *smartphones*, *smart TV*, *smart plugs* e sensores) que permitem que os usuários interajam com ambiente, podendo ligar, desligar e até mesmo visualizar dados estatísticos do consumo de energia desses dispositivos. No entanto, as principais plataformas que permitem habilitar a IoT ainda são pouco utilizadas devido à sua complexidade, sendo restritas apenas a usuários especialistas, ou seja, a usuários que possuem conhecimento técnico em IoT. Além disso, as atuais ferramentas não permitem que usuários não especialistas possam adaptar essas soluções e criar ambientes para IoT de acordo com as suas necessidades. Nesse contexto, esse trabalho apresenta o *framework Êxodo*, uma nova abordagem para IoT com foco em usuário especialista e no usuário não especialista. O *Êxodo* é uma solução para IoT que pode ser estendida e adaptada para atender as diferentes necessidades do ambiente como a inclusão de novos dispositivo e serviços e/ou a definição de novos comportamentos ao ambiente. Para a definição dos diferentes comportamento que o ambiente poderá assumir, o Êxodo possui uma solução gráfica que faz uso de um modelo conceitual denominado BDM4IoT (*Behavior Definition Model for Internet of Things Ecosystems*) que possibilita que os usuários modelem o comportamento do ambiente de acordo com as suas necessidades, permitindo que esses possam criar desde modelos simples, como um esquema de iluminação de uma casa, a modelos mais complexos, como o comportamento de uma linha de produção de um ambiente industrial. Os resultados obtidos com experimentos realizados com o framework Êxodo demonstram a eficiência dessa abordagem, possibilitando a inclusão de usuários não especialistas em processos antes realizados somente por usuários especialistas em IoT.

Abstract

The Internet of Things or IoT is a paradigm of computing in which devices and objects of the physical world (e.g. televisions, refrigerators and conditioners air) are connected through the Internet, providing services and exchanging information between in a global scale. This new paradigm has led the market to present solutions that increasingly bring IoT closer to everyday life. Manufacturers such as Samsung, Philips, GE and Belkin already have hardware solutions (e.g. smartphones, smart TV, smart plugs and sensors) that allow users to interact with the environment, and can turn on, off and even view statistical data on power consumption these devices. However, the main platforms that allow IoT to be enabled are still little used due to their complexity, being restricted only to expert users, that is, users who have technical knowledge in IoT, not allowing non-expert users to adapt these solutions and create environments for IoT according to your needs. In this context, this work presents the Exodus framework, a new approach to IoT with a focus on expert user and non-expert user. Exodus is a solution for IoT that can be extended and adapted to meet the different needs of the environment such as the inclusion of new devices and services and / or the definition of new behaviors to the environment. In order to define the different behavior that the environment can take, the Exodus has a graphical solution that makes use of a conceptual model called BDM4IoT (Behavior Definition Model for Internet of Things Ecosystems) that together allow the users to model the behavior of the environment according to With their needs, allowing them to create from simple models such as a home lighting scheme to more complex models such as the behavior of a production line in an industrial environment. The results obtained with experiments carried out with the BDM4IoT model and with the graphical tool, demonstrate the efficiency of this approach, allowing the inclusion of non-expert users in processes previously performed only by IoT specialists.

Lista de Figuras

Lista de Tabelas

Abreviaturas

6LOWPAN - *IPv6 over Low power Wireless Personal Area Networks*

ALE - *Application Level Event*

APDL - *Aspire Process Description Language*

API - *Application Programming Interface*

ASPIRE - *Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise Application*

BDM4IoT - *Behavior Definition Model for Internet of Things EcoSystems*

BEG - *Business Event Generator*

BPMN - *Business Process Modeling Notation*

BPWME - *Business Process Workflow Management Editor*

CoAP - *Constrained Application Protocol*

DA - *Directory Agent*

DPWS - *Devices Profile for Web Services*

EBBITS - *Enabling Bussiness-Based Internet of Things and Services*

EcoDiF - *Ecossistema Web de Dispositivos Físicos*

EPC - *Electronic Product Code*

EPC-IS - *EPC Information Sharing*

EPC-LLRP - *EPC Low-Level Reader Protocol*

EPC-RP - *EPC Reader Protocol*

ESB - *Êxodo Service Bus*

F&C - Filtragem e Coleção

GPS - *Global Position System*

GSN - *Global Sensor Network*

GUI - *Graphical User Interface*

HAL - *Hardware Abstraction Layer*

HID - *Hydra Identification*

HTTP - *Hypertext Transfer Protocol*

IDE - *Integrated Development Environment*

IETF - *Internet Engineering Task Force*

IoT - *Internet of Things*

IP - *Internet Protocol*

IPV4 - *Internet Protocol version 4*

IPV6 - *Internet Protocol version 6*

ISM - *Industrial Scientific and medical*

JADI - *Java Agent Development Framework*

JMS - *Java Messaging Service*

JNI - *Java Notation Interface*

JSON - *JavaScript Object Notation*

JVM - *Java Virtual Machine*

M2M - *Machine-to-Machine*

MAC - *Media Access Control*

MQTT - *Message Queue Telemetry Transport*

NFC - *Near Field Communication*

OWL - *Ontology Web Language*

P2P - *Peer-to-Peer*

PDF – *Portable Document Format*

PLC - *Programmable Logic Controller*

RDF - *Resource Description Framework*

REST - *Representational State Transfer*

RFID - *Radio Frequency IDentification*

RSSF - Redes de sensores sem fio

SA - *Service Agent*

SDP - *Service Discovery Protocol*

SIoT - *Social Internet of Things*

SLP - *Service Location Protocol*

SM4ALL - *Smart Home for All*

SMS - *Short Message Service*

SMTP - *Simple Mail Transfer Protocol*

SOA - *Service-Oriented Architecture*

SOAP - *Simple Object Access Protocol*

SOCRADES - *Service-Oriented Cross-Layer Infrastructure for Distributed smart embedded devices*

TCP/IP - *Transmission Control Protocol/Internet Protocol*

UA - *User Agent*

UBIROAD - *Ubiquitous Road*

UML - *Unified Modeling Language*

UPnP - *Universal Plug and Play*

URI - *Uniform Resource Identifier*

UUID - *Universally Unique IDentifier*

Wi-Fi - *Wireless Fidelity*

XMII - *xManufacturing Integration and Intelligence*

XML - *eXtensible Markup Language*

YAML - *Ain't Markup Language*

SUMÁRIO

Capítulo 1 - Introdução

Atualmente, com a Internet cada vez mais presente no cotidiano das pessoas, acessível por computadores pessoais, *notebooks* ou até mesmo ao simples toque de tela de um *smartphone*, ocorreram mudanças na forma como as pessoas veem o mundo com relação às novas tecnologias de dispositivos que estão surgindo. Dispositivos eletrônicos como celulares e TVs, antes monofuncionais, são atualmente aparelhos multifuncionais, capazes não só de buscar informações, mas também de processar dados, além de conectar pessoas por meio da Internet.

Essa evolução tecnológica tem motivado o mercado a apresentar novas soluções de dispositivos cada vez mais atraentes e economicamente acessíveis às pessoas. Neste sentido, uma nova geração de dispositivos (por exemplo, sensores, atuadores, *smart TV* e *smart freezers*), denominados dispositivos inteligentes ou *Smart Devices*, tem surgido com a capacidade de sensoriar e interpretar os diversos eventos e mudanças que ocorrem no ambiente, como temperatura, luminosidade e movimento, e, com base nesses eventos, decidir como o ambiente deve se comportar, provendo uma maior comodidade às pessoas [Kortuem et al. 2010].

A principal vantagem quanto ao uso de dispositivo inteligente está na possibilidade de automatizar o ambiente e otimizar atividades antes executadas somente por usuários, como, por exemplo, ligar e desligar luzes e equipamentos eletrônicos, monitorar a temperatura e luminosidade do ambiente e, até mesmo, customizar o comportamento do ambiente de acordo com as diferentes necessidades de cada usuário.

Para melhor exemplificar as vantagens quanto ao uso de dispositivos inteligentes, imaginemos um **cenário ilustrativo –** um modo mais pedagógico possível para se descrever a adoção de dispositivos inteligentes em um ambiente doméstico –, conforme o seguinte:

- uma casa composta de objetos inteligentes, como uma geladeira que seja capaz tanto de verificar a falta de um produto essencial, quanto de fazer o pedido desse mesmo produto a um supermercado credenciado por meio da Internet, sem que haja intervenção humana;

- Nessa casa, todos os cômodos são constantemente monitorados por sensores, capazes de colher informações do ambiente como: temperatura, movimento, consumo de água, energia e gás; e propor ao usuário soluções para economia desses recursos;
- Além disso, sensores e atuadores são também utilizados para monitorar constantemente a saúde dos moradores e capazes de acionar a emergência caso necessários.

Agora, imaginemos um pouco mais além, esse cenário em uma escala global, nos hospitais, trânsito, indústria, comércio e, até mesmo, nas cidades, em que os dispositivos interagem uns com os outros para prover maior segurança e comodidade às pessoas. Esse ambiente global de dispositivos e objetos interconectados é conhecido como Internet das Coisas ou IoT do Inglês (*Internet of Things*).

IoT, de acordo com Mattern, Floerkemeier [2010] e Zhang, Yang, Huang [2011], é um paradigma no qual dispositivos e objetos do mundo real podem trocar informações e serviços por meio da Internet, possibilitando a otimização de ambientes e o controle dos recursos, acessíveis de qualquer lugar, a qualquer momento e de qualquer coisa.

São grandes as vantagens que a IoT pode proporcionar, tanto em aspectos tecnológicos com a inclusão de novas tecnologias em dispositivos, quanto em aspectos econômicos com o monitoramento de recursos e bens de consumo, como água, luz e gás [Evans 2011].

No entanto, para que a IoT possa se tornar uma realidade no cotidiano das pessoas, bem como em um contexto global, existem alguns importantes desafios a serem superados, dentre os quais pode-se citar a heterogeneidade existente nos dispositivos criados atualmente para atender ao paradigma da IoT, que, em sua grande maioria, são criados por diferentes fornecedores, utilizando diferentes tecnologias e protocolos, o que dificulta e, até mesmo, impossibilita que esses dispositivos possam interoperar.

Escalabilidade é outro importante desafio a ser superado para que a IoT possa acompanhar a evolução e surgimento de novos dispositivos que poderão emergir no

futuro [Xu, He, Li 2014] e [Miorandi et al. 2012]. Outros desafios serão apresentados e melhor explorados mais adiante, no Capítulo 3 desta tese.

Na literatura, os principais desafios enfrentados pela IoT são tratados por meio de camadas de software (*middlewares*). Segundo Atzori, Iera e Morabito [2010], *middleware* é uma camada ou um conjunto de subcamadas sobrepostas entre os dispositivos e o usuário, uma vez que que permite esconder detalhes sobre tecnologia, isentando os usuários de questões que não estão diretamente relacionadas ao seu foco, provendo meios mais simples para acesso aos dispositivos e serviços.

Dessa forma, nos últimos anos, *middlewares* têm estado cada vez mais presentes no desenvolvimento de novas soluções e serviços para IoT. Entretanto, existem problemas que precisam ser resolvidos para que tais soluções possam ser melhor aproveitadas. A seção a seguir apresenta os problemas que se pretendem abordar neste trabalho.

1.1. Problemas

As atuais tendências de mercado têm mudado a forma como os usuários interagem com os sistemas e dispositivos, pois esses possuem serviços e funções que permitem que os usuários possam personalizar e customizar os sistemas e dispositivos de acordo com seus perfis e necessidades. Como exemplo, podem ser citados os *smart phones* e *smart TVs*, que atualmente permitem aos usuários não só configurar os dispositivos de acordo com suas preferências e necessidades, como também de instalar e desinstalar aplicativos.

Assim, acredita-se que há uma tendência de a IoT seguir nessa direção, ou seja, permitir que os usuários possam adaptar/customizar seus próprios ambientes conforme suas necessidades. Entretanto, a maioria das soluções de *middleware* para IoT possuem limitações que inviabilizam a IoT seguir atualmente essa tendência de mercado.

De um modo geral, as limitações existentes nas atuais soluções de *middleware* para IoT se devem a dois fatores:

1. A maioria das soluções de *middleware* para IoT são criadas para atender a domínios específicos, não permitindo a modelagem de domínios para os quais não foram projetadas. Por exemplo: para o domínio casas se tem o

middleware SM4ALL [Warriach et al. 2014]; já para o domínio industrial, o *middleware SOCRADES* [De Souza et al. 2008]; enquanto para domínios públicos, o *middleware UBIROAD* [Terziyan, Kaykova, Zhovtobryukh 2011]; e, para o domínio de transporte e logística, o *middleware* ASPIRE [Anggorjati et al. 2010]; e

2. As soluções existentes que permitem ser estendidas e adaptadas são de difícil entendimento e utilização, tanto por usuários especialistas quanto não especialistas, como, por exemplo, o *middleware LinkSmart* [Zhang e Hansen 2008], GSN [Aberer e Hauswirth 2006] e *UBIWARE* [Katasonov et al. 2008], os quais são soluções projetadas para serem estendidas e utilizadas em diferentes domínios, mas necessitam de que o usuário possua conhecimento elevado nas principais tecnologias que habilitam a IoT como protocolos, dispositivos e tecnologias de comunicação (*bluetooth*, *zigbee* e *wifi*).

Dado o atual cenário das soluções de *middleware* existentes, as seguintes questões precisam ser respondidas:

1. Como possibilitar que usuários especialistas e não especialistas possam estender uma solução de middleware para IoT sem que, para isso, sejam necessários conhecimentos aprofundados das principais tecnologias que compõem o paradigma da IoT?
2. Como prover uma solução para IoT que possibilite que tais usuários possam criar, modelar, manipular e gerenciar seus próprios ambientes para IoT, adaptando-os às suas necessidades diárias?

Para fornecer maior flexibilidade à arquitetura de uma solução de *middleware* para IoT, possibilitando ao usuário estender, modificar e adaptar a solução sem que seja necessário adquirir conhecimentos profundos dos níveis mais baixos da arquitetura, é possível fazer uso de padrões de projeto (do Inglês, *Design Patterns*) [Freeman 2007], estrategicamente aplicados em partes da arquitetura, possibilitando que essa possa ser pontualmente modificada, sem que haja necessidades de modificações em diversas partes da arquitetura. A utilização de padrões de projeto permite modularizar partes da arquitetura, encapsulando do usuário toda complexidade da arquitetura, dispondo ao usuário um conjunto de classes extensível para permitir que a arquitetura possa ser adaptada a necessidades específicas do usuário.

Para possibilitar a inclusão de usuários com pouco ou quase nenhum conhecimento em computação no processo criativo e gerencial de ambientes para IoT, é possível fazer uso de um importante conceito da engenharia de S*oftware* denominado **modelagem do comportamento** [Booch, Rumbaugh e Jacobson 2000]. Tal abordagem consiste em utilizar modelos conceituais, como, por exemplo, o modelo da UML[1] (*Unified Modeling Language*) e o BPMN [2] (*Business Process Modeling Notation*), provendo meios mais simples e de fácil compreensão para permitir que esses usuários possam realizar atividades complexas, aquelas executadas somente por usuários especialistas.

Logo, o uso de modelos conceituais em soluções para IoT possibilitaria que usuários não especialistas pudessem modelar todo o comportamento de um ambiente para IoT, sem a necessidade de adquirir conhecimentos avançados em computação, mas somente conhecimentos sobre a utilização do modelo conceitual.

1.2. Objetivo Principal

Com base nos problemas expostos, este trabalho tem como objetivo principal o desenvolvimento de um *framework* para modelagem de comportamento de ambientes para IoT, utilizando padrões de projeto para definir pontos específicos para extensão e adaptação de arquitetura e modelos conceituais, a fim de possibilitar que os usuários possam modelar o comportamento de seus próprios ambientes para IoT de acordo com suas necessidades.

A meta é possibilitar que o *framework* forneça serviços a dois tipos de usuários: especialista e não especialista. Para o usuário especialista, o *framework* proposto fornecerá soluções que lhe permitam modelar os mais diversos domínios, com uma arquitetura de fácil utilização e com componentes extensíveis, a fim de que esses usuários possam adaptar e ajustar as soluções para atender a diferentes domínios, como casas, indústria, locais públicos, entre outros. Para os usuários não especialistas, o *framework* deve permitir que estes possam modelar, monitorar e gerenciar o comportamento do ambiente por meio de uma interface gráfica amigável e de fácil utilização.

[1] http://www.uml.org/
[2] http://www.bpmn.org/

Ademais, para se atingir o objetivo principal desta tese, um conjunto de resultados intermediários deve ser alcançado, cada qual contribuindo como um elemento auxiliar em busca da meta final pretendida:

1. Definir um modelo arquitetural flexível para o *framework* com base em padrões de projeto aplicados em partes estratégicas da solução, permitindo que essa possa ser estendida de acordo com as necessidades dos usuários especialistas;
2. Definir um modelo conceitual para auxiliar os usuários não especialistas na modelagem de comportamento de seus próprios ambientes para IoT;
3. Criar uma solução de *CORE* para gerenciamento dos dispositivos, eventos, serviços e principais processamentos a serem realizados pelo *framework*;
4. Criar uma solução extensível para comunicação do *framework* com os dispositivos e serviços do mundo físico;
5. Criar uma solução gráfica para possibilitar que usuários não especialistas possam manipular o modelo de comportamento proposto, a fim de projetar diferentes ambientes para IoT; e
6. Criar uma solução centralizada para a integração das soluções propostas, permitindo que essas possam trocar informações e serviços entre si.

1.3. Contribuições

A maioria das soluções de *middleware* para IoT estudadas neste projeto são complexas e, consequentemente, difíceis de serem utilizadas por usuários especialistas e praticamente inviáveis a usuários não especialistas. Pretende-se, após o desenvolvimento desse projeto, obter as seguintes contribuições:

- Um modelo conceitual para modelagem de comportamento de ambientes para IoT denominado BDM4IoT, para permitir a inclusão do usuário não especialista no processo criativo de ambientes para IoT. Diferentemente dos modelos existentes para IoT, como em [Song, Cárdenas e Masuoka 2010], [Anggorjati et al. 2010] e [De Souza 2008], o modelo BDM4IoT possibilitará que usuários não especialistas possam customizar diferentes tipos de ambiente, desde ambientes simples, como o doméstico, até os modelos mais complexos, como ambientes industriais, utilizando, para isso, um conjunto de objetos auxiliares e de regras

de negócio, como operadores *AND*, *OR*, estrutura de decisão *IF*, repetição, entre outros;

- Uma linguagem de marcação denominada BDML (*Behavior Definition Markup Language*), projetada para permitir a criação, validação e portabilidade de modelos conceituais para IoT, assim como o modelo BDM4IoT;
- Uma arquitetura flexível para possibilitar que usuários especialistas possam alterar e estender a solução, a fim de melhor adaptá-la às suas necessidades;
- Uma solução gráfica criada para auxiliar os usuários não especialistas na manipulação e configuração dos objetos que compõem o modelo BDM4IoT; e
- Um *framework Open Source* flexível com foco no usuário denominado *Êxodo*, composto por soluções para permitir a modelagem, monitoração e o gerenciamento de ambientes para IoT por usuários especialistas e não especialistas.

1.4. Organização da Tese

Este trabalho está organizado em seis (6) capítulos, sendo o **Capítulo 1** dedicado a apresentar a introdução, problemas e o contexto da pesquisa a ser realizada. O restante deste texto segue a seguinte estrutura:

- **Capítulo 2 – Fundamentação:** este capítulo apresenta a fundamentação básica sobre os principais conceitos relacionados a Internet das Coisas, suas diferentes visões, arquitetura e sua aplicabilidade em diferentes domínios;
- **Capítulo 3 – Plataformas para Internet das Coisas:** já este capítulo busca apresentar o estado da arte de *middleware* para IoT, apresentando as principais soluções de *middleware* para IoT existentes na literatura, os principais desafios enfrentados, tecnologias utilizadas e uma avaliação das soluções quanto ao atendimento dos desafios e tecnologias apresentadas;
- **Capítulo 4 – O Modelo de Comportamento BDM4IoT**: Em tal capítulo, será exposto o modelo BDM4IoT, especialmente criado para possibilitar a inclusão do usuário não especialista no processo criativo de comportamentos de ambientes para IoT;
- **Capítulo 5 – BDML (*Behavior Definition Markup Language*)**: de forma dinâmica, pretende-se expor neste capítulo a linguagem de marcação BDML,

criada para ser utilizada como base de desenvolvimento do modelo BDM4IoT, permitindo que esse possa ser facilmente validado e portado por meio de dispositivos de armazenamento, além de permitir que o modelo possa ser facilmente modificado e estendido;

- **Capitulo 6 – Êxodo *Framework*:** neste item, pretende-se trazer a lume o *Framework Êxodo*, uma solução criada para permitir que usuários com diferentes perfis de conhecimento possam manipular os dispositivos presentes no ambiente, com a finalidade de modelar os comportamentos que deverão ser executados no ambiente: Primeiramente, será apresentada a arquitetura da solução e seus principais componentes. Em seguida, a solução gráfica *Êxodo GUI*;
- **Capitulo 7 – Experimentos e Avaliações:** neste capítulo, serão exibidas a condução e avaliação dos experimentos realizados, cujo objetivo é analisar a expressividade, o uso do modelo BDM4IoT por usuários com diferentes perfis e a utilização da ferramenta Êxodo GUI, para que se obtenham dados sobre a experiência e opinião dos usuários a respeito das soluções desenvolvidas; e
- **Capítulo 8 – Conclusões e Trabalhos Futuros:** por meio deste capítulo, este discente irá demonstrar e reafirmar tudo o que foi investigado e discutido no decorrer da presente tese, de forma concisa e judiciosa, buscando-se, sobretudo, ilustrar não só as conclusões sobre as abordagens utilizadas e resultados obtidos com os experimentos realizados, ainda também apresentar, em seguida, uma lista de trabalhos e correções futuras que precisarão ser realizadas para que a solução proposta possa se tornar viável na prática.

Capítulo 2 – Internet das Coisas

Devido aos avanços tecnológicos nos últimos anos, tem se tornado cada vez mais comum a presença de dispositivos inteligentes no cotidiano das pessoas. Essa presença tem aproximado cada vez mais a IoT de usuários comuns, permitindo a esses controlar os dispositivos por meio da Internet com um simples “toque de tela” de um celular. Contudo, a IoT é mais do que apenas controlar objetos inteligentes. Existem importantes conceitos que precisam ser entendidos para saber o que realmente é a IoT e quais a vantagens que esse novo paradigma proporcionará no futuro.

Em vista disso, será apresentada uma visão geral sobre os principais conceitos e fundamentos que norteiam a IoT, desde os aspectos históricos, que levam a origem do termo, até os conceitos importantíssimos como visões, arquitetura e aplicabilidades. Assim, a organização da ideia segue os seguintes parâmetros: primeiramente, serão apresentados os principais conceitos e visões disponíveis na literatura com relação à IoT; em seguida, a arquitetura genérica da IoT. Por fim, serão discutidas as aplicabilidades da IoT em diferentes domínios do mundo real.

2.1. Conceitos e Visões

Nos últimos anos, a Internet das Coisas, ou simplesmente IoT, tem recebido grande atenção por parte da comunidade acadêmica e da indústria, sendo considerada um paradigma promissor que mudará a forma de interação das pessoas com o mundo por meio dos objetos e dispositivos do cotidiano, permitindo criar ambientes altamente dinâmicos e automáticos para melhor atender as necessidades das pessoas [Perera et al. 2014].

Apesar da recente atenção, o termo "*Internet of Things*" não surgiu de um fato isolado, mas da evolução e “casamento” de importantes conceitos e tecnologias já existentes, como a Internet, protocolos de comunicação, redes de sensores e atuadores, miniaturização dos dispositivos eletrônicos e a evolução das interfaces sem fio. Contudo, essa miríade de tecnologias, que dá origem ao termo, dificulta a definição de

um conceito único sobre o que realmente é a IoT, visto que vários pesquisadores apresentam diferentes definições para a IoT.

Bassi e Horn [2008] descrevem que a IoT é composta por dois conceitos chaves: a Internet e as “coisas”. A Internet pode ser definida como uma rede mundial de computadores interligados baseada em protocolos de comunicação e padrões. As “coisas” referem-se a objetos unicamente identificados, capazes de se comunicar por meio da Internet para trocar informações e serviços. Portanto, IoT é uma rede mundial de objetos, com identidade única, interconectados com base em protocolos de comunicação padrão.

Para Tan e Wang [2010], a IoT é definida como uma rede composta de coisas que têm identidade e personalidades virtuais, operando em espaços inteligentes, utilizando interfaces e protocolos para se conectar e comunicar dentro de diferentes contextos, tais como: sociais, ambientais e de usuário.

Vermesan et al. [2011] descreve um conceito mais abrangentes para a IoT, conforme segue:

> “ Internet das Coisas é uma infraestrutura de rede global e dinâmica, com capacidade de autoconfiguração baseada em padrões e protocolos de comunicação interoperáveis, onde "Coisas" físicas e virtuais possuem identidades, atributos físicos, personalidades virtuais e interfaces inteligentes perfeitamente integradas em uma rede de informação. ”

Esses diversos conceitos têm direcionado os pesquisadores a fornecerem diferentes visões sobre a IoT, tanto as que são baseadas no desenvolvimento de novas tecnologias, como por exemplo, o surgimento de novos dispositivos, padrões e protocolos [Weiser 1991] e [Guinard e Trinfa 2009], ou aquelas visões fundamentadas em modelos sintáticos, como em De et al. [2011], que apresenta os principais conceitos da IoT como entidades, recursos, serviços, dispositivos e as relações existentes entre esses.

Com objetivo de unificar essas diferentes visões em um único modelo, Atzori, Iera e Morabito [2010] propuseram um modelo com base em três visões: visão orientada às “coisas”, visão orientada à “*Internet*” e visão orientada à “semântica”, conforme pode ser observado mais adiante, ilustrado por meio da Figura 2.1. Adicionalmente, é importante destacar que, além dessas visões, os autores descrevem outros importantes

conceitos que surgem da interseção entre as diferentes visões do modelo, tais como o conceito de conectividade para qualquer coisa, existente entre as visões “coisas” e “Internet”. No entanto, como não foram previstos conceitos e/ou tecnologias na interseção entre “coisas” e “semântica”, foi inserido no modelo de visões (Figura 2.1) três conceitos que permitem a inclusão de usuários não especialistas no contexto da IoT:

1. **Ferramentas gráficas** capazes de permitir que usuários não especialistas possam facilmente manipular o ambiente, bem como os dispositivos nele presente.
2. **Modelos** conceituais de comportamento, considerados nesse trabalho como fundamentais para inclusão de usuários não especialistas no processo criativo do comportamento de ambientes; e
3. **Virtualização** de objetos do mundo real, a fim de permitir que esses possam ser virtualmente monitorados e manipulados por quaisquer usuários.

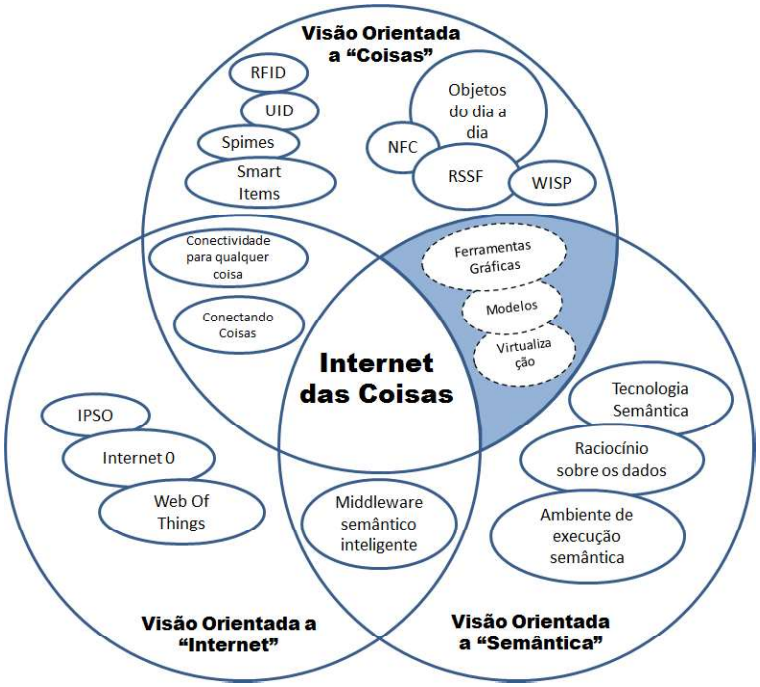


Figura 2.1. As diferentes visões da IoT. Adaptado de Atzori, Iera e Morabito [2010].

O modelo de visões agrupa os principais conceitos e tecnologias existentes ao redor da IoT, distribuídos entre as diferentes visões. A visão orientada às "Coisas" é um resumo das principais tecnologias de dispositivos que permitem tornar a IoT uma realidade. Dentre essas, a que mais se destaca, segundo Perera et al. [2014] são as redes de sensores sem fio (RSSF). No contexto de IoT, redes de sensores e atuadores são

comumente empregadas para detecção de eventos do mundo real e adaptação do ambiente de acordo com as necessidades dos usuários [Christin et al. 2009 e Mainetti, Patrono e Vilei 2011].

Na visão orientada à "*Internet*", há a representação das principais tecnologias, protocolos e conceitos utilizados para permitir que dispositivos e/ou "Coisas" possam se conectar à Internet, com o propósito de prover ou consumir serviços. Atualmente, a principal tecnologia empregada para esse fim é o uso do protocolo IPV6 (*Internet Protocol version 6*) e protocolos IPV6 modificados. Como exemplo, o 6LOWPAN (*IPv6 over Low power Wireless Personal Area Networks*) é utilizado para permitir que dispositivos com baixo poder computacional e consumo de energia possam ser endereçados e disponibilizados na Internet [Montavont, Roth e Noël 2014]. Em alguns casos, o protocolo IPV4 (*Internet Protocol version 4*) também é empregado para permitir a conectividade de objetos por meio da Internet. Entretanto, devido à sua limitação de endereço, atualmente o protocolo IPV6 tem estado em destaque como principal tecnologia para endereçar objetos/dispositivos.

Por outro lado, na visão orientada à "semântica", são representadas as principais linguagens para construção de modelos semânticos. De acordo com De Souza [2008], modelos semânticos são criados para definir a relação entre termos a fim de dar significado de algo ou alguma coisa. Na IoT, modelos semânticos são frequentemente utilizados na concepção de *middlewares* semânticos e/ou no desenvolvimento de importantes funcionalidades, com o uso de ontologias para permitir a automatização de processos, como a descoberta de dispositivos [Zhang e Hansen 2008] ou para representação abstrata de tecnologias [Song, Cárdenas e Masuoka 2010]. Atualmente, as principais linguagens semânticas utilizadas na IoT são: OWL (*Ontology Web Language*) e a RDF (*Resource Description Framework*), que serão melhor especificadas no Capítulo 3.

O modelo de visão de Atzori, Iera e Morabito [2010], é uma importante contribuição na literatura para IoT, pois fornece um conjunto de diferentes visões sobre os principais conceitos e tecnologias que constituem o paradigma da IoT, permitindo visualizar exatamente qual seu real significado e quais os conceitos e tecnologias que abrangem esse novo paradigma. No entanto, além dessas visões, é importante se ter uma visão arquitetural sobre a IoT, a fim de conhecer as principais camadas e componentes

que formam esse novo paradigma. A Figura 2.2, abaixo, apresenta um modelo arquitetural genérico com as cinco principais camadas que compõem a IoT:

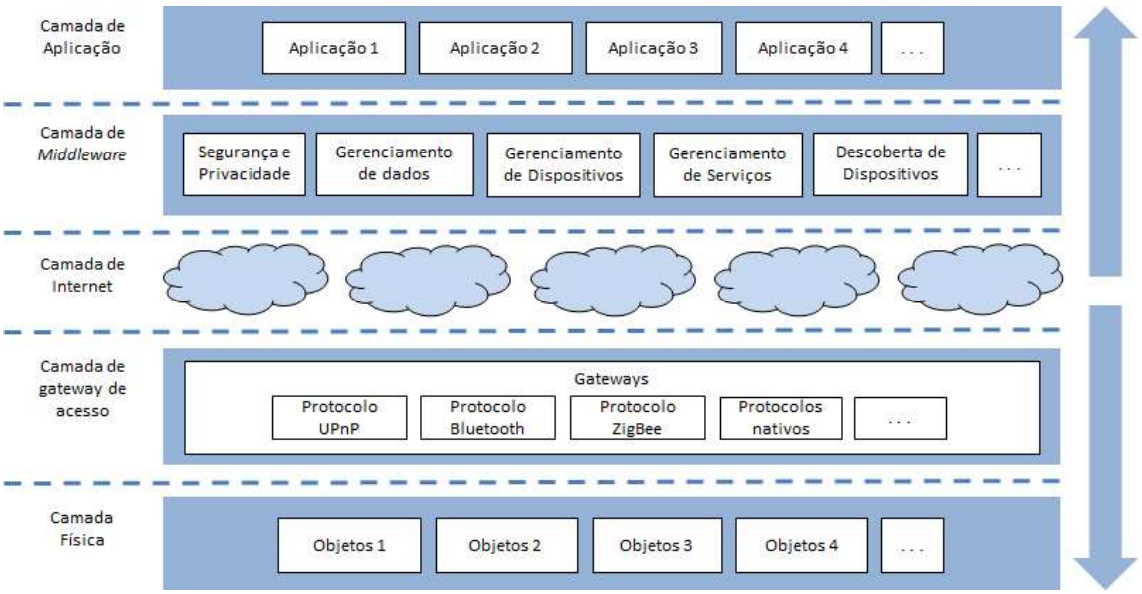


Figura 2.2 Modelo arquitetural genérico para IoT.

A camada física representa os dispositivos e objetos físicos existentes no ambiente, tais como: televisão, telefones, condicionadores de ar, sensores de movimento e, até mesmo, objetos pessoais, como roupas, joias e sapatos. Tais dispositivos são capazes de se conectar à Internet utilizando geralmente diferentes interfaces de comunicação sem fio, como *Wi-Fi* (*Wireless Fidelity*), *Bluetooth*, *Zigbee*, *RFID* (*Radio Frequency IDentification*), entre outros.

A camada de gateway de acesso representa os protocolos das principais interfaces de comunicação, as quais permitem que um dispositivo possa trocar de informações e disponibilizar serviços. Como exemplo, o protocolo UPnP (*Universal Plug and Play*), que é usado largamente em solução para permitir que dispositivos possam ser descobertos e configurados automaticamente em uma rede; os protocolos *Bluetooth* e *ZigBee*, que recentemente foram adaptados para utilizar o protocolo IPV6 (6LowPAN) [Lu, Li e Wu 2011; Siekkinen et al. 2012; Wang et al. 2013 e Chang 2014]; e protocolos proprietários, criados para controlar interfaces de fornecedores específicos.

A camada de Internet é responsável por prover conectividade global aos dispositivos, permitindo que esses possam ser acessados e gerenciados de qualquer

lugar e a qualquer momento, além de possibilitar que esses possam trocar informações e disponibilizar serviços em uma escala global.

A camada de *middleware* fornece um conjunto de serviços para as aplicações como os serviços de descoberta de dispositivos, segurança e privacidade, gerência de contexto, gerência de dados, gerência de dispositivos, entre outros. Além disso, essa camada pode fornecer serviços críticos, como agregar e filtrar dados e prover controle de acesso aos dispositivos e aplicações.

Por fim, a camada aplicação é responsável por permitir que aplicações possam acessar os dados e serviços gerenciados pelo *middleware*. Essa camada também permite que aplicações externas possam prover serviços na rede, assim como os dispositivos.

2.2. Áreas de Aplicação

As potenciais vantagens oferecidas pela IoT tornam possível o desenvolvimento de um grande número de aplicações, das quais apenas uma parte muito pequena está atualmente disponível para sociedade. São muitos os domínios e ambientes que se beneficiarão com a IoT, como casas, escritórios, saúde dos usuários, entre outros. Atualmente, grande parte desses ambientes é equipada com objetos com inteligência primitiva, sem quaisquer capacidades de comunicação. Assim, dando a esses objetos a possibilidade de se comunicarem uns com os outros resultaria em ambientes nos quais uma variedade de aplicações pode ser implementada [Atzori, Iera e Morabito 2010]. Sendo assim, as seções a seguir destinam-se a apresentar esforços iniciais disponíveis na literatura quanto ao uso de soluções para IoT para automatização de diferentes domínios e ambientes, como segue:

Transporte e Logística

Nesse domínio, tecnologia de processamento de informações em tempo real, baseado em RFID, NFC (*Near Field Communication*) e sensores, permite realizar o monitoramento em tempo real de quase todos os elos de uma cadeia de abastecimento, desde a compra de matéria prima, produção, transporte, armazenamento, distribuição e venda, permitindo ao usuário consultar, em tempo real, as informações e o estado dos produtos [Atzori, Iera e Morabito 2010].

Para avaliar a flexibilidade e versatilidade do uso de soluções de *middleware* para IoT no contexto de transporte e logística, Kefalakis et al. [2008] propõem o *middleware* ASPIRE para automatização dos processos de rastreabilidade de produtos e gerenciamento de cadeias de suprimentos. Para isso, foram elaborados cenários utilizando sensores, etiquetas RFID e NFC, GPS (*Global Position System*), a fim de capturar o máximo de informações sobre o ambiente, como temperatura, localização dos objetos, informações sobre os objetos, entre outros. Os resultados obtidos demonstram que o ASPIRE pode suportar uma ampla gama de implementações existentes no domínio de transporte e logística, como a gestão de ativos, rastreabilidade e controle de acesso.

Além do ASPIRE, existem outras soluções na literatura que podem ser usadas para automatização de domínios de transporte e logística. Brizzi et al. [2013] apresentam uma visão geral sobre o projeto EBBITS, suas principais características e o seu uso aplicado na automatização de um processo para rastreabilidade de alimentos. Nesse cenário, todos os animais e alimentos, como vacas, porcos, galinhas e rações, são exclusivamente identificados com etiquetas RFID, possibilitando o acompanhamento de cada fase do processo de produção, como alimentação, matadouro e varejo, garantindo maior qualidade dos alimentos produzidos.

Considerando ainda o transporte inteligente, Katasonov et al. [2008] apresentam uma interessante solução para automatização desse tipo de ambiente, denominado de UBIWARE. Esse *middleware* que faz uso de agentes inteligentes e modelos semânticos, para permitir a interoperabilidade e gerenciamento de sensores e atuadores heterogêneos. Nessa solução, modelos semânticos são utilizados para definir todo comportamento da própria solução, bem como os papéis e comportamento dos agentes. Em Terziyan, Kaykova e Zhovtobryukh [2009], a arquitetura do middleware UBIWARE é estendida para o desenvolvimento de uma solução para gerenciamento e controle de dispositivos embutidos em veículos e estradas. A ideia é permitir que os veículos possam se comunicar, reduzindo acidentes e infrações diretamente relacionadas à falha humana.

Assistência Médica

No contexto de assistência médica, sensores sem fio têm sido embutidos em acessórios, como roupas e braceletes, com o propósito de permitir que o próprio paciente possa

monitorar sua saúde (pressão sanguínea, batimentos cardíacos, temperatura corporal, entre outros), utilizando como interfaces dispositivos móveis (*smartphone*, *tablets* e *smart watch)* [Maia et al. 2015] e [Yuce 2013].

Maia et al. [2015] apresentam o projeto *EcoHealth*, um *middleware* para permitir a integração e gerenciamento de dispositivos corporais heterogêneos. O *middleware* foi criado para conectar médicos e pacientes utilizando sensores corporais e atuadores para fins de monitoramento, processamento, visualização e armazenamento de dados sobre a saúde do paciente, bem como permitir a notificação e atuação referentes às suas condições atuais e seus sinais vitais. *EcoHealth* é uma solução criada a partir de um projeto maior denominado *EcoDiF* [Delicato et al. 2013], uma plataforma web genérica para IoT, cujo principal objetivo é integrar dispositivos físicos heterogêneos.

Eikerling et al. [2009] discutem a utilização do *middleware Hydra* [Eisenhauer, Rosengren e Antolin 2010], atualmente denominado como *LinkSmart,* no setor de saúde por meio de um sistema de gestão de doenças, devido à sua capacidade para integração e configuração automática de dispositivos heterogêneos. Para fins de comprovações, foram utilizados dispositivos reais, como um medidor de pressão com interface *bluetooth* para transmissão de dados, um notebook com interface *bluetooth* utilizado como *gateway* e um servidor para processamento e armazenamento dos dados. Os resultados obtidos demonstram a facilidade do uso do *LinkSmart* na integração de dispositivos heterogêneos em ambientes de assistência à saúde.

Automação de Ambientes

O desenvolvimento tecnológico e o aumento do número de produtos de TI possibilitaram a criação de ambientes inteligentes em diferentes áreas, como casas, escritórios, indústria e locais públicos (e.g. academias, praças e shoppings).

No contexto industrial, Cannata, Gerosa e Taisch [2008] apresentam o *middleware* SOCRADES, que implementa tecnologias e ferramentas para permitir a modelagem, projeto, implementação e operação de sistemas de redes industriais compostas por dispositivos inteligentes, que utilizam tanto interfaces cabeadas quanto sem fio. De acordo com os autores, as vantagens proporcionadas quanto ao uso do SOCRADES incluem a redução das atividades de produção, de custos com produção, maior visibilidade, de erros humanos, entre outros.

No contexto de automação de casas e escritórios, Warriach [2013] apresenta o middleware SM4ALL criado exclusivamente para automatização de ambiente domiciliar. SM4ALL possibilita que dispositivos domésticos inteligentes e heterogêneos possam interoperar por meio de diferentes interfaces de comunicação, além de possuir um conjunto de ferramentas, tais como comandos de voz, interfaces cérebro máquina, dispositivos móveis, entre outros, para permitir que os usuários possam interagir com o ambiente. SM4ALL é uma solução que possui uma variedade de dispositivos de entrada para permitir que diferentes usuários como idosos e deficientes possam interagir com o ambiente.

Hosek et al. [2014] fazem uso de uma plataforma, denominada como *openHAB*, para facilitar a integração universal entre todos os dispositivos inteligentes existentes em ambientes domésticos como TVs, geladeiras, condicionadores de ar, sensores, entre outros. Por ser uma solução puramente Java, *openHAB* pode ser adaptado a soluções que fazem uso da linguagem Java, além de permitir que possa ser executado em qualquer plataforma compatível com a JVM.

Song, Cárdenas e Masuoka [2010] apresentam o *Task Computing*, uma solução de *middleware* para automatização de pequenos ambientes como escritórios e locais públicos de pequenas proporções. *Task Computing* possibilita integrar dispositivos inteligentes heterogêneos que fazem uso dos protocolos UPnP e *Bluetooth*. Além disso, a principal vantagem do *Task Computing* é permitir que os próprios usuários possam definir como uma determinada tarefa será realizada. Por exemplo, um usuário pode selecionar uma música em formato MP3 de seu repositório de música, direcionando-a para tocar em um alto-falante *bluetooth* de sua preferência e obter informações sobre a canção, artista e o CD por meio de serviços web disponibilizados por diferentes fornecedores como o *Amazon Web Service*.

Em um contexto mais amplo, como cidades, praças e shoppings, onde há uma grande quantidade de serviços providos por sensores, é possível fazer uso do *middleware* GSN (do Inglês, *Global Sensor Network*), uma solução *Open Source* projetada para o gerenciamento de redes de sensores. Em Perera et al. [2012] GSN, é utilizado para capturar dados provenientes dos sensores existentes nos telefones celulares, a fim de obter informações sobre o ambiente e sobre o dia a dia dos usuários. Em Predic et al. [2013], é apresentada a solução *Exposuresense*, criada para capturar

dados do ambiente utilizando os sensores existentes nos telefones celulares, a fim de analisar a qualidade do ar. Para tal, foi criado um adaptador para integrar a solução ao middleware GSN, responsável por capturar e armazenar os dados sobre a qualidade do ar para processamento futuro. Dessa forma, é possível observar que GSN é uma excelente solução a ser utilizada para gerenciamento de redes sensores em locais públicos.

Redes Sociais

Em redes sociais, usuários podem acessar, publicar ou compartilhar conteúdo gerado ou obtido através de sensores no dispositivo móvel para interação com os seus contatos em redes sociais, como *twitter*, *facebook* e *instagran*.

Qin et al. [2011] apresentam o *RestThing*, uma solução para IoT que, além de permitir a interoperabilidade entre dispositivos heterogêneos, permite que o ambiente possa ser integrado com redes sociais, como *twitter*, *facebook* e o *Sina micro-blog* (rede social similar ao *Twitter* muito utilizada na China), possibilitando que dispositivos, como sensores, televisões, entre outros, possam postar eventos detectados no ambiente nessas redes sociais.

2.3. Considerações Finais

A ideia básica por trás dos conceitos da IoT está na presença acentuada de uma grande variedade de dispositivos e objetos do mundo real, como sensores, atuadores, TVs, celulares e, até mesmo, objetos pessoais (joias, sapatos e roupas), unicamente identificados e acessíveis por meio de esquemas de endereçamento, protocolos e tecnologias de comunicação. Tal endereçamento permite que esses objetos possam compartilhar e consumir informações e serviços utilizando a Internet como principal canal de comunicação. Nesse contexto, a IoT torna-se um paradigma com grande potencial para contribuir positivamente em vários domínios do mundo real, como, por exemplo, transporte e logística, ambientes inteligentes, assistência à saúde e indústria, permitindo vislumbrar o desenvolvimento de uma vasta quantidade de aplicações para otimização das atividades executadas nesses ambientes.

No entanto, para que a IoT possa se tornar uma realidade cada vez mais presente no cotidiano das pessoas, questões desafiadoras ainda precisam ser melhor investigadas,

principalmente relacionadas ao desenvolvimento de soluções providas de interfaces de fácil utilização, para possibilitar que usuários não especialistas – usuários com pouco ou quase nenhum conhecimento em computação – possam manipular essas soluções adequando o ambiente às suas necessidades. Assim, para melhor investigar as soluções de *middleware*, previamente comentadas nesse capítulo, o Capitulo 3 apresenta tais soluções com maiores detalhes, avaliando-as com relação a desafios superados e tecnologias utilizadas.

Capítulo 3 – Plataformas para Internet das Coisas

Este capítulo apresenta uma visão geral sobre o estado da arte das soluções de *middleware* para IoT, bem como os desafios enfrentados e tecnologias utilizadas, estando organizado da seguinte forma: primeiramente, serão explicados em maiores detalhes as soluções previamente comentadas no capítulo anterior, isto é, *Aspire*, *Ebbits*, *Ubiware*, *EcoDIF*, *LinkSmart*, *Socrades*, *Sm4all*, *openHAB*, *Task Computing*, *GSN* e *RestThing*; em seguida, serão apresentados os principais desafios enfrentados pelos usuários quanto ao desenvolvimento desse tipo de solução, assim como as principais tecnologias utilizadas como protocolos, padrões e dispositivos; e, finalmente, as soluções de *middleware* apresentadas serão avaliadas de acordo com os seguintes critérios: *i)* desafios atendidos; e *ii)* tecnologias utilizadas.

3.1. Desafios de *Middleware* para IoT

O desenvolvimento de uma plataforma de *middleware* para IoT não é uma tarefa trivial. Essa é uma atividade que requer conhecimento e experiência no tocante ao uso de diferentes tecnologias – tanto de *hardware* quanto de *software* – e no desenvolvimento de meios que permitam superar a grande variedade de desafios existentes no desenvolvimento desse tipo de solução. Alguns desses desafios são descritos e discutidos a seguir:

1. **Interoperabilidade -** Atualmente, a maioria dos dispositivos existentes no mundo real é criada com base em protocolos e *hardwares* proprietários fechados, impedindo que esses possam ser inseridos na arquitetura da aplicação. Portanto, as soluções de *middleware* para IoT devem ser projetadas para permitir que diferentes dispositivos possam interoperar entre si, independente do protocolo ou *hardware* utilizados em sua concepção. Para superar esse desafio, algumas arquiteturas de *middleware* utilizam modelos semânticos para abstrair características comuns entre os objetos e tecnologias, possibilitando assim a

interação entre os objetos [Zhang e Hansen 2008] e [Aberer e Hauswirth 2006]. Além disso, serviços de interoperabilidade baseados nos protocolos de descobertas de serviços, como UPnP e JINI (*Java Intelligent Network Infrastructure*), podem ser utilizados para prover um canal de comunicação comum entre os dispositivos que utilizam diferentes tecnologias [Allard et al. 2003], [Sameh e El-Kharboutly 2004] e [Jo et al. 2008].

2. **Conectividade** - Objetos físicos podem fazer uso de diferentes meios de comunicação, dentre as interfaces de comunicação mais utilizadas estão *Wi-Fi*, *Bluetooth*, *ZigBee*, 3G/4G, RFID e NFC. Em alguns casos, a conexão desses objetos à Internet depende da integração da tecnologia utilizada com a pilha TCP/IP (*Transmission Control Protocol/Internet Protocol*). Por exemplo, as soluções de *middleware* que utilizam tecnologias como RFID ou NFC devem desenvolver componentes auxiliares como *gateways* e *proxies*, para permitir acesso às principais funcionalidades dos dispositivos e à troca de informações entre os demais dispositivos na IoT [Jara et al. 2013] [Zhu et al. 2010]. Outras abordagens baseadas no padrão IEEE 802.15.4 (e.g. ZigBee, ISA100.11a e Wireless HART) podem fazer uso do protocolo IPv6 para estabelecer conectividade de dispositivos com a Internet por meio do protocolo 6LoWPAN [Montavont, Roth e Noël 2014].
3. **Escalabilidade –** De acordo com Sundmaeker et al. [2010], espera-se que até 2020 o número de dispositivos conectados à Internet esteja na faixa dos bilhões. Portanto, qualquer solução de middleware para IoT deve ser capaz de lidar com o número crescente de dispositivos e requisições, inclusive funcionar corretamente, mesmo em situações de uso intenso. Devido à sua facilidade de provisão sob demanda, o paradigma da computação nas nuvens é uma das principais alternativas de prover escalabilidade nas soluções de middleware para IoT [Soldatos, Serrano e Hauswirth 2012].
4. **Esquemas de Nomeação -** Com o advento da Internet das Coisas, os esquemas de nomeação tornam-se parte integrante das soluções e, por consequência, um novo desafio a ser superado. No contexto de IoT, inúmeras tecnologias e protocolos poderão ser utilizados como esquema de nomes para os diferentes objetos. Atualmente, a mais importante abordagem utilizada é o uso do protocolo IP (*Internet Protocol*) na versão 6. O IETF (*Internet Engineering Task Force*), por

exemplo, definiu um protocolo de padrão aberto que permite usar IPv6 nas redes IEEE 802.15.4, denominado de 6LoWPAN, o qual é usado em muitas soluções para IoT [Jara, Zamora e Skarmeta 2012]. Outras alternativas usadas como esquema de nomeação são: *i)* identificação por rádio frequência, largamente utilizadas na indústria e comercio para identificação única de objetos, conforme pode ser observado em [Welbourne et al. 2009] e [Angeles 2005]; *ii)* UUID (*Universally Unique IDentifier*) que é um identificador extraído do objeto/dispositivo ou de partes dele, como, por exemplo, um identificador *Bluetooth* ou o endereço MAC (*Media Access Control*) de uma interface *Wi-Fi*; e *iii)* identificadores gráficos como código de barras e *QRCode* [Adelmann, Langheinrich e Flörkemeier 2014] e URIs (*Uniform Resource Identifier*) comumente utilizadas em soluções como em [Pires et al. 2015] e [Qin et al. 2011].

5. **Flexibilidade -** Por estar em seu estágio inicial, a IoT passará por constantes mudanças decorrentes do surgimento de novas tecnologias, protocolos, padrões e hardware. Assim, as arquiteturas de middleware para IoT devem ser flexíveis o suficiente para se adaptar ao surgimento de novas tecnologias. O uso de modelos semânticos tem contribuído fortemente para construção de soluções mais flexíveis, facilitando a inclusão de novas tecnologias e novos objetos físicos [Aberer e Hauswirth 2006], [Katasonov 2008] e [Song, Cárdenas e Masuoka 2010].
6. **Reusabilidade -** Um importante desafio para os projetistas de *middleware* para IoT é possibilitar que a solução concebida possa permitir a reutilização de diferentes partes de sua arquitetura na composição de novas soluções. Isso é importante, pois reduz as dificuldades enfrentadas por outros desenvolvedores quanto à criação de novas soluções, cabendo a esses profissionais a preocupação apenas com as funcionalidades direcionadas ao domínio específico do problema a ser resolvido. Algumas soluções de *middleware* existentes já permitem a reutilização de componentes arquiteturais por meio da disponibilização de APIs (*Application Programming Interface*), como, por exemplo, os middleware *LinkSmart*[3] e *Ebbits*[4].

[3] https://www.linksmart.eu/redmine
[4] http://www.ebbits-project.eu/news.php

7. **Portabilidade -** Um dos principais desafios da IoT está em lidar com a diversidade dos diferentes tipos de objetos que podem participar da rede. Assim, é importante que uma solução de middleware para IoT possa dar suporte a dispositivos mantidos sob diferentes plataformas de *hardware* ou sistema operacional. Uma possível solução para lidar com esse desafio é o desenvolvimento de soluções de *middleware* que utilizem linguagens de programação portáveis, como a linguagem Java. Por exemplo, os *middlewares Task Computing*, ASPIRE e SOCRADES são soluções portáveis, desenvolvidas na linguagem de programação Java, que podem ser implantadas em diferentes sistemas operacionais pelo uso da JVM (*Java Virtual Machine*). Além disso, as soluções Web também têm sido utilizadas na literatura para prover soluções portáveis, como o *middleware* EcoDiF, uma solução Web criada para integração de objetos heterogêneos por meio da Internet.
8. **Usabilidade -** A ampliação da Internet das Coisas requer o desenvolvimento de soluções de *software* que tornem mais simples e fácil o gerenciamento dos objetos. Nas atuais soluções de *middleware* para IoT, percebe-se que essas, em sua grande maioria, são criadas para serem manipuladas por usuários com grau elevado de conhecimento técnico, ou seja, se, por um lado, tais soluções já são difíceis de serem manipuladas por usuário especialistas, por outro lado, apresentam-se praticamente inviáveis a usuários não especialistas. Até mesmo as soluções que oferecem *interfaces* de gerenciamento, como os *middleware LinkSmart*, *Ebbits*, *Socrades* e *EcoDiF*, são criadas para atender às necessidades de usuários especialistas em IoT com relação às configurações de parâmetros relacionados ao ambiente ou objeto a ser gerenciado. Assim, o desenvolvimento de *middleware* que forneçam interfaces de gerenciamento simples e fáceis ainda é um problema em aberto. O uso de padrões e modelos pode ser uma solução para que usuários não especialistas possam criar e modelar os diferentes tipos de comportamentos que o ambiente poderá assumir, possibilitando a sua inclusão no processo criativo e gerencial de ambientes inteligentes para IoT.
9. **Segurança e Privacidade –** Na maioria das vezes, o papel dos dispositivos integrados à Internet das Coisas é o de coletar dados privados que podem ser transportados por meio de redes sem a segurança adequada. Por esse motivo, qualquer solução de *middleware* para IoT deve garantir que os dados trafegados

na rede estejam seguros e que os serviços e dispositivos da rede só poderão ser acessados por pessoas e/ou objetos com permissão de acesso. Segurança e privacidade devem ser consideradas os pilares centrais de toda solução para IoT, pois, sem a implementação dessas características, é possível que qualquer dispositivo venha se conectar a um sistema IoT particular e fazer seus serviços, comprometendo a segurança do ambiente e das pessoas.

10. **Gerenciamento de grande volume de dados –** Devido à grande quantidade de dispositivos na IoT, o gerenciamento de um maior volume de dados é uma importante funcionalidade que uma plataforma de *middleware* para IoT deve prover, permitindo melhor acompanhamento da demanda de coleta e análise de dados e, consequentemente, prover respostas em tempo hábil [Pires et al. 2015]. Para tratar esse problema, novos algoritmos e técnicas de busca e indexação para grandes volumes de dados deverão ser utilizados conforme a demanda de dados for aumentando. Na literatura é possível encontrar propostas para o uso de modelos como *big data* [Ciobanu et al. 2014] e *data warehouse* [Bin, Yuan e Xiaoyi, 2010] para armazenamento do grande volume de dados da IoT.
11. **Gerência de contexto –** Outro importante desafio para as soluções de *middleware* para IoT é a ciência de contexto, que consiste em obter dados de contexto de um determinado domínio e, com base nesses mesmos dados, definir o estado atual do ambiente, objeto ou pessoa, a fim de definir a melhor ação a ser tomada no ambiente [Gu, Pung e Zhang 2005]. O atendimento dessa funcionalidade, em geral, requer a utilização de gerenciadores de contexto, os quais são responsáveis por detectar um evento ocorrido no contexto e definir a melhor ação a ser tomada.
12. **Descoberta de Dispositivos –** Apensar de não ser um requisito obrigatório, mecanismos de descoberta de dispositivos são de grande importância em soluções para IoT, permitindo que dispositivos e serviços possam ser descobertos e configurados automaticamente, sem que, para isso, seja necessária a criação de mecanismos exclusivos para lidar com as diferentes tecnologias e protocolos utilizados pelos dispositivos. Atualmente, o protocolo de descoberta de dispositivo mais utilizado para esse propósito é o protocolo UPnP.

Em vista dos conceitos ilustrados, de forma concisa e judiciosa, são vários os desafios a serem superados para o desenvolvimento de novas solução de *middleware* para IoT. Na literatura existem soluções promissoras que podem ser utilizadas como base para

o desenvolvimento de soluções mais robustas e utilizáveis na prática. Contudo, uma grande parte das atuais soluções são fechadas, não permitindo serem alteradas e/ou são de difícil extensão e reutilização, até mesmo por um usuário especialista.

3.2. Plataformas de *Middlewares* para IoT

Plataformas de *middleware* têm sido adotadas como principal solução para lidar com os diversos desafios existentes na IoT. No entanto, devido à sua complexidade, essas soluções são de difícil manuseio. Dessa forma, as seções a seguir destinam-se a apresentar as soluções de *middleware* previamente discutidas no capítulo 2.

3.2.1. *ASPIRE*

ASPIRE (*Service-Oriented Cross-Layer Infrastructure for Distributed smart Embedded devices*) é uma solução de *middleware* criada para implementar o paradigma da IoT em ambientes industriais baseados em tecnologias RFID. ASPIRE permite criar novas soluções utilizando uma arquitetura centralizada e compatível com as normas da EPC (*Eletronic Product Code*) [Anggorjati et al. 2010]. Tais normas são utilizadas para facilitar o controle e interoperação entre objetos existentes no mundo real, como leitores, etiquetas, sensores e atuadores que utilizem tecnologia RFID [Kefalakis et al. 2009].

ASPIRE possui um ambiente de desenvolvimento integrado para que usuários especialistas possam criar soluções de alto nível utilizando apenas os metadados dos objetos, os quais descrevem as informações relevantes que, por sua vez, identificam unicamente um dispositivo [Sundmaeker et al. 2010]. A Figura 3.1 apresenta a arquitetura do *middleware* ASPIRE e seus principais componentes.

O ambiente de desenvolvimento integrado (IDE) do ASPIRE permite que um usuário especialista possa acessar, por meio de ferramentas, os demais componentes e funcionalidades da arquitetura, a fim de se criarem soluções para um determinado domínio ou de se definir um contexto para atender às necessidades da aplicação. A comunicação entre os componentes da arquitetura é realizada por meio de serviços Web, utilizando para tal finalidade os protocolos HTTP (*Hypertext Transfer Protocol*) como principal canal para transporte de dados e o SOAP (*Simple Object Access Protocol*), para encapsulamento e envio dos dados.

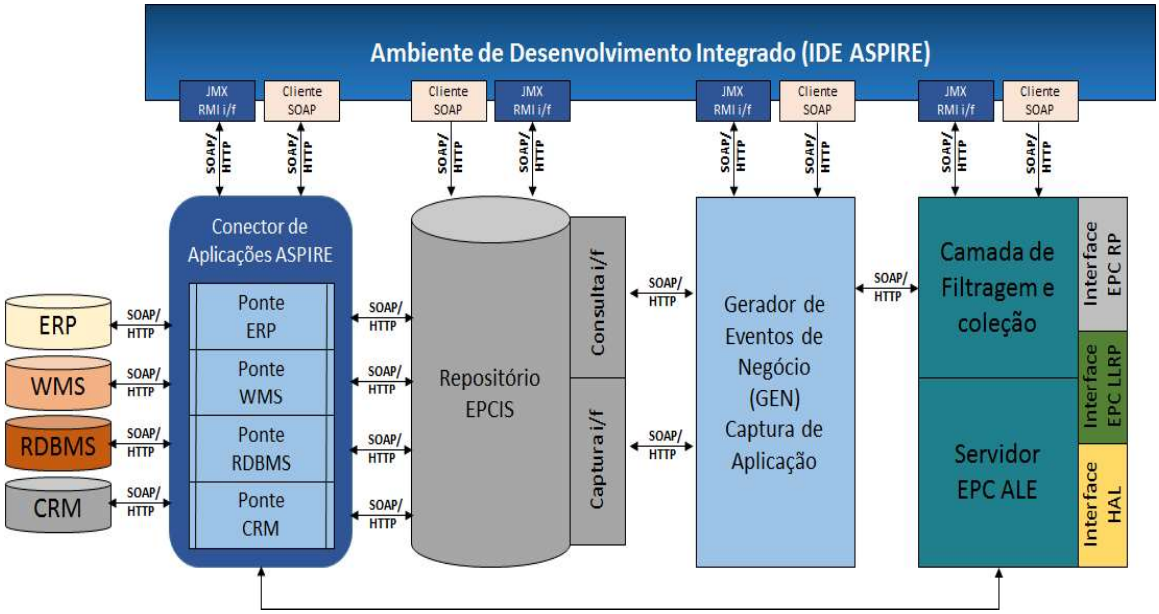


Figura 3.1. Arquitetura do *middleware* aspire
Fonte: Adaptado de Anggorjati et al. [2010]

Os componentes EPC-RP (EPC *Reader Protocol*) e EPC-LLRP (EPC *Low-Level Reader Protocol*) permitem a comunicação entre leitores de diferentes fornecedores. A arquitetura define uma camada de abstração de *hardware* (*HAL - Hardware Abstraction Layer*) para possibilitar a virtualização de etiquetas e a padronização de mensagens. Os componentes *filtragem e coleção* (F&C) e ALE (*Application Level Event*) são responsáveis pela coleta, junção e filtragem dos dados produzidos pelos sensores, enquanto o *gerador de eventos de negócio* (BEG - *Business Event Generator*) é responsável por mapear e configurar automaticamente as informações providas pelos componentes F&C no repositório EPC-IS (*Information Sharing*). Por fim, as informações mapeadas pelo BEG são armazenadas no repositório EPCIS, onde serão utilizadas por aplicações corporativas que podem se conectar ao ASPIRE por meio de diferentes conectores.

A principal vantagem do ASPIRE é possibilitar que leitores RFID heterogêneos que não implementam o protocolo EPC possam ser integrados à arquitetura por meio de uma conversão semântica, realizada pela camada HAL, permitindo que uma quantidade maior de dispositivos possam interoperar. Além disso, ASPIRE possui uma ferramenta denominada BPWME (*Business Process Workflow Management Editor*), que permite a modelagem gráfica de arquivos APDL (*Aspire Process Description Language*), a qual é

uma linguagem utilizada para definição dos processos de negócio gerenciados pelo ASPIRE, conforme apresentado pela Figura 3.2.

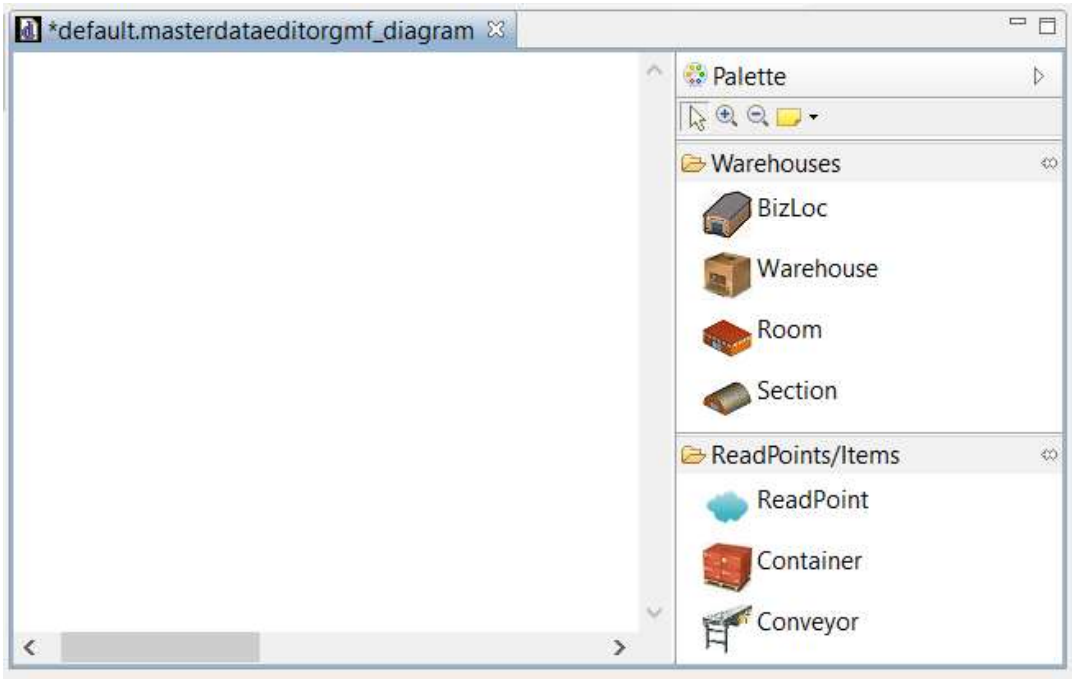


Figura 3.2. Aspire RFID editor
Fonte: Adaptado de Anggorjati et al. [2010]

No entanto, BPWME é uma ferramenta criada para que usuários especialistas possam definir as regras de negócio de diferentes ambientes de acordo com os eventos das etiquetas RFID. É importante destacar que o editor do Aspire não se restringe apenas a usuários especialistas, porquanto usuários não especialistas podem também manusear o editor com o propósito de moldar o ambiente de acordo com as suas necessidades, desde que, para isso, obtenham prévio conhecimento por meio de documentação sobre o modelo a ser manipulado.

3.2.2. EBBITS

O projeto EBBITS (*Enabling Bussiness-Based Internet of Things and Services*) é uma solução de *middleware* para IoT para ambientes industriais e agrícolas [Ebbits 2011]. EBBITS é uma plataforma de *middleware* extensível, reutilizável, escalável e orientada a serviços para automatização e monitoramento de processos. O EBBITS possibilita que uma grande variedade de dispositivos industriais, como, por exemplo, sensores, robôs, etiquetas RFID e PLC (*Programmable Logic Controller*), possam ser descobertos, configurados e unicamente identificados de forma automática, permitindo que esses

possam interoperar, provendo ambientes mais dinâmicos e interativos [Kostelnik, Sarnovsk e Furdik 2011].

Em um ambientes industriais, a comunicação entre dispositivos em uma rede sem fio, pode ser significativamente degradada devido aos ruidos gerados pelos equipamentos industriais ou, até mesmo, pelos sinais de rádio emitidos por outros aparelhos industriais, que podem afetar diretamente a comunicação em uma rede sem fio [Vajda et al. 2011]. Em tais ambientes, redes sem fio podem ter rendimento reduzido e uma maior perda de pacotes, podendo, até mesmo, parar de funcionar. Nesse contexto, o EBBITS utiliza como base o paradigma da comunicação oportunista, que permite melhorar a comunicação em termo de maior disponibilidade e confiabilidade. A Figura 3.3, abaixo ilustrada, apresenta o módulo Gerenciador de Rede do EBBITS e os componentes responsáveis por implementar as caracterísitcas oportunistas na arquitetura:

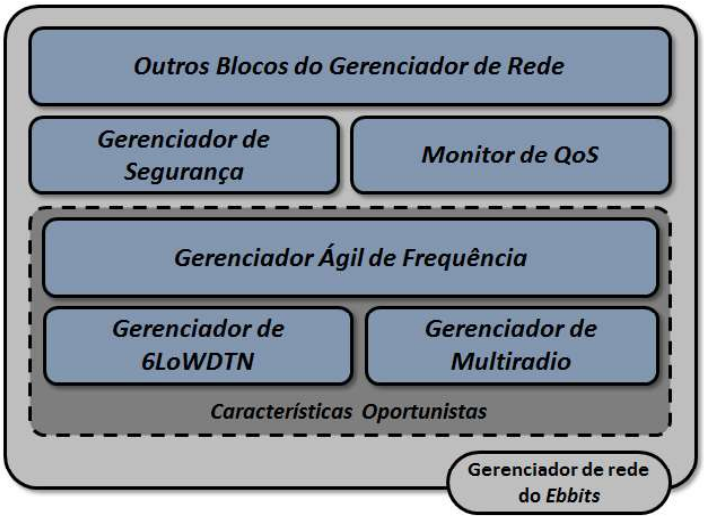


Figura 3.3. Componentes do módulo network manager.
Fonte: Adaptado de EBBITS [2011]

A implementação dos conceitos de comunicação oportunista na arquitetura do EBBITS é realizado por meio da introdução de três componentes principais ao módulo Gerenciador de Rede. O primeiro componente, Gerenciador Ágil de Frequência, analisa frequentemente o espectro e os dados da rede com objetivo de definir o melhor canal para a comunicação e envio de dados entre os dispositivos da rede. Já o segundo componente, Gerenciador de 6LoWDTN, permite ao EBBITS fornecer melhor confiabilidade na transferência das informações, garantida por meio da análise da taxa de transmissão bem sucedida e dos atrasos na entrega das informações. E, por fim, o componente Gerenciador de Multirádio analisa as interfaces de comunicação existente nos dispositivos, a fim de

definir qual a melhor interface a ser utilizada para realizar a comunicação ou enviar dados entre os dispositivos. Esse tipo de gerenciador é empregado para garantir maior conectividade e mobilidade de um dispositivo EBBITS [Ebbits 2011].

A principal vantagem da abordagem EBBITS está nos módulos que garantem a conectividade dos dispositivos e a entrega de pacotes em ambientes sem fio ruidosos. Conforme já foi mencionado, isso é realizado por meio da troca dinâmica de canais, gerenciamento multiradio e implementação do conceito de redes tolerantes a atraso. Contudo, por ser uma ferramenta voltada especificamente para a indústria, somente usuários especialistas estão habilitados a manusear o EBBITS. Cabe ressaltar que seria interessante a inclusão de funcionalidades que permitissem aos usuários modelar novas regras de negócio ou definir diferentes comportamentos ao ambiente, por exemplo, para atender a uma necessidade específica da empresa em função de mudanças impostas pelo mercado.

3.2.3. UBIWARE

UBIWARE é uma plataforma de *middleware* baseada em uma arquitetura orientada a serviço, que permite ser estendida e reutilizada para o desenvolvimento de novas soluções para IoT. A flexibilidade da arquitetura do UBIWARE, para permitir que essa possa atender a diferentes cenários, é alcançada pelo uso de um conjunto de modelos semânticos, que pode ser estendido e adaptado para atender as diferentes necessidades dos usuários e ambientes. Também, UBIWARE faz uso de um sistema multi-agentes, responsável por gerenciar e executar as principais funcionalidades da arquitetura, como a descoberta de dispositivos e serviços, além do gerenciamento dos dados [Katasonov 2008].

Cada dispositivo no UBIWARE é conectado a um agente que permite que os usuários possam ter o pleno conhecimento sobre estado de cada dispositivo. Como resultado, o usuário pode gerenciar e monitorar o ambiente por meio das informações fornecidas pelos agentes. Dessa forma, tal conhecimento pode ser trocado, com a finalidade de melhorar a execução dos serviços e objetos vinculados ao UBIWARE [Pires et al. 2015]. A Figura 3.4, abaixo, apresenta a arquitetura do UBIWARE e seus componentes principais:

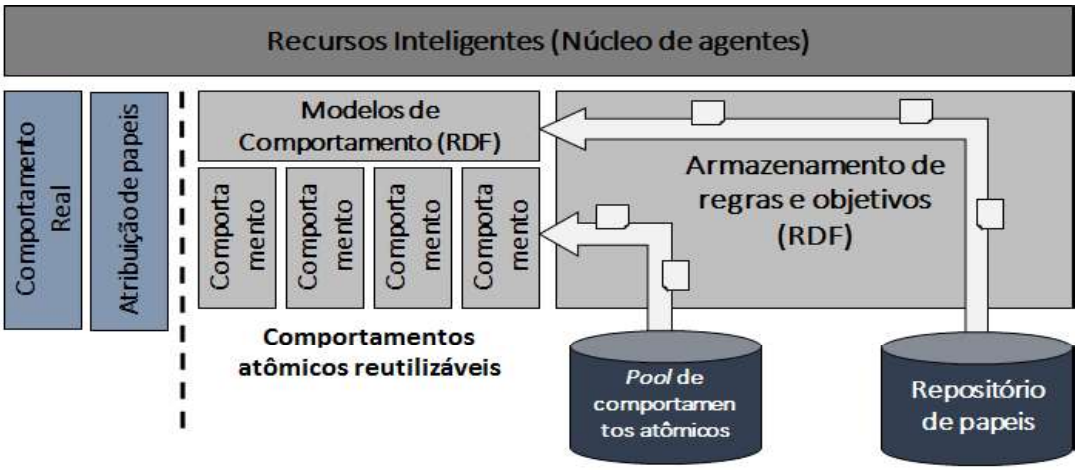


Figura 3.4. Plataforma Core do Ubiware.
Fonte: adaptado de Katasonov [2008]

A arquitetura do UBIWARE é composta por duas camadas principais: recursos inteligentes e modelos de comportamentos. A camada de recursos inteligentes, implementada como base o *framework* JADE[5] (*Java Agent Development Framework*), é responsável por criar, alterar, manter e gerenciar o ciclo de vida dos agentes no UBIWARE.

A camada de modelos de comportamentos gerencia os comportamentos e papéis utilizados pelos agentes. Essa camada é subdividida em duas partes: comportamentos reais, que são comportamentos pré-definidos pela solução que não podem ser modificados; e os reutilizáveis, que são os que podem ser estendidos e modificados pelo usuário, a fim de atender a uma necessidade específica [Terziyan, Kaykova e Zhovtobryukh 2011].

UBIWARE é uma atraente solução de criação de ambientes inteligentes para IoT, sua principal vantagem está no uso de modelos reutilizáveis, responsáveis por definir os diferentes papéis e comportamentos dos agentes, permitindo assim que a solução possa ser modificada para atender a diferentes domínios. No entanto, UBIWARE é uma solução de difícil manuseio, pois somente usuários com grau elevado de conhecimento técnico em linguagem de programação e no padrão semântico RDF podem adaptar ou estender os modelos propostos no UBIWARE.

[5] http://jade.tilab.com/

3.2.4. EcoDiF

Diferente das soluções anteriores, EcoDiF (Ecossistema de Dispositivos Físicos) não é uma plataforma de *middleware*, mas, sim, uma plataforma Web capaz de conectar uma variedade de dispositivos heterogêneos, formando um ecossistema de coisas [Pires et al. 2014]. O principal objetivo do EcoDiF é permitir que qualquer coisa possa ser configurada e conectada à Internet, permitindo que essa possa interoperar de forma transparente em uma rede heterogênea. EcoDiF, assim como as demais soluções, não se limita ao gerenciamento de um domínio específico, podendo ser configurada e estendida para atender a cenários e domínios diversos. Pires et al. [2015] apresentam alguns estudos de caso utilizando o EcoDiF como base para implementação de diferentes cenários, como no monitoramento de *datacenters*, monitoramento remoto de pacientes, sensoriamento participativo e monitoramento de dutos e redes de água e gás.

EcoDiF utiliza os princípios de REST (*Representational State Transfer*) para conectar e gerenciar dispositivos físicos por meio da Internet, possibilitando que seus dados e serviços possam ser utilizados e reutilizados como serviços da Web [Delicato, Pires e Batista et al. 2013]. A Figura 3.5, a seguir, apresenta a arquitetura do EcoDiF e seus módulos principais:

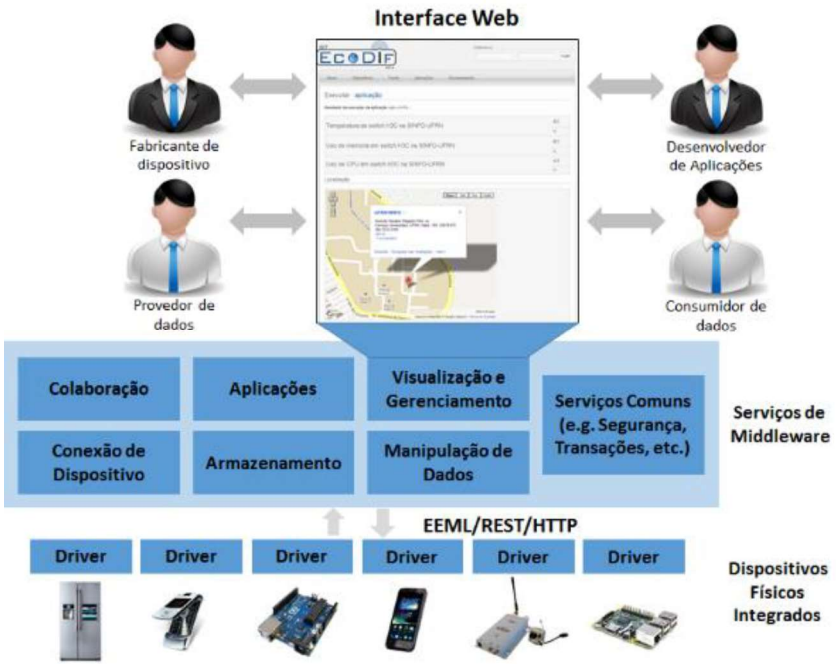


Figura 3.5. Arquitetura da EcoDiF
Fonte: adaptado de Pires et al. [2015]

A arquitetura do EcoDiF é dividida em três camadas principais: camada física, camada de *middleware* e a camada de interface. A camada física fica localizada na parte inferior da arquitetura, responsável por prover meios para integração dos dispositivos físicos presentes no ambiente ao EcoDiF. Essa integração é realizada por meio do desenvolvimento de *drivers*, que são mecanismos criados por usuários técnicos utilizando partes do EcoDiF, para permitir que dispositivos físicos ou, até mesmo, *softwares* legados possam ser integrados e gerenciados pelo EcoDiF, o qual, particularmente, possuem dois tipos de *drivers*: ativos e passivos. Enquanto os *Drivers* ativos obtém periodicamente as informações dos dispositivos por meio de um processo de requisição contínua, independentemente se houve ou não mudança nos dados; os *drivers* passivos, por sua vez, esperam por notificações advindas dos dispositivos quando há mudança nas informações monitoradas.

A camada de *middleware* do EcoDiF provê e gerencia os principais serviços oferecidos pela solução, como a manipulação de dados, segurança, privacidade e controle de transações, armazenamento de dados, gerenciamento de aplicações, conexão dos dispositivos e o gerenciamento das interfaces para visualização dos dados e serviços, quais sejam:

- O módulo de conexão de dispositivos visa facilitar a conexão de dispositivos físicos a IoT, fazendo uso da API do EcoDiF e dos *drivers* customizados. O módulo de visualização e gerenciamento provê uma interface Web para permitir o acesso dos usuários aos dispositivos e serviços gerenciados pelo EcoDiF, podendo monitorar seu estado e localização;
- O módulo de armazenamento consiste em gerenciar e armazenar dados em uma base relacional e *scripts* de aplicação em um sistema de arquivos, podendo também ser configurado para operar em nuvens;
- O módulo de colaboração permite que os usuários possam realizar buscas por dispositivos e aplicações a partir dos seus respectivos metadados, utilizando para isso uma interface Web. O módulo manipulação de dados é responsável por gerenciar e registrar os dados advindos das requisições HTTP;
- O módulo aplicações provê um modelo e um ambiente para programação e execução de aplicações que fazem uso dos dispositivos do EcoDiF;

- O módulo de serviços comuns fornece meios para garantir a segurança e privacidade dos dados, gerenciamento do ciclo de vida das aplicações, transações; entre outros.

Por fim, a camada de interface Web provê os principais meios de acesso dos usuários aos serviços, dispositivos e aplicações disponíveis pelo EcoDiF.

Quanto ao uso do EcoDiF, a principal vantagem para elaboração de ambientes inteligentes para IoT está na sua simplicidade, o que é alcançada com o uso moderado de protocolos e tecnologias para gerenciamento de dispositivos por meio da Internet e padronização das informações trocadas entre os dispositivos. Outras vantagens dessa solução são a flexibilidade e conectividade, as quais se alcançam pelo uso de *drivers*, que, por sua vez, são soluções criadas por usuário com conhecimento técnico, utilizando como base a API do EcoDiF, permitindo que se possa atender a diferentes domínios e dispositivos. No entanto, EcoDIF não faz uso de interfaces gráficas que permitam a usuários não especialistas a possibilidade de adaptar o ambiente de acordo com as suas necessidades, mas somente interfaces para permitir o acesso aos dispositivos e serviços.

3.2.5. *LinkSmart*

O projeto HYDRA[6] criou o middleware *LinkSmart*[7] para sistemas embarcados em rede que permite automatizar ambientes por meio da interoperabilidade entre objetos heterogêneos oriundos do próprio ambiente, como, por exemplo, sensores, atuadores e eletroeletrônicos [Zhang e Hansen 2008], que pode ser alcançada pelo uso de uma arquitetura orientada a um tipo de serviço que permite a troca de informações entre os objetos via Internet.

Além disso, para lidar com as diferentes limitações existentes nos objetos do mundo real, como, por exemplo, processamento, armazenamento e memória, a arquitetura do *LinkSmart* foi dividia em dois núcleos. Esses núcleos são denominados como elementos de aplicação, isto é, que foram criados para dar suporte a dispositivos com poder elevado de processamento (por exemplo, *notebooks*); e elementos de dispositivo, os quais dão suporte a objetos com baixo poder de processamento (por exemplo, sensores e atuadores). Figura 3.6 apresenta a arquitetura do *LinkSmart*:

[6] http://www.hydramiddleware.eu/news.php
[7] https://www.linksmart.eu/redmine

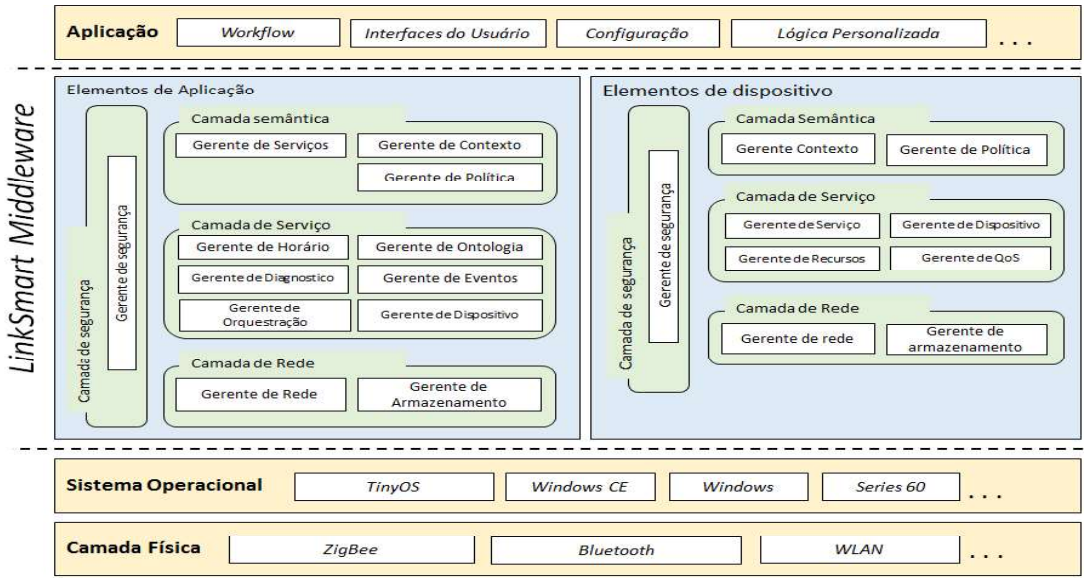


Figura 3.6. Arquitetura em camadas do *LinkSmart*
Fonte: adaptado de Zhang e Hansen [2008].

Os núcleos (elementos de aplicação e de dispositivo) são compostos pelas camadas de: i) segurança: responsável por prover meios para garantir a segurança dos dados e acesso controlado aos serviços; ii) semântica: responsável pelo gerenciamento das principais funcionalidades semânticas providas pelo LinkSmart, como, por exemplo, o gerenciamento de contexto e políticas; iii) serviços: responsável por fornecer um conjunto de serviços aos núcleos, como, por exemplo, orquestração de dados, controle de eventos, controle de qualidade, controle de dispositivos e recursos; e iv) rede: camada que provê a comunicação entre os diversos dispositivos e objetos do mundo real, fornecendo algoritmos otimizados para roteamento e armazenamento de dados a nível de dispositivo.

LinkSmart é uma solução completa para a criação de ambientes inteligentes, provendo funcionalidades e características importantes para o contexto da IoT, como a interoperabiliade entre dispositivos heterogêneos, um esquema eficiente para unicidade de dispositivos denominado HID (*Hydra Identification*), gerenciamento de contexto, descoberta e classificação automática de dispositivos, segurança e um conjunto de ferramentas técnicas para que usuários especialistas possam manipular e adaptar a solução para as necessidades dos usuários e do ambiente. Nesse contexto, o uso dos modelos semânticos do *LinkSmart* fornece maior flexibilidade para que a arquitetura possa ser extendida e adaptada. Apesar das vantagens, *LinkSmart* é uma solução focada em atender a usuários especialistas, não provendo nenhum meio para possibilitar que

usuários não especialistas possam assumir diferentes papéis na arquitetura, como definir os comportamento do ambiente com base em modelos.

3.2.6. *SOCRADES*

Semelhante ao *Ebbits,* SOCRADES (*Service-Oriented Cross-Layer Infrastructure for Distributed smart Embedded devices*) é um *middleware* orientado a serviços que explora a integração entre dispositivos no processo de chão de fábrica, com a finalidade precípua de prover canais de comunicação confiáveis e eficientes [Cannata, Gerosa e Taisch 2008]. SOCRADES utiliza como base para a integração e descoberta de dispositivos a especificação de serviços web para dispositivos DPWS (*Devices Profile for Web Services*) [Jammes et al. 2007]. A Figura 3.7 apresenta a arquitetura do *middleware* SOCRADES e seus componentes principais:

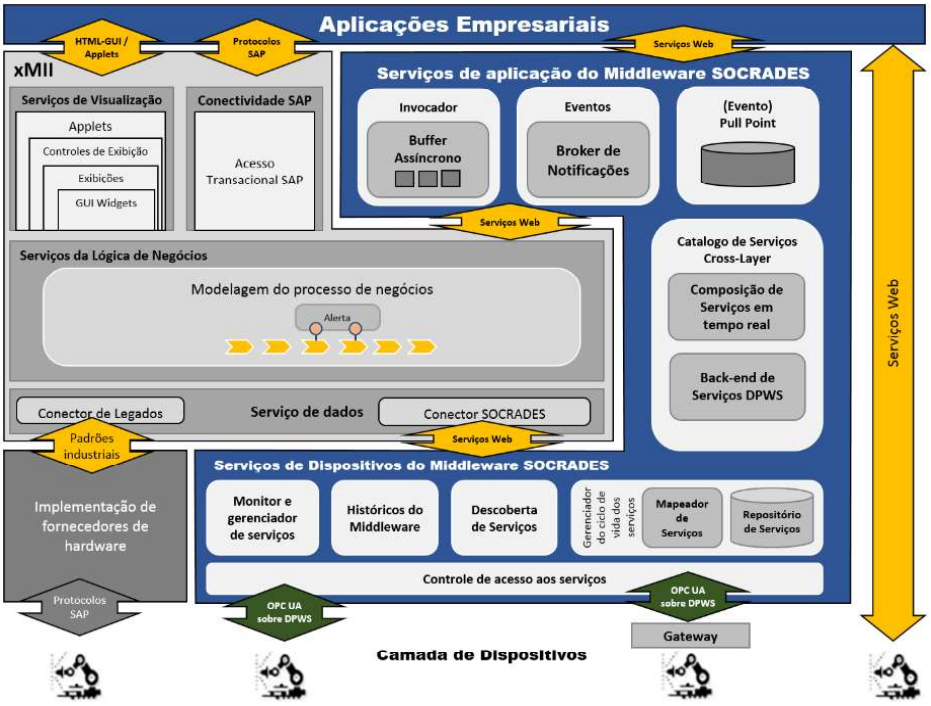


Figura 3.7. Arquitetura em camadas do *middleware* Socrades
Fonte: Adaptado de De Souza [2008].

A arquitetura do *middleware* SOCRADES é dividida em 4 camadas. **Camada de aplicações empresarias** é o *front-end* utilizado pelos usuários finais e aplicações empresarias externas que necessitem enviar e receber dados advindos do SOCRADES diretamente relacionados ao processo de negócio executado no chão de fábrica. A comunicação das aplicações dessa camada com as camadas inferiores, ocorre por meio de

Web services, interfaces HTML e ou por protocolos específicos do sistema SAP (Sistemas, Aplicações e Programas) [De Souza 2008].

As interfaces HTML são utilizadas na arquitetura para acesso à **camada xMII** (*xManufacturing Integration and Intelligence*), que é uma conexão direta entre o SOCRADES, o chão de fábrica e as operações de negócios controladas pelo sistema SAP [Kirkham 2008]. Essa camada garante que todos os dados que afetam diretamente a fabricação sejam visíveis em tempo real, incluindo informações a respeito de materiais, estado de equipamentos, custo e qualidade dos produtos [Gorbach 2006].

A camada de *middleware* fornece serviços específicos para os dispositivos como descoberta de dispositivos, mapeamento e monitoração de serviços providos pelos dispositivos, gerenciamento de dados históricos e um controlador de eventos.

Por fim, a **camada de dispositivos** representa os dispositivos existentes no chão de fábrica e os diferentes meios utilizados por esses para conexão com o SOCRADES.

SOCRADES é uma excelente alternativa para implementação de IoT em ambientes industriais, principalmente com relação à interoperabilidade de dispositivos heterogêneos e a integração desses ao processo de negócio da empresa, uma vez possibilitada por meio da camada xMII. A identificação de dispositivos e serviços no SOCRADES é realizada por intermédio de URI, a qual permite identificar um dispositivo ou serviço como único na arquitetura. SOCRADES possibilita, ainda, que dispositivos possam ser descobertos automaticamente por meio do protocolo *WS-Discovery*, implementado no padrão DPWS. No entanto, questões relacionadas à escalabilidade, gerenciamento de grandes volumes de dados e contexto não são cobertas no SOCRADES.

Assim como o ASPIRE, SOCRADES possui uma interface gráfica para permitir que usuários especialistas possam definir todo o processo de negócio da linha de produção. Contudo, esse processo se restringe apenas a usuários especialistas, além de também estar restrito a dispositivos específicos como RFID e sensores.

3.2.7. *SM4ALL*

SM4ALL (*Smart Home for All*) é uma plataforma inovadora que visa estudar e desenvolver interoperabilidade entre serviços inteligentes de *softwares* embarcados em ambientes imersivos, utilizando como base uma abordagem orientada a serviços e

técnicas de composição. SM4ALL permite criar cenários desafiadores para atender as prioridades de pessoas com diferentes capacidades, incluindo idosos e deficientes [Catarci et al. 2011]. A Figura 3.8, adiante, busca apresentar uma visão geral da arquitetura do SM4ALL;

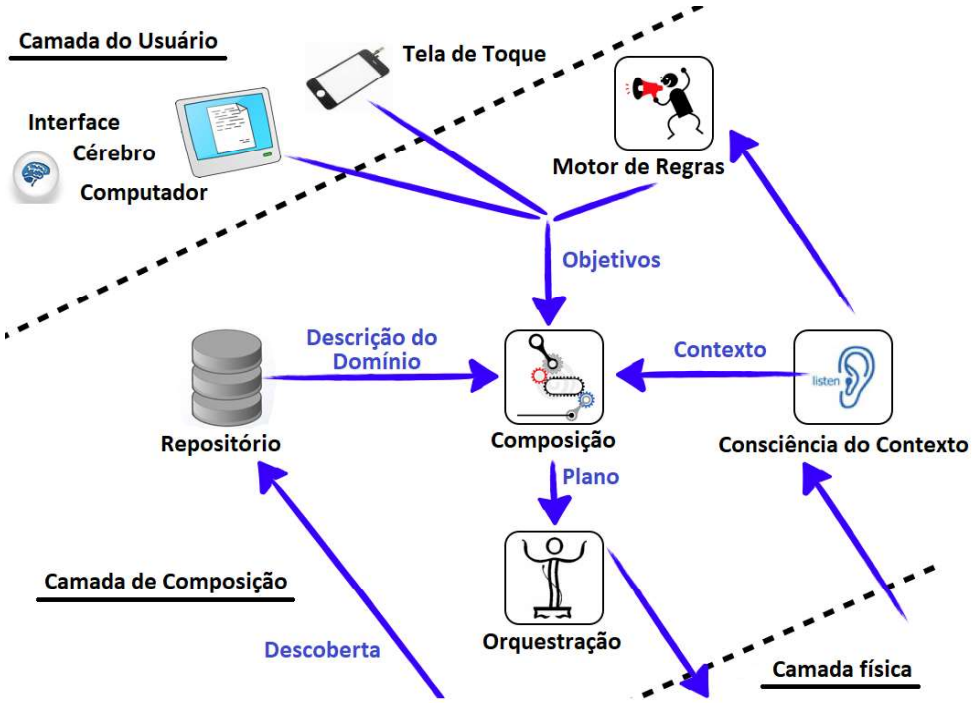


Figura 3.8. Visão Geral da Arquitetura do SM4ALL
Fonte: Adaptada de Warriach et al. [2011].

A arquitetura do SM4ALL é composta por três camadas principais, i) a camada do usuário, ii) a camada de composição e iii) a camada física.

A camada de usuário abrange as principais interfaces de entrada e saída utilizadas pelos diferentes tipos de usuário para acesso aos serviços, dados e dispositivos gerenciados pelo SM4ALL [Warriach et al. 2010]. Como mecanismo de entrada, o SM4ALL dispõe do uso de interfaces para reconhecimento de comando de voz, interfaces cérebro-computador, interfaces para toque de tela e uma interface em 3D do ambiente monitorado, especialmente para permitir que o usuário possa navegar no ambiente e controlar os objetos por meio dessa interface.

A camada central da arquitetura do middleware é a camada de composição, a qual fornece a abstração para cinco componentes.

O componente repositório é responsável por manter a descrição das listas de tipos de serviços suportados e providos pelo middleware, incluindo uma marcação semântica apropriada a respeito das operações propostas, bem como dos atuais registros sobre os dispositivos instanciados, que podem ser ativados a qualquer momento.

O componente, consciência de contexto, monitora constantemente o status dos dispositivos e a localização dos usuários, coletando e agregando as informações, inclusive, por meio de um mecanismo de publish-subscribe, notifica as partes interessadas.

O componente motor de regras utiliza informações sobre as mudanças do contexto e, se certas condições forem atendidas, realiza as ações esperadas. Por exemplo, quando um incêndio é detectado, um plano de emergência deverá ser posto em prática, o qual é diretamente inovado pelo módulo de composição.

O módulo de composição recebe objetivos complexos e de alto nível, emitidos pela camada de usuário ou pelo motor de regras, tentando cumprir os objetivos pela geração de composições apropriadas com os serviços disponíveis.

O motor de orquestração é o componente responsável pelas orquestrações sintetizadas através da composição. Esse componente controla os serviços por meio de um ou mais “proxies” de dispositivos e coordena esses serviços para executar um processo de acordo com o pedido recebido da orquestração.

Ademais, a camada física é composta pela infraestrutura de hardware dos objetos presentes no mundo físico, como sensores, eletroeletrônicos e dispositivos em geral. O principal objetivo da camada física é, continuamente, integrar redes e dispositivos heterogêneos, para que o middleware SM4ALL possa prover serviços de dispositivos e informações as camadas superiores, utilizando, para isso, uma interface padrão de abstração, não importando a tecnologia utilizada pelo dispositivo (Bluetooth, ZigBee e UPnP).

Dentre as soluções apresentadas até o momento, as inúmeras interfaces de comunicação constituem a principal vantagem do SM4ALL, sendo implementadas pelo SM4ALL para inclusão de usuários diversificados, como: idosos, portadores de deficiência física, entre outros. Por ser uma solução em fase inicial, há ainda algumas melhorias que precisam ser realizadas, como, por exemplo, a criação de um módulo

gráfico, a fim de possibilitar que as composições executadas atualmente pelo middleware possam ser feitas pelos usuários de acordo com as suas necessidades, permitindo, também, que, quando necessário, possam ser modificadas ou excluídas, podendo-se utilizar, para isso, um modelo conceitual bem definido, como, por exemplo, a BPMN.

3.2.8. *OpenHAB*

OpenHAB (Open Home Automation Bus) é uma solução de *middleware* de código aberto desenvolvida utilizando a linguagem de programação Java. Possui uma arquitetura orientada a serviço baseada em REST, implementada utilizando o *framework* OSGi (*Open Service Gateway Initiative*) *Eclipse Equinox*, que fornece um conjunto de pacotes que implementam vários serviços da especificação OSGi [OSGi Alliance 2017], provendo ao *openHAB* uma arquitetura altamente modular e permitindo adicionar e/ou remover funcionalidades em tempo de execução, sem que, para isso, seja necessário a interrupção dos demais serviços providos pela arquitetura.

OpenHAB faz uso de uma grande coleção de bibliotecas que o permite gerenciar e integrar diferentes tecnologias (*bluetooth* e *wifi*), protocolos (MQTT e TCP/UDP) e dispositivos (Sensores e eletroeletrônicos) de diferentes fornecedores [Salihbegovic et al. 2015]. A Figura 3.9 apresenta a arquitetura do *OpenHAB*.

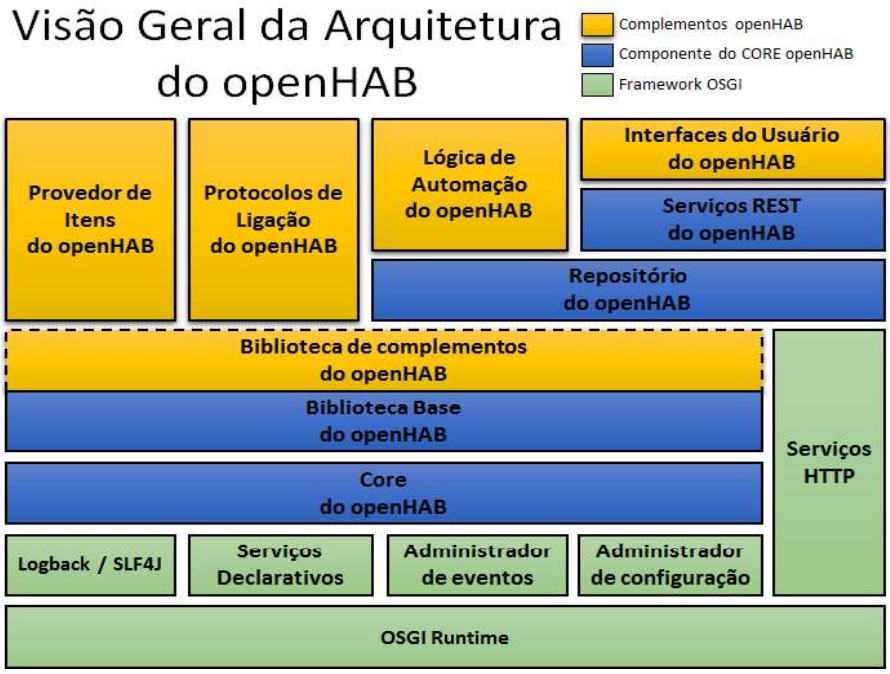


Figura 3.9. Arquitetura do *openHAB*
Fonte: adaptado de Hosek et al. [2014]

Na arquitetura do *openHAB*, os componentes em amarelo são denominados como *openHAB Add-ons*, os quais são componentes que podem ser instalados opcionalmente pelos usuários em tempo de execução. Tais componentes são responsáveis por permitir que o *openHAB* possa se comunicar e se integrar com diferentes tecnologias, de diferentes fornecedores, sendo também conhecidos como componentes de ligação. Por padrão, o *openHAB* vem sem quaisquer componentes de ligação, cabendo ao usuário definir quais componentes utilizar [Hosek et al. 2014].

Os componentes em azul pertencem ao core da solução, os quais são responsáveis pelo gerenciamento dos dispositivos integrados à arquitetura e orquestração dos dados. O componente *openHAB REST Service* é responsável por permitir que a solução possa trocar informações e serviços utilizando o estilo arquitetural *Restful*. O componente *openHAB Repository* é um repositório que mantém o registro de todos os itens instalados. Esse está diretamente conectado ao barramento de eventos do *openHAB*, permitindo-o que mantenha o controle do status atual de todos os itens. O componente *openHAB Base Library* é responsável por manter todas as principais bibliotecas utilizadas pela arquitetura do *openHAB* na implementação dos serviços oferecidos pela arquitetura. Finalmente, o componente *openHAB core* é o responsável por implementar e gerenciar os principais serviços providos pela arquitetura [Hosek et al. 2014].

Os componentes em verde são os responsáveis pela implementação da arquitetura OSGI, permitindo assim que o openHAB possa suportar a inclusão e exclusão de componentes em tempo de execução e sem a necessidade de interrupção dos principais serviços. Além da flexibilidade arquitetural provida pelo OSGI, openHAB possui uma interessante interface gráfica que permite aos usuários inferir regras de negócio ao ambiente, criando assim diferentes comportamentos [Hosek et al. 2014]. A Figura 3.10, abaixo, apresenta a interface gráfica para inferência de regras de negócio no *openHAB*:

OpenHAB é uma importante solução para implementação da IoT em ambientes domesticos. Isso se dá devido à sua ampla biblioteca de itens que permitem que os usuários possam incorporar tecnologias de diferentes fornecedores à solução. Além disso, a principal vantagen do *openHAB* das demais soluções apresentadas é a possibilidade do usuário, por meio de uma interface gráfica, criar regras de negocio para melhor controlar o ambiente.

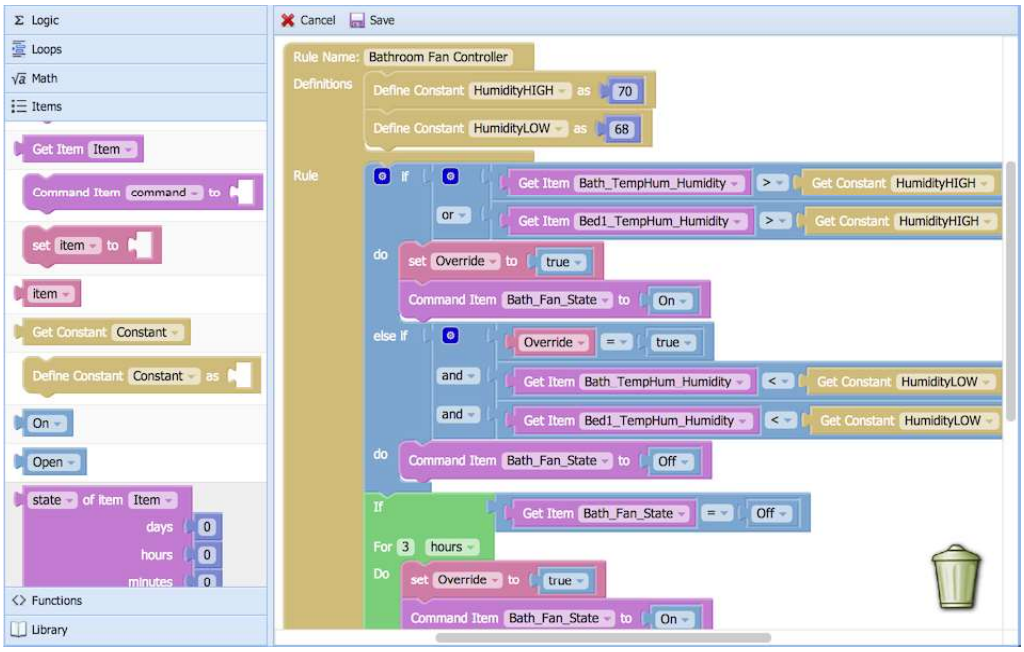


Figura 3.10. Interface gráfica para definição de regras de negócio no openHAB. Fonte: adaptado de Salihbegovic et al. [2015]

3.2.9. *Task Computing*

Task Computing é um *middleware* semântico projetado para interoperar objetos e serviços heterogêneos, permitindo criar ambientes inteligentes baseado em fluxos de tarefa criados pelos usuários por meio de uma interface gráfica de alto nível. No *Task Computing*, cada serviço descoberto na rede é disponibilizado na interface gráfica, como um objeto que pode ser manipulado e, quando interligado com outros objetos, formam fluxos de trabalhos lógicos [Song et al. 2004]. Além disso, o usuário pode combinar e interligar os serviços internos com serviços externos, por exemplo, interligar um serviço de músicas disponível na Internet com um serviço interno de saída de som de uma caixa *Bluetooth*.

Para realizar a descoberta de dispositivos e serviços, *Task Computing* utiliza como base os protocolos UPnP e *Bluetooth-SDP* (*Service Discovery Protocol*). Contudo, os serviços externos devem ser configurados de forma manual pelo usuário final.

Quando um dispositivo é descoberto na rede, suas principais características e serviços são extraídos e adicionados para serem expostos e manipulados por meio da interface gráfica. No *Task Computing*, cada serviço é unicamente identificado por meio de URIs, criadas automaticamente quando os serviços são descobertos e configurados

[Masuoka, Persia e Labrou 2003]. Pela apresentação da Figura 3.11, abaixo, busca-se demonstrar uma visão geral da arquitetura do *middleware Task Computing*:

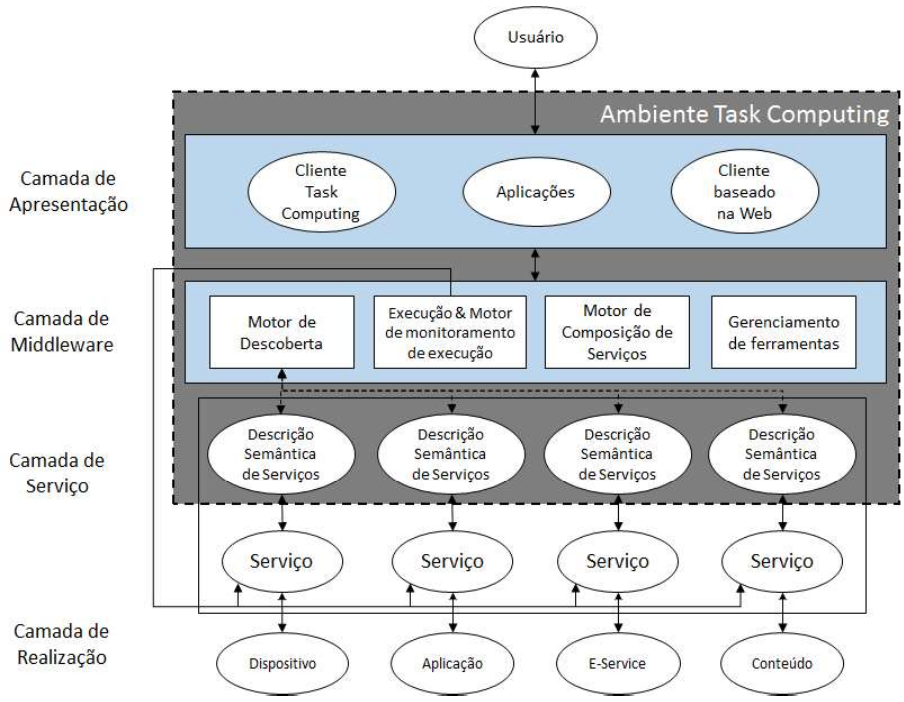


Figura 3.11. Arquitetura do *middleware* Task Computing
Fonte: Adaptado de Song, Cárdenas e Masuoka [2010].

A arquitetura do *middleware Task Computing* é composta de quatro camadas principais:

- Camada de realização: representa as principais fontes de serviço suportadas pelo *Task Computing*. O *middleware* não se limita apenas a dispositivo, mas a diversas fontes provedoras de serviços;
- Camada de serviços: a qual representa o processamento de descoberta e extração de serviços das fontes provedoras e a codificação desses de acordo com o modelo semântico utilizado pela solução [Song, Labrou, Masuoka 2004];
- Camada de *middleware:* que representa as principais funcionalidades providas pelo *middleware* como a descoberta, execução, monitoramento e composição dos serviços; e
- Camada de apresentação: essa camada do *Task Computing* faz uso dos serviços das camadas inferiores, abstraindo do usuário final todas as complexidades nelas existentes, fornecendo aos seus usuários interfaces gráficas de fácil compreensão e acesso. Além disso, fornece aos usuários um ambiente onde as funcionalidades

providas pelas fontes de serviço possam ser manipuladas para montar e executar tarefas complexas em tempo de execução.

A Figura 3.12, abaixo apresentada, busca esboçar a interface para composição de serviços do Task Computing:

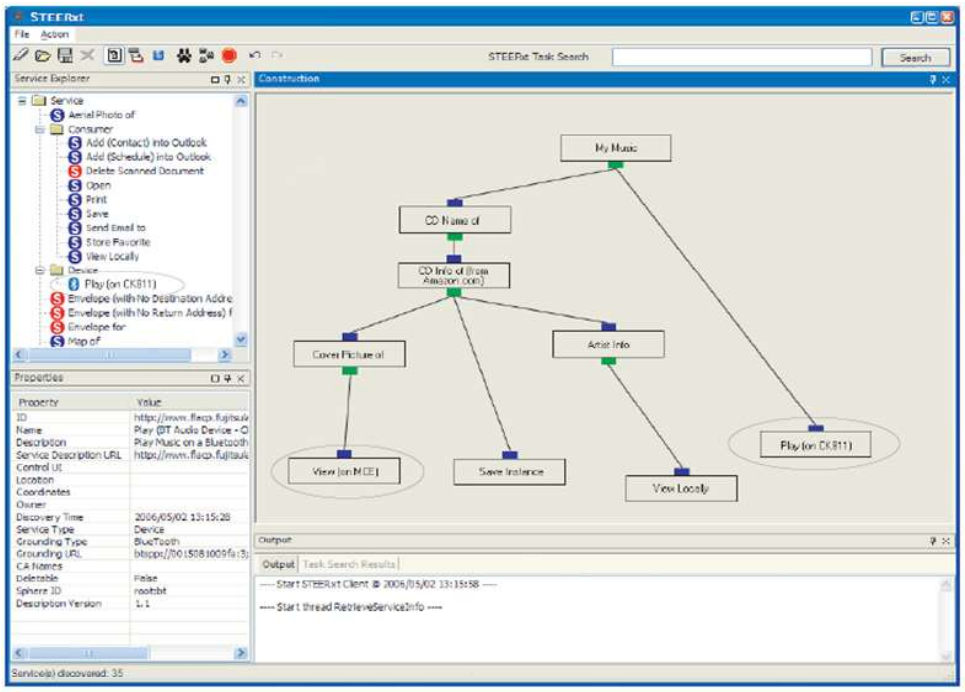


Figura 3.12. IDE para composição de serviços do *Task Computing*.
Fonte: Adaptado de Song, Cárdenas e Masuoka [2010]

A plataforma *Task Computing* fornece uma interface gráfica que permite aos usuários a definição de tarefas por meio da composição dos serviços, além de permitir que possam monitorar a execução das tarefas. Além da interface gráfica, *Task Computing* dispõe de funcionalidades básicas para implementação de ambientes inteligentes para IoT, como interoperabilidade e descoberta de serviços e dispositivos, esquema de nomeação e usabilidade, característica pouco encontrada nas atuais soluções de *middleware* para IoT. No entanto, melhorias podem ser adicionadas ao *Task Computing*, como, por exemplo, a definição de um modelo mais eficiente para criação das tarefas, permitindo que os usuários possam definir não apenas fluxos contínuos de tarefa, mas também os comportamentos ao ambiente com base em um modelo conceitual. Tal modelo permitiria ao usuário criar regras para execução lógica de serviços, tais como: fluxos condicionais, regras de negócio, estados, entrada e saída de dados e fluxo de mensagens trocadas entre os dispositivos e serviços.

3.2.10. GSN

GSN (*Global Sensor Network*) é uma plataforma de *middleware* para integração e interoperabilidade de sensores heterogêneos por meio de sensores virtuais. Sensores virtuais são representados utilizando uma estrutura, em formato XML, que provê todos os parâmetros necessários para representação de um sensor na arquitetura do GSN [Aberer e Hauswirth 2006]. A principal vantagem advinda do uso de sensores virtuais está na sua eficiência em abstrair a heterogeneidade existente nos sensores reais, possibilitando que esses possam interoperar independente das tecnologias de *hardware* e *software* utilizadas.

A arquitetura do GSN segue um modelo baseado em *containers*, responsáveis por fornecer serviços aos demais objetos da arquitetura, sem que esses tenham sido explicitamente programados para suportar tais serviços [Hallstrom et al. 2004]. A Figura 3.13, em seguida, ilustra uma visão geral da arquitetura do GSN e suas principais camadas:

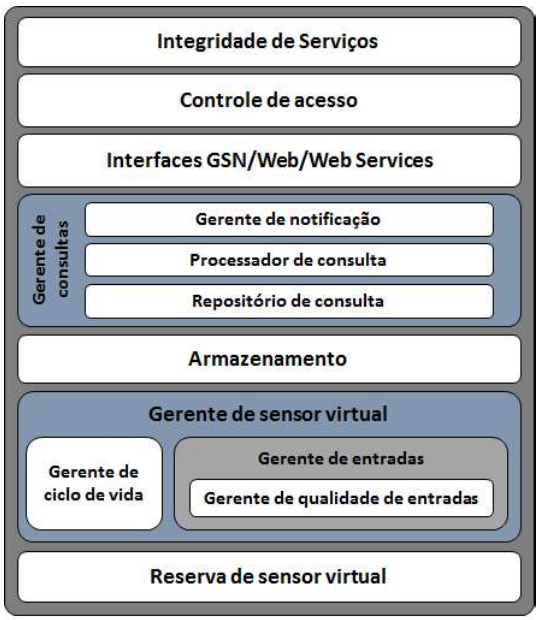


Figura 3.13. Modelo arquitetural do *middleware* GSN
Fonte: Adaptado de Aberer e Hauswirth [2006].

Buscando-se explicar de modo pormenorizado, o gerente de sensor virtual gerencia a entrega dos dados dos sensores e provê meios para administração da infraestrutura do GSN. O subcomponente gerente de ciclo de vida administra os recursos fornecidos aos sensores virtuais e a interação entre eles, enquanto que o gerente de entradas atua na qualidade da comunicação (por exemplo, desconexões, atrasos inesperados e falta de valores) entre os sensores. O Componente armazenamento fornece

a comunicação entre os dados fornecidos pelos sensores e o acesso a esses por meio das interfaces de alto nível, que utilizam o componente gerente de consultas para acesso aos dados. O gerente de consultas é composto de três subcomponentes: o repositório de consultas, que é o responsável por gerenciar todas as consultas registradas no GSN; o processador de consultas, que busca analisar, planejar e executar as consultas SQL; o gerente de notificação, que procura fornecer aos clientes os resultados da execução das consultas. Os demais componentes, localizados no topo da arquitetura, são os responsáveis pelo acesso, controle, integridade e segurança dos dados [Aberer e Hauswirth 2006].

GSN é uma interessante solução para implementação da IoT em ambientes diversificados, em especial devido ao fato não focar um domínio específico, mas, sim, em tecnologias de sensores, o que dá possibilidade ao GSN ser utilizado na implementação da IoT em ambientes diversificados, como na indústria, casas, comercio, transporte e logística, entre outros. Na literatura, GSN é utilizado para implementação de diferentes propostas para IoT, conforme pode ser visto em [Perera et al. 2013] e [Perera et al. 2013b]. Diferente do *Ubiware*, GSN possui um conjunto de interfaces (em nível administrativo) que permitem aos usuários técnicos manipularem soluções baseadas no GSN. Contudo, o GSN não fornece meios que permitam usuários manipular visualmente os sensores presentes no domínio, a fim de definir os diferentes tipos regras de negócio ao qual o ambiente deverá atender. Entretanto, nas primeiras versões do GSN, foi disponibilizada uma interface gráfica para permitir a interação do usuário com os sensores presentes no ambiente, conforme apresentado na Figura 3.14, a seguir, embora essa interface tenha sido descontinuada e removida da arquitetura do GSN:

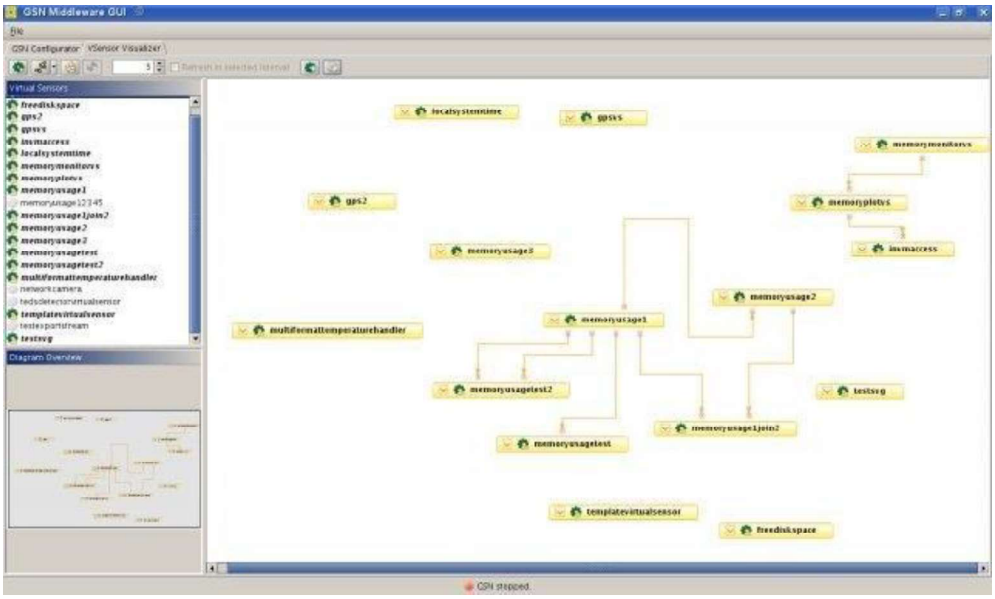


Figura 3.14. Interface gráfica para definição visual de comportamento no GSN. Fonte: Adaptado de Aberer e Hauswirth [2006].

3.2.11. *RestThing*

RestThing é uma infraestrutura Web criada para interconectar e interoperar dispositivos heterogêneos por meio da Internet, utilizando como base o protocolo REST [Qin et al. 2011]. A plataforma *RestThing* permite que desenvolvedores criem soluções para IoT, usando-se os recursos do protocolo HTTP, como os métodos *GET*, *POST*, *PUT* e *DELETE*, com a finalidade de possibilitar tanto a comunicação, como a troca de informações e serviços entre os dispositivos do mundo real, independentemente da interface de comunicação utilizada pelo dispositivo. Outro importante recurso da Web empregado pelo *RestThing* é o uso de URI para identificação única dos recursos e serviços providos pela solução. A seguir, através da Figura 3.15, busca-se aclarar uma visão geral da arquitetura do *RestThing*:

A arquitetura do *RestThing* é composta por quatro camadas principais. A primeira camada, denominada de dispositivos embutidos, representa os recursos de *hardware* (sensores, atuadores e objetos inteligentes) e *software* (sistemas legados, redes sociais), que interagem com a solução. Devido à grande heterogeneidade de dispositivos do mundo real, é necessário que muitos outros dispositivos e aplicações sejam adaptados para que possam ser gerenciados. Assim como o EcoDiF, o *RestThing* também utiliza a implementação de *drivers* para dispositivos. *Drivers* permitem expor as funcionalidades

dos dispositivos pela Web e estabelecer, então, uma ligação entre o *RestThing* e o dispositivo.

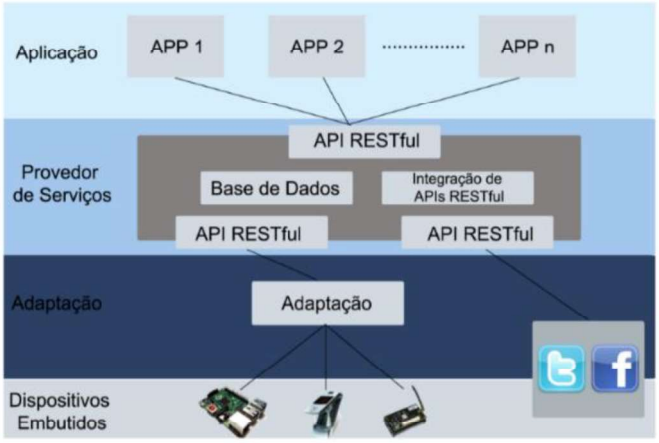


Figura 3.15. Arquitetura em camadas do RestThing
Fonte: Pires et al. [2015].

A camada de adaptação representa o processo manual que o usuário deve realizar para possibilitar que dispositivos embutidos possam ser integrados ao RestThing e assim prover serviços por desse.

A camada provedor de serviços é o ponto central para acesso a infraestrutura do *RestThing*. A API *Restful* é o fator chave da plataforma para esconder a heterogeneidade existente nos dispositivos, provendo uma interface acessível para acesso aos dispositivos e serviços. Por fim, a camada aplicações representa as aplicações externas das quais os recursos providos fazem uso pelos dispositivos [Qin et al. 2011].

Cabe ressaltar que a principal vantagem apresentada pelo *RestThing* é o uso de recursos Web, com a finalidade de se esconder a heterogeneidade dos dispositivos mais facilmente, o que possibilita que possam interoperar por meio da Internet. Além disso, o uso de *drivers* permite que, independente da interface de comunicação utilizada, um dispositivo possa ser adaptado, disponibilizado e gerenciado pelo *RestThing*. No entanto, diferentemente do EcoDiF, *RestThing* não possui grandes contribuições quanto à implementação e uso de interfaces de alto nível para facilitar o uso da solução.

3.2.12. Avaliação I - *Middlewares* x Desafios

Essa seção tem por objetivo analisar as plataformas de *middlewares*, a fim de se verificar o atendimento dos critérios supracitados. A Tabela 3.1 está organizada da seguinte

forma: o campo "***Middleware***" descreve o nome das plataformas de *middleware* avaliadas; o campo "**Desafios**" apresenta os desafios apresentados na seção 3.1 (interoperabilidade, conectividade, escalabilidade, esquema de nomeação, flexibilidade, portabilidade, usabilidade, segurança e privacidade, gerência de grande volume de dados, contexto e descoberta de dispositivos e serviços); e o campo **referência** apresenta as principais bibliografias utilizadas para realizar a avaliação.

Cabe destacar que a avaliação foi realizada com fundamento em 4 (quatro) parâmetros: (●) critério atendido por completo, utilizado quando a aplicação utiliza de abordagem que permitem superar totalmente o desafio avaliado; (◖) critério atendido parcialmente, utilizado quando a aplicação faz uso de uma abordagem que permite superar o desafio, contudo não em sua totalidade; (○) critério não atendido, utilizado quando uma aplicação não faz uso de qualquer abordagem para superar um desafio e, (Ø) não especificado, utilizado quando não há registro na literatura que evidenciem a superação de um desafio por uma aplicação.

Tabela 3.1. Tabela de avaliação de desafios atendidos.

Middleware	Desafios												Referências
	Interoperabilidade	Conectividade	Escalabilidade	Nomeação	Flexibilidade	Reusabilidade	Portabilidade	Usabilidade	Segurança e Privacidade	Grandes Volume de dados	Gerência de Contexto	Descoberta de dispositivos	
ASPIRE	●	○	●	●	●	●	●	●	○	○	○	○	[Anggorjati et al. 2010], [Kefalakis et al. 2009], [Gama, Touseau e Donsez 2012], [Kefalakis et al. 2011], [Bandyopadhyay et al. 2011]
Ebbits	●	●	○	●	●	●	●	○	●	Ø	●	●	[Ebbits 2011], [Kostelnik, Sarnovsk e Furdik 2011]
Ubiware	●	●	Ø	Ø	●	●	●	○	○	Ø	●	●	[Katasonov et al. 2008], [Terziyan, Zhovtobryukh, Katasonov 2009], [Nagy et al. 2009], [Pires et al. 2015]
EcoDiF	●	●	●	●	●	Ø	●	◖	●	●	Ø	◖	[Pires et al. 2015], [Pires et al. 2014]
LinkSmart	●	●	○	●	●	●	●	○	●	○	●	●	[Hydra 2007], [Sarnovský et al. 2008], [Pires et al. 2015]
Socrades	●	●	●	●	○	○	●	●	●	○	○	●	[De Souza et al. 2008], [Spiess et al. 2009], [Bandyopadhyay et al. 2011], [Wu et al. 2012], [Warriach 2013]
SM4ALL	●	●	Ø	●	○	○	●	◖	○	○	○	●	[Warriach et al. 2014], [Warriach et al. 2011], [Warriach et al. 2010] e [Warriach 2013]
OpenHAB	●	●	○	●	●	●	●	●	Ø	○	○	●	[Hosek et al. 2014], [Shrestha, Kubler e Främling 2014] e [Salihbegovic et al. 2015]
Task Computing	●	○	○	●	○	○	●	●	○	○	○	●	[Song, Cárdenas e Masuoka 2010] e [Song et al. 2004]
GSN	●	●	●	●	●	●	●	○	●	Ø	○	●	[Aberer e Hauswirth 2006], [Bandyopadhyay et al. 2011], [Aberer, Hauswirth, Salehi 2007]
RestThing	●	●	○	●	○	Ø	●	◖	○	○	○	○	[Qin et al. 2011],[Pires et al. 2015]

De acordo com a Tabela 3.1 supramencionada, é possível observar que todas as soluções apresentadas atendem, de alguma forma, os principais desafios que devem ser superados no projeto de *middleware* para IoT, como, por exemplo, a *interoperabilidade*, pois, uma vez alcançados, possibilitam que dispositivos e tecnologias heterogêneas possam trocar informações. *Esquemas de nomeação*, para permitir que dispositivos e serviços sejam unicamente identificados e *portabilidade* para que a solução seja executada, independentemente da plataforma de sistema operacional utilizada pelo dispositivo.

A solução *Ubiware*, apesar de atender ao desafio de esquema de nomeação, foi avaliada como não especificada, pois, nas literaturas revisadas, não se tem especificação quanto ao tipo de esquema de nomeação utilizada pela solução, como IP, Código de barras, RFID, URI, entre outros.

Apesar das soluções apresentadas atenderem a importantes desafios, ainda há outros que precisam ser melhor explorados, como por exemplo, *escalabilidade*, que possibilita que uma solução possa crescer de acordo com a demanda de dispositivos que, porventura, devam surgir no futuro. *Usabilidade*, que possibilita aos usuários manipular as soluções, tanto em questões arquiteturais, aumentando a possibilidade de modificação e extensão das soluções, quanto em ambiente monitorado, modelando e definindo o comportamento de diferentes ambientes para IoT, com base em interfaces gráficas e modelo conceitual. Finalmente, segurança e privacidade, promovendo o acesso seguro e restrito aos dispositivos e serviços oferecidos pelas soluções para IoT.

3.3. Tecnologias de *Middleware* para IoT

O desenvolvimento de soluções de *middleware* para IoT não é uma tarefa trivial, pois, para criá-las, é necessário que se tenha grande conhecimento técnico sobre uma variedade de tecnologias, como protocolos, dispositivos, padrões semânticos, entre outras. Dessa forma, as seções, a seguir, destinam-se a apresentar as principais tecnologias utilizadas no desenvolvimento de soluções de *middleware* para IoT.

Primeiramente serão apresentados os principais protocolos utilizados para transporte de mensagens entre os dispositivos como os protocolos REST, SOAP, MQTT (*Message Queue Telemetry Transport*) e CoAP (*Constrained Application Protocol*). Em

seguida, os protocolos utilizados para descoberta e configuração automática de dispositivos e serviços na rede serão apresentados como UPnP, *Bluetooth* SDP, SLP (*Service Location Protocol*) e o JINI.

Após a apresentação dos protocolos, serão apresentados os principais padrões semânticos utilizados pelas soluções de *middleware*, como a OWL e o RDF, para se definirem os modelos arquiteturais e/ou os dados interoperáveis.

Em seguida, as principais tecnologias físicas de comunicação sem fio utilizadas na IoT serão apresentadas, como Wi-Fi, ZigBee, RFID. E, para concluir, serão apresentados os dispositivos ou "Coisas" do mundo real que, provavelmente, estarão cada vez mais presentes no contexto da IoT, seja diretamente, fornecendo serviços e dados, ou indiretamente, auxiliando na inclusão de dispositivos antigos ou legados.

3.3.1. Protocolos de Transporte de Mensagens

Um dos principais desafios para as soluções de *middleware* IoT é o de possibilitar a troca de dados e serviços entre dispositivos heterogêneos, independente da tecnologia de *hardware* ou *software* utilizada. Esta seção busca apresentar, resumidamente, os principais protocolos de comunicação utilizados em soluções de *middleware* para IoT:

i) **REST (*Representational State Transfer*) –** é um estilo arquitetural projetado por *Roy Fielding*, um dos principais autores da especificação HTTP, que utiliza os mais importantes mecanismos disponíveis na infraestrutura do protocolo HTTP, como os métodos GET, POST, PUT e DELETE, com a finalidade de se fazer o uso de recursos remotos [Rodriguez 2008]. Assim, o método GET é utilizado para recuperar o estado atual de um recurso, o POST atualiza o estado de um recurso, o PUT cria um novo recurso e o DELETE remove um recurso existente. Atualmente, o REST tem sido um dos principais protocolos utilizados pelos desenvolvedores de soluções de *middleware* para IoT para transporte de dados. Isso se deve ao fato do REST ser leve e de fácil adaptabilidade, principalmente em dispositivos compatíveis com os protocolos IP e HTTP. Além disso, os principais *frameworks* utilizados para implementação de serviços Web, como o

Apache Axis2[8] e CXF[9], possuem total suporte à arquitetura REST, [Richardson e Ruby 2008].

ii) **SOAP (*Simple Object Access Protocol*) –** é um protocolo leve que permite a troca de mensagens entre dispositivos em um ambiente distribuído descentralizado, podendo ser usado em combinação com uma variedade de protocolos, incluindo o HTTP [Mitra e Lafon 2007]. Uma mensagem SOAP é formada por três elementos: o envelope, que define o conteúdo da mensagem e informa como processá-la; o cabeçalho, o qual delimita um conjunto de regras de codificação para os tipos de dados; e, por último, o corpo, que define a mensagem em si, podendo ser a chamada a um método ou o resultado de uma chamada. A principal vantagem do SOAP com relação ao REST é o seu método de transporte genérico, podendo-se usar o HTTP para o envio de requisições, além de outros protocolos, como o SMTP (*Simple Mail Transfer Protocol*) e o JMS (*Java Messaging Service*) [Freire 2002]. Tal fato o torna um forte candidato para ser utilizado em qualquer sistema IoT.

iii) **MQTT (*Message Queue Telemetry Transport*) –** é um protocolo cliente/servidor para transporte de mensagens, baseado em uma arquitetura *publish/subscribe* e voltado para dispositivos restritos e redes inseguras, com baixa largura de banda e alta latência. Criado por *Andy Stanford-Clark* da IBM e *Arlen Nipper* da *Eurotech,* MQTT é um protocolo confiável e de fácil implementação [Lee et al. 2013]. Tais características tornam o MQTT ideal para uso em situações onde existem restrições com a comunicação M2M (*Machine-to-Machine*), o que geralmente inclui dispositivos IoT. O MQTT é geralmente empregado em cenários em que a implementação exige o mínimo de linhas de código e o máximo da largura de banda [Martins e Zem 2015]. Atualmente, a especificação do MQTT apresenta uma série de características interessantes, como: i) a possibilidade desse ser executado sobre o protocolo TCP/IP ou sobre outro com a mesma finalidade; ii) pequena sobrecarga de transporte e trocas minimizadas de protocolos para reduzir o tráfego na rede; e iii) um mecanismo que notifica partes interessadas, quando um cliente se desconecta da rede de modo anormal. Por meio do uso de uma arquitetura *publish/subscribe*, os dispositivos MQTT se conectam

[8] http://axis.apache.org/axis2/java/core/
[9] http://cxf.apache.org/

a um servidor denominado *Broker*, onde as mensagens a serem transmitidas são publicadas para um endereço denominado *Tópico*. Assim, os clientes podem se sobrescrever para vários tópicos, tornando-se capazes de receber as mensagens que outros clientes publicam nesse tópico [Jaffey 2014].

iv) **CoAP (*Constrained Application Protocol*) –** O CoAP é um protocolo para se realizar transferência de dados na web, compatível com o HTTP e criado para uso em dispositivos eletrônicos com restrição de energia e memória, tais como: sensores, atuadores e outros componentes que precisam ser controlados e supervisionados remotamente [Shelby 2014]. O protocolo CoAP classifica as mensagens em quatro tipos: i) confiável**,** mensagens que precisam ser confirmadas no destino; ii) não-confiável, aquelas que não necessitam de confirmação de recebimento, útil no caso de aplicações que necessitam receber leituras constantes de um sensor (por exemplo, dados de temperatura) em um espaço muito curto de tempo. A perda de uma ou outra mensagem não é motivo para preocupações; iii) confirmação, as mensagens que confirmam o recebimento de uma outra mensagem do tipo confiável (*acknowledgment*); e iv) reset**,** que indica se outra mensagem (Confiável ou Não-Confiável) foi recebida, mas, por falta de algum contexto, não pôde ser devidamente processada. Nenhuma das soluções apresentadas na Seção 3.2 faz uso do protocolo CoAP, devendo-se ao fato do baixo suporte de bibliotecas para implementação do CoAP [Pires et al. 2015].

3.3.2. Protocolos para Descoberta de Dispositivos

Com o advento da IoT e a inclusão cada vez maior de dispositivos do mundo real no mundo virtual, acredita-se que o número de serviços oferecidos por meio da Internet sofra, no futuro, um aumento significativo. Dessa forma, uma solução de *middleware* para IoT deve prover mecanismos para automatizar a localização e configuração dinâmica de serviços. Neste contexto, o uso de protocolos de descoberta de dispositivos e serviços tem sido adotado pela maioria das soluções de *middleware* para IoT.

Esse tipo de protocolo estabelece um conjunto de mensagens usadas para descobrir, configurar e registrar um dispositivo e seus serviços, tornando-o visível ao uso dos demais participantes da rede. Adiante, serão apresentados alguns protocolos que podem

ser usados como alternativa de implementação da descoberta de dispositivos e serviços em soluções para IoT:

i) **UPnP (*Universal Plug and Play*) –** é uma arquitetura de rede distribuída e aberta, que utiliza TCP/IP e as tecnologias da Web para conectividade de aparelhos inteligentes, dispositivos sem fio e computadores, a fim de formar uma rede P2P (*Peer-to-Peer*) desses dispositivos. É projetada para trazer conectividade de fácil utilização, flexível e baseada em padrões, podendo ser utilizada em casas, pequenas empresas, escritórios e espaços públicos [Cheng 2000]. Por ser uma tecnologia de fácil utilização e possuir grande aceitabilidade na indústria e no comércio [Song, Cárdenas e Masuoka 2010], UPnP tem sido utilizado como o principal protocolo para implementação de descoberta e gerenciamento de dispositivos e serviços em diversas soluções para IoT, como, por exemplo, nos middlewares *LinkSmart*, *SM4ALL* e *Task Computing*. Uma das vantagens da arquitetura UPnP é que esta utiliza protocolos comuns como HTTP, SOAP e XML (*eXtensible Markup Language*) para transferência dos dados. Além disso, UPnP requer menos recursos de *hardware*, quando comparado com outros protocolos de descoberta de serviço [Kim et al. 2002].

ii) **SDP (*Service Discovery Protocol*) –** é um protocolo pertencente à pilha do *Bluetooth*. O protocolo SDP fornece um meio pelo qual os serviços executados em diferentes dispositivos *Bluetooth* possam descobrir a existência um do outro, além de trocarem informações para determinar suas características [Bluetooth 2014]. No SDP, cada dispositivo é representado pelo seu perfil (*profile*), que contém informações referentes ao dispositivo e seus serviços. Cada *profile* é formado por um identificador de 16 ou 32 bits, denominado UUID. Toda consulta SDP é baseada nos UDDIs [Lee e Helal 2002]. Devido à larga utilização do *Bluetooth* em dispositivos e objetos do cotidiano, é importante que uma solução para IoT forneça suporte para descoberta de dispositivos com base nessa tecnologia.

iii) **SLP (*Service Location Protocol*) –** é um protocolo de localização de recursos, como impressoras, unidades de disco e bancos de dados em uma rede [IETF 2015]. Diferente do *Bluetooth* SDP, SLP é utilizado por dispositivos de anunciar serviços em uma rede local por meio de URLs. A arquitetura do SPL é composta de três componentes principais: o agente de usuário (*User Agent* - UA), o qual

executa a descoberta dos serviços; o agente de serviço (*Service Agent* - SA), responsável por anunciar a localização e características dos serviços; e o agente de diretório (*Directory Agent* - DA), que coleta informações recebidas dos SAs e responde as requisições de serviços dos UAs [IETF, 2009]. Devido à sua simplicidade e à possibilidade de ser utilizado em dispositivos com pouco poder de processamento e memória, o protocolo SLP é uma alternativa para implementação de mecanismos de descoberta em soluções de *middleware* para IoT.

iv) **JINI (*Java Intelligent Network Infrastructure*) –** é um protocolo para gerenciamento de redes e descoberta de serviços, desenvolvida pela *Sun Microsystem*, com suporte na linguagem de programação Java [Allard et al. 2003]. O protocolo Jini fornece um ambiente distribuído para possibilitar que dispositivos heterogêneos se comuniquem, sendo necessário apenas que esses dispositivos publiquem seus serviços para que os demais dispositivos os localizem e, assim, os utilizem [Gupta et al. 2002]. Existem três entidades principais utilizadas por esse protocolo: provedores de serviço, serviço de pesquisa e usuários. Para que um serviço possa ser localizado, os provedores de serviço são obrigados a registrar todos os seus trabalhos em um serviço de pesquisa. Esse, por sua vez, envia os serviços registrados para que os usuários possam tomar conhecimento e, se necessário, fazer uso dos serviços. Uma desvantagem do Jini é a necessidade recursos de processamento e armazenamento elevada, deixando de lado dispositivos que não atendam a esses requisitos. Contudo, na literatura existem trabalhos propondo o uso do Jini em dispositivos domésticos, conforme Al-Muhtadi et al. [2000]. Apesar da necessidade de adaptações, o protocolo Jini pode ser uma alternativa para criação de mecanismos de descoberta de dispositivos e serviços para sistemas IoT.

3.3.3. Modelos Semânticos

Modelos semânticos são modelos que representam um determinado domínio do mundo real ou partes dele, utilizando, para isso, um conjunto de entidades, símbolos, relações lógicas e regras [Date 1996] e [Gruber 1993].

Na IoT, devido à grande diversidade tecnológica e heterogênea diretamente relacionada ao desenvolvimento de soluções para IoT, como protocolos, linguagens de programação, dispositivos e padrões arquiteturais, há uma grande dificuldade, por parte dos programadores, em criar soluções que consigam abstrair toda a complexidade existente entre essas diversas tecnologias e prover mecanismos para lidar com essas diferenças. A solução largamente adotada para contornar tal problema é o uso de modelos semânticos utilizados em diferentes partes da arquitetura das soluções de *middleware*, conforme em [Song, Cárdenas e Masuoka 2010] [Eisenhauer, Rosengren e Antolin 2010] e [Katasonov 2008], provendo, dessa forma, estruturas mais simples, abstratas e interoperáveis. A seguir, são apresentadas as linguagens semânticas mais utilizadas na concepção de soluções para IoT:

i) **OWL (*Ontology Web Language*) –** é uma linguagem para definir e instanciar ontologias na Web [Mcguinness et al. 2004]. A OWL possui três sub-linguagens incrementais projetadas para serem usadas por diferentes comunidades de implementadores e usuários: i) *OWL-Lite*, uma sub-linguagem da OWL-DL que usa somente algumas características da OWL e, portanto, com poder de expressividade limitado; ii) *OWL-DL*, usada por usuários que queiram o máximo de expressividade, incluindo todas as construções da linguagem OWL, mas podendo ser usadas somente sob certas restrições; e iii) *OWL-Full*, que é usada por usuários que queiram o máximo de expressividade e independência sintática, sem nenhuma garantia computacional. Nos *middlewares* LinkSmart, EBBITS, *Task Computing* e SM4ALL, a OWL é fortemente utilizada na criação de modelos semânticos para auxiliar no gerenciamento dos dados e interoperabilidade dos dispositivos.

ii) **RDF (*Resource Description Framework*) –** O RDF é composto de uma arquitetura genérica que utiliza sintaxe da linguagem XML para intercâmbio e tratamento dos metadados [Miller 1998]. O uso da linguagem XML provê ao RDF independência, extensibilidade, validação, legibilidade humana e a capacidade de representar estruturas complexas [Antoniou e Van Harmelen 2004]. O principal objetivo do RDF é representar semanticamente informações de recursos presentes na web, facilitando o processamento e a busca dos recursos com base em suas semânticas. No RDF, tudo é visto como recurso, incluindo páginas web, pessoas, dispositivos, entre outros. RDF usa as seguintes

terminologias para descrever um recurso: i) sujeito, responsável por identificar o objeto de declaração ou recurso por meio de uma URI; ii) predicado, identificador das propriedades do objeto declarado; e iii) objeto, que identificar o valor de uma propriedade [Vertan e Merkmale 2004].

Por ser uma solução extensível, grandes projetos fazem uso do RDF como base para o desenvolvimento de linguagens mais expressivas, como a linguagem OWL, largamente utilizada no desenvolvimento de soluções para IoT. Os *middleware* UBIWARE é um exemplo de solução de *middleware* que faz uso do padrão RDF.

3.3.4. Tecnologias de Comunicação Sem Fio

A comunicação entre dispositivos, no paradigma de IoT, pode ser realizada de duas formas. A primeira se dá por meio de interfaces de comunicação cabeada, que, apesar de serem extremamente eficientes, necessitam, na maioria das vezes, de modificações no ambiente. A segunda, por meio de interfaces de comunicação sem fio, que não são tão eficientes quanto às interfaces cabeadas, mas não necessitam de que o ambiente seja modificado. Além disso, a comunicação sem fio é o principal meio utilizado para conexão e troca de dados entre dispositivos na IoT. Para facilitar o entendimento das tecnologias envolvidas com a IoT, essa seção tem por objetivo apresentar as principais tecnologias de comunicação sem fio utilizadas neste paradigma:

i) ***Wi-Fi* (*Wireless Fidelity*) -** é uma tecnologia de comunicação sem fio de alta velocidade, com base no padrão IEEE 802.11, criado para permitir que um dispositivo possa trocar informações e serviços em uma rede local de computadores ou por meio da Internet [Warriach et al. 2014]. *Wi-Fi* permite que dispositivos possam trocar informações a uma taxa alta de dados e a distâncias que podem chegar de 100 a 300 metros, utilizando-se, para isso, a banda não-licenciada ISM (*Industrial Scientific and Medical*) [Warriach 2013]. Devido à larga adoção do *Wi-Fi* em dispositivos domésticos, como televisores e câmeras, existem na literatura diversas soluções que fazem uso do *Wi-Fi* como principal interface de comunicação no desenvolvimento de ambientes inteligentes, conforme pode ser visto em [Zhi-Gang e Wei 2012] e [Li et al. 2011].

ii) ***Bluetooth*** **–** é uma tecnologia especificada com base no padrão IEEE 802.15. B*luetooth* utiliza a banda de 2.4GHz de frequência de rádio, não licenciada para se comunicar, enviar e receber dados a uma distância limite de até 10 metros [Sairam, Gunasekaran e Redd 2002]. Contudo, quando estendida, pode chegar ao limite máximo de 100 metros [Ramlee, Tang e Ismail 2012]. A principal desvantagem do B*luetooth* em relação às demais tecnologias sem fio é que, por estar constantemente ativada para busca automática de dispositivos habilitados, o seu consumo de energia tende a ser maior em relação a outras tecnologias, como o *ZigBee*. No entanto, na literatura é possível encontrar diversas propostas de projeto de baixo custo para IoT que fazem uso do *bluetooth*, como em [Piyare e Tazil 2011], [Zhao 2013] e [Long-Gang 2011].

iii) ***ZigBee*** **-** é uma promissora tecnologia de comunicação sem fio de curta distância, com base no padrão IEEE 802.15.4. É especialmente adequada para uso em dispositivos que não necessitam de uma transferência de dados muito elevada, como sensores, atuadores, medidores de energia, entre outros [Warriach 2014]. Devido à sua baixa taxa de transmissão, *ZigBee* é uma das tecnologias sem fio que menos consome recursos de energia, podendo ficar ativa por um longo período de tempo [Ergen 2004]. Assim como as demais tecnologias sem fio apresentadas, ZigBee utiliza a banda não-licenciada ISM para transmissão de dados. ZigBee é uma tecnologia largamente utilizada na indústria, mas pouco utilizada em ambientes inteligentes domésticos. Qu et al. [2012], Fan et al. [2011], Ha [2009] e Hribernik et al. [2011] apresentam resultados que demonstram a viabilidade do uso de ZigBee em ambientes domésticos.

iv) **RFID (*Radio Frequency IDentification*)** - é uma tecnologia de rádio frequência largamente utilizada na IoT para identificação de dispositivos e coleta automática de dados [Juels 2006] e [Liu e Zhou 2012]. Segundo Yang, L., Yang, S. e Plotnick [2013], a ampla utilização de RFID se deve ao apoio empresarial em tornar as RFID mais baratas e viáveis na prática. RFID são divididas em três classes: i) etiquetas passivas; ii) etiquetas ativas e iii) etiquetas semi-passivas [Want 2006]. As etiquetas passivas possuem tempo de vida ilimitado, não dependem de uma fonte de energia para funcionar e são ativadas por meio de um sinal de rádio frequência emitido por um leitor de RFID. As etiquetas ativas são totalmente dependentes de baterias e possuem um transceptor de rádio que

permitem emitir sinais a longas distâncias, não necessitando de um leitor para ativá-las [Ni et al. 2004]. As etiquetas semi-passivas são dispositivos que possuem as características de ambas as etiquetas, ou seja, podem ser ativadas por meio de sinal de rádio frequência ou por meio da emissão do seu próprio sinal [Angeles 2005]. Na literatura, encontram-se diversos trabalhos e propostas que fazem uso das etiquetas RFIDs para controle e identificação de dispositivos e/ou objetos do cotidiano, como em [Kortuem et al. 2010], [Welbourne et al. 2009] e [Domingo 2012].

3.3.5. Coisas

Dispositivos ou "coisas" são as principais entidades para existência do paradigma da IoT no mundo real. Atualmente existem diversos e diferentes tipos de dispositivos que deverão no futuro estar integrados à IoT. Para tanto, ainda há grandes desafios a serem alcançados para que tais dispositivos possam se tornar parte da IoT, compartilhando recursos como armazenamento e processamento e provendo/consumindo serviços. Assim, partindo do princípio da importância dos dispositivos para existência da IoT, essa seção resume as classes principais de dispositivos que farão parte da IoT, são elas:

i) **Sensores sem Fio** - é a principal tecnologia da IoT, responsável por representar os eventos ocorridos no mundo real no mundo virtual. Redes compostas por sensores e atuadores interligados serão cada vez mais comuns em ambientes IoT. Os atuadores serão responsáveis por executar ações na rede, e os sensores por coletar dados e eventos para que esses possam ser enviados aos usuários por meio de um nó sensor especial denominado como sorvedouro (ou *gateway*) [Perera et al. 2013]. Tais redes, comumente denominadas de Redes de Sensores sem Fio estarão tão presentes na IoT que atualmente há uma grande dificuldade por parte de alguns usuários em entender a diferença entre RSSF e IoT. Segundo Perera et al. [2014], a IoT compreende tudo que uma RSSF possui e ainda uma espessa camada de *software* como *middleware*, *frameworks*, APIs e outros diversos componentes de *software* [Perera et al. 2014]. Na literatura existem diversos trabalho propondo técnicas para integração de RSSF a IoT, como em [Alcaraz 2010], [Hong et al. 2010], [Kouche 2012] e [Shah e Ambareen 2014].

ii) **Dispositivos inteligentes domésticos –** refere-se a todo e qualquer dispositivo doméstico provido de poder computacional para processar dados e interface de comunicação para prover e consumir serviços por meio da Internet. Como exemplo de dispositivos inteligentes domésticos, pode-se citar: *smart TVs*, *smart freezers*, *smartphones*, câmeras e sensores. Apesar de já serem criadas com funcionalidades que permitem comunicação por meio da Internet, a utilização de tais dispositivos não garante uma integração total entre eles, uma vez que esses são criados por diferentes fabricantes, que utilizam diferentes protocolos e tecnologias. No entanto, tal problema pode ser resolvido por meio do uso de soluções que garanta a interoperabilidade entre esses dispositivos, como por exemplo, o uso de *middlewares* [Perera et al. 2012] [Chen et al. 2000].

iii) **Dispositivos Legados –** Esta classe caracteriza os dispositivos que não possuem a capacidade de processar, armazenar dados ou se comunicar com outros dispositivos por meio de uma rede local ou pela Internet. Televisores, ventiladores, cafeteiras e geladeiras são exemplos de dispositivos legados. Para que esta classe de dispositivos possa ser integrada a IoT, é necessário que se faça uso de sensores e/ou plataformas de *hardware* para permitir que esses possam ser controlados pela Internet. Em [Doukas 2012], [Hribernik et al. 2011], [Allan 2011] e [Mehta Karankumar, Mehta Shreya e Raviya Kapil 2014] apresentam interessantes projetos para integração de dispositivos legados por meio da Internet utilizando tomadas inteligentes (*smart plugs*) e as plataformas programáveis *Arduino* e *Raspberry*. Entretanto, soluções como *EcoDIF, RestThing e OpenHAB* podem ser adaptadas a diferentes tecnologias de comunicação com o uso de *drivers* e *plug-ins*.

3.3.6. Avaliação II – *Middlewares* x Tecnologias

Com objetivo de identificar as tecnologias utilizadas na concepção das soluções de *middleware* apresentadas, a Tabela 3.2 a seguir está organizada da seguinte forma. O campo "***Middleware***" apresenta as soluções avaliadas, em seguida o campo "**Tecnologias**" apresenta as tecnologias supracitadas. Por fim, as referências utilizadas para avaliação das soluções são apresentadas pelo campo "**Referência**".

Cabe destacar que a avaliação foi realizada com fundamento em 3 (três) parâmetros: (●) critério atendido por completo, utilizado quando a aplicação faz uso da tecnologia apresentada; (○) critério não atendido, utilizado quando uma aplicação não faz uso da tecnologia apresentada e, (Ø) não especificado, utilizado quando não há registro na literatura que evidenciem que a aplicação faça uso da tecnologia apresentada.

Tabela 3.2. Tecnologias suportadas pelos *middlewares* avaliados.

Middleware	Tecnologias																	Referências
	Protocolos de transporte de mensagens				Protocolos de descoberta de dispositivos				Padrões Semânticos		Tecnologias sem fio				Coisas			
	REST	SOAP	MQTT	CoAP	UPnP	SDP	SLP	JINI	OWL	RDF	WiFi	Bluetooth	ZigBee	RFID	RSSF	Dispositivos Inteligentes	Dispositivos legados	
ASPIRE	○	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	[Anggorjati et al. 2010], [Kefalakis et al. 2009], [Kefalakis et al. 2011], [Bandyopadhyay et al. 2011]
Ebbits	●	●	○	○	●	●	○	○	●	●	●	●	●	●	●	○	○	[Ebbits 2011]
Ubiware	○	●	○	○	○	○	○	○	○	●	○	○	○	●	●	○	○	[Katasonov et al. 2008], [Bandyopadhyay et al. 2011], [Khriyenko, Nagy 2011]
EcoDiF	●	○	○	○	Ø	Ø	Ø	Ø	○	○	●	●	●	Ø	●	○	○	[Pires et al. 2015], [Pires et al. 2014]
LinkSmart	●	●	○	○	●	●	○	○	●	●	●	●	●	●	●	●	○	[Hydra 2007], [Bandyopadhyay et al. 2011]
Socrades	○	●	○	○	○	○	○	○	Ø	Ø	○	○	○	●	●	○	○	[De Souza et al. 2008], [Bandyopadhyay et al. 2011]
SM4ALL	●	●	○	○	●	●	○	○	●	○	●	●	●	●	●	●	○	[Warriach 2014]
OpenHAB	●	Ø	●	○	Ø	●	○	○	○	○	●	●	●	●	●	●	●	[Hosek et al. 2014], [Shrestha, Kubler e Främling 2014] e [Salihbegovic et al. 2015]
Task Computing	○	●	○	○	●	●	○	○	●	○	●	●	○	○	●	●	○	[Song, Cárdenas e Masuoka 2010] [Song et al. 2004]
GSN	○	●	○	○	○	○	○	○	○	○	●	○	○	●	●	○	○	[Aberer e Hauswirth 2006]
RestThing	●	○	○	○	○	○	○	○	○	○	●	●	●	●	●	○	○	[Qin et al. 2011], [Pires et al. 2015]

Com base na avaliação apresentada, é possível perceber a grande quantidade de tecnologias aplicadas no desenvolvimento de soluções de *middleware* para IoT. A combinação dessas diversas tecnologias torna o desenvolvimento de soluções para IoT um desafio para usuários com pouco conhecimento no desenvolvimento desse tipo de solução. Contudo, essa combinação tecnológica deve-se ao fato da IoT ser altamente heterogênea, ou seja, a IoT é uma arquitetura que deve possibilitar que dispositivos, que utilizam diferentes protocolos, padrões e tecnologias de comunicação possam trocar informações e serviço entre si em uma rede totalmente interoperável.

Além da grande quantidade de tecnologias que podem ser utilizadas e combinadas no desenvolvimento solução para IoT, é também possível perceber que as soluções apresentadas não se concentram em atender a todos os protocolos, dispositivos e

tecnologias atualmente disponíveis pelo mercado. Ao invés disso, tais soluções concentram esforços em atender a um conjunto específico de tecnologias, como por exemplo, o *aspire*, *Socrades*, *sm4all*, *Task Computing*, *GSN* e *RestThing*. Contudo, existem soluções que fazem uso de mecanismos capazes de permitir que novas tecnologias possam ser adaptadas a arquitetura das soluções como por exemplo, o uso do OSGi, utilizado pelas soluções *Ebbits*, *LinkSmart* e o*penHAB*.

3.4. Considerações Finais

Conforme se pôde observar, o desenvolvimento de soluções de *middleware* para IoT não é uma tarefa fácil que possa ser executada por qualquer tipo de usuário. É necessário que esse possua conhecimentos e domínio sobre as principais tecnologias que comumente são utilizadas na concepção de soluções para IoT, conforme apresentado na seção 3.3. Dessa forma, esse capítulo apresentou uma visão geral sobre tecnologias e desafios na IoT.

No contexto tecnológico, foram apresentadas as principais tecnologias usadas para: i) transporte de mensagens, ii) descoberta de dispositivos, iii) desenvolvimento de modelos sintáticos e semântico, iv) protocolos de comunicação sem fio e v) dispositivos inteligentes e legados. Os resultados obtidos quanto a utilização dos protocolos para transporte de mensagem, evidenciam que os protocolos REST e SOAP são os mais utilizados entre as plataformas de middleware apresentadas. Contudo, devido à sua leveza e simplicidade, os protocolos MQTT e CoAP têm sido utilizados como alternativa envio e recebimento de mensagens entre dispositivos com baixo poder de processamento e armazenamento na IoT.

Com relação à descoberta de dispositivos, das soluções que fazem uso desse tipo de funcionalidade, SDP é o protocolo mais utilizado entre as soluções apresentadas, ocorrendo devido ao fato de o SDP ser um protocolo nativo para descoberta de dispositivos que utilizam *Bluetooth* para comunicação. Para os dispositivos que fazem uso da tecnologia Wi-Fi, uma alternativa para facilitar a sua descoberta é o uso do protocolo UPnP, conforme implementado pelas soluções *Ebbits*, *LinkSmart*, *sm4all* e *task computing*. Os demais protocolos, apesar de as soluções apresentadas não fazerem uso, são importantes alternativas que podem ser utilizadas na implementação desse tipo de funcionalidade.

Plataformas que fazem uso de padrões e modelos semânticos, tais como o *LinkSmart*, *Ebbits* e *Task Computing*, utilizam-se da linguagem OWL, devido aos diferentes níveis de expressividade existentes na linguagem. No entanto, RDF é também uma excelente alternativa quanto ao desenvolvimento de modelos semânticos, porém com um nível menor de expressividade.

Com relação às tecnologias de comunicação sem fio, percebe-se que a grande maioria das soluções tentam alcançar grande parte das interfaces de comunicação sem fio, como *wifi*, *bluetooth*, *zigbee*. Contudo, é importante que novas abordagens de inclusão e gerenciamento de diferentes tecnologias de comunicação sem fio, para que as plataformas sejam melhor investigadas, o que facilitaria às atuais plataformas, de modo a adaptá-las às novas tendências de mercado.

Com relação às "Cosias", refere-se aos dispositivos comumente encontrados no dia-a-dia das pessoas, como sensores, dispositivos inteligentes (*smart TV, smart Freezer,* entre outros) e dispositivos legados, que são eletroeletrônicos antigos interligados por meio de plataformas de hardware livres como *Arduino* e *Raspberry*. Das soluções apresentadas, somente *LinkSmart*, *sm4all*, *openHAB* e *task computing* são suporte para atender a dispositivos inteligentes e/ou legados. Contudo, a maioria das soluções apresentadas fornece suporte ao gerenciamento de redes de sensores sem fio (RSSF), principais dispositivos que permitem tornar a IoT uma realidade.

No contexto de desafios, foram apresentados e discutidos doze (12) desafios a serem superados, no que tange ao desenvolvimento de soluções de *middleware* para IoT, que são: *i)* interoperabilidade, *ii)* conectividade, *iii)* escalabilidade, *iv)* esquema de nomeação, *v)* flexibilidade, *vi)* reusabilidade, *vii)* portabilidade, *viii)* usabilidade, *ix)* segurança e privacidade, *x)* gerenciamento de grande volume de dados, *xi)* gerência de contexto e *xii)* descoberta de dispositivos.

Dos desafios avaliados, é possível perceber que os principais desafios a serem superados pelas plataformas de *middleware* para IoT são atendidos pela grande maioria das plataformas, tais como, interoperabilidade, conectividade, esquemas de nomeação e portabilidade. Entretanto, desafios importantes, como usabilidade, escalabilidade e segurança e privacidade, precisam ser melhor investigados. Com relação ao gerenciamento de grandes volumes de dados, que até é um importante desafio, mas não obrigatoriamente a ser superado, somente a solução EcoDIF menciona o uso de

computação nas nuvens para armazenamento e gerenciamento de grandes volumes de dados.

Usabilidade é outro importante desafio a ser superado pelas atuais plataformas para IoT. Nesse quesito, foram avaliadas as principais interfaces gráficas utilizadas pelas soluções no sentido de permitir que usuários especialistas e não especialistas consigam manipular tais soluções e suas principais funcionalidades. Dessa forma, soluções como EcoDiF, sm4all e RestThing são as que atendem parcialmente ao requisito de usabilidade. Isso porque tais soluções fazem uso de interfaces gráficas para permitir que os usuários acessem somente aos serviços dos dispositivos, não os permitindo criarem comportamentos e/ou regras a partir dos serviços disponíveis. Já as soluções como *aspire*, *socrades*, *openHAB* e *Task Computing* são as únicas das avaliadas que fazem uso de interfaces gráficas e modelos para permitir que os usuários criem comportamentos e/ou regras ao ambiente, de acordo com as suas necessidades. Quanto às demais soluções, verifica-se que não fazem uso ou não especificam uso de interfaces gráficas, conforme literatura utilizada na avaliação.

Capítulo 4 – O Modelo de Comportamento BDM4IoT

Atualmente, a maioria das soluções para IoT são concebidas com a finalidade de serem manipuladas por usuários que possui um conhecimento técnico elevado. Entretanto, existem esforços da indústria e academia propondo soluções (*hardware* e *software*) no sentido de promover a inclusão de usuários com pouco ou quase nenhum conhecimento técnico em relação ao controle de sistemas para IoT, como em Perera et al. [2013b] e Warriach et al. [2014]. Fabricantes de dispositivos, como *Samsung*, *Philips*, *GE* e *Belkin*, já fornecem soluções de *hardware,* como *smart plugs*, *smart TV*, *sensores* e *lâmpadas inteligentes,* que permitem ao usuário interagir com o ambiente por meio de soluções específicas, mas limitando-se apenas à configuração e manipulação dos dispositivos existentes no mundo real, não permitindo ainda que esses possam criar com base em modelos, diferentes comportamentos que o ambiente poderia assumir.

Assim, este capítulo destina-se a apresentar o BDM4IoT (*Behavior Definition Model for Internet of Things Ecosystems*), um modelo conceitual criado para possibilitar que usuários possam definir os diferentes tipos de comportamentos de um ambiente para IoT, sem que, para isso, haja necessidade de adquirir elevados conhecimentos técnicos na realização de sua atividade. Dessa forma, este capítulo está organizado de acordo com o seguinte: a seção 4.1 apresenta uma introdução sobre o modelo de comportamento proposto. Em seguida, serão apresentados os objetos de dispositivo, bem como seus *layouts* e propriedades (seção 4.2). Após isso, a seção 4.3 descreve, com maior propriedade, os objetos de regras de negócio definido no modelo proposto. Depois, a seção 4.4 vai demonstrar alguns exemplos do uso de cada objeto definido. Por fim, será apresentada a linguagem de marcação BDML (seção 4.5) e as considerações finais (seção 4.6).

4.1. Introdução ao Modelo BDM4IoT

O BDM4IoT é um modelo para definição de comportamento de ecossistemas para IoT. O modelo BDM4IoT foi criado com base na premissa segundo a qual o comportamento de um ambiente pode ser definido a partir dos dispositivos eletrônicos existentes no ambiente e das ligações lógicas existentes entre eles. Portanto, a ideia principal desse modelo é a de possibilitar aos usuários definir o comportamento de diferentes ecossistemas, a partir da ligação lógica entre os dispositivos, utilizando, para tal, um conjunto de objetos, a fim de tornar possível que eles possam inferir diferentes regras ao comportamento modelado, como desvio condicional, repetição de blocos, tratamento dos dados, execução de ações, entre outros.

No desenvolvimento do modelo conceitual BDM4IoT, procuraram-se atender importantes características, as quais estão diretamente relacionadas, e, assim, permitir que o modelo possa ser manipulado tanto por usuários especialistas quanto por usuários não especialistas, quais sejam:

- **Simplicidade –** os comportamentos criados a partir do modelo BDM4IoT são simples, de fácil manipulação, compreensão e leitura, permitindo que tanto usuários especialistas quanto não especialistas consigam entender os modelos de comportamento criados sem grandes dificuldades;
- **Expressividade –** os objetos que compõe o modelo BDM4IoT possuem *layouts* simples e ícones expressivos, justamente para que os usuários compreendam a real finalidade de cada objeto do modelo, possibilitando, então, que eles possam utilizá-los corretamente e de forma intuitiva;
- **Facilidade de aprendizado –** devido ao atendimento das características de simplicidade e expressividade, o modelo BDM4IoT também possibilita que usuários especialistas e não especialistas compreendam rapidamente o modelo, criando diferentes comportamentos; e
- **Versatilidade –** o modelo BDM4IoT não se restringe apenas a modelar comportamento de ambientes específicos, mas também possibilita que comportamentos de diferentes ecossistemas sejam modelados. Essa característica se deve ao fato de o modelo possuir um conjunto de objetos que permite ao usuário criar comportamentos, independentemente do ambiente a ser modelado.

O modelo BDM4IoT é composto por duas classes distintas de objetos: *i)* Objetos de dispositivos, usados para representar os diferentes dispositivos existentes no ambiente e *ii)* Objetos de regras de negócio, responsáveis por definir as diferentes regras de negócio, que devem ser obedecidas quanto ao comportamento que o ambiente assumir.

4.2. Objetos de Dispositivos

O elemento dispositivo afigura-se como o principal objeto do modelo BDM4IoT, por meio do qual o usuário não especialista define as ações a serem executadas no ambiente. No modelo, objetos de dispositivos possuem três classificações: objetos de início de fluxo, objetos intermediários e objetos de fim de fluxo. A Figura 4.1, a seguir, apresenta os diferentes tipos de *layouts* definidos para cada tipo de objeto de dispositivo, seguidos respectivamente de seus exemplos:

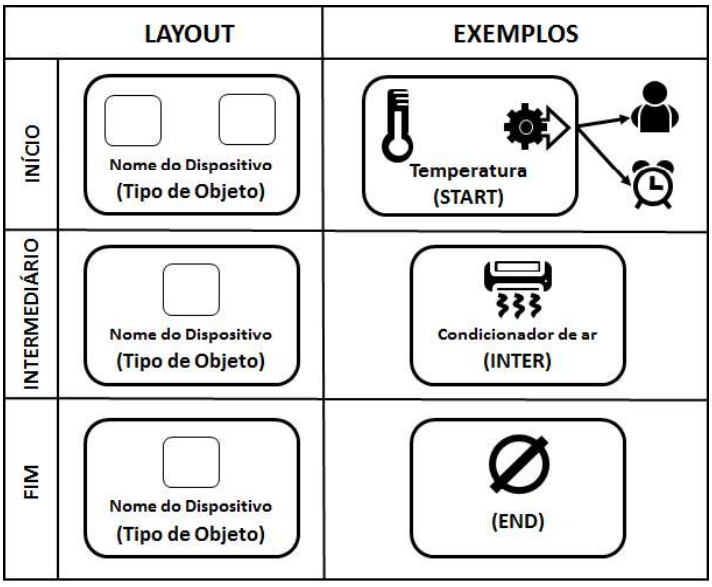


Figura 4.1. Layouts e Exemplos de Objetos de Dispositivo.

Todo objeto do tipo dispositivo é representado por um retângulo, contendo em seu centro o nome do dispositivo, o tipo de objeto e um *layout* específico para cada tipo de objeto. Objetos de início de fluxo são os dispositivos que iniciam a execução de um fluxo lógico de comportamento. Para isso, é necessário definir as condições de disparo do evento desse objeto. Um objeto de início de fluxo possui três tipos de disparo: *i)* disparo engatilhado, que é executado quando um determinado método e condição são atendidos; *ii)* disparo temporizado, executado somente quando uma condição temporal é atendida, ou seja, em uma determinada data e/ou hora; e *iii)* disparo manual, ocorrendo apenas quando o usuário manualmente dá início ao fluxo.

Objetos de início de fluxo possuem o *layout* a seguir:

i) O quadrado localizado no canto superior direito é reservado para expor o tipo de disparo utilizado pelo objeto, sendo cada tipo de disparo representado por um ícone específico, conforme pode se observar na Figura 4.17;

ii) O quadrado localizado no canto superior esquerdo é reservado para evidenciar o ícone que representa as diferentes classes de dispositivos, como sensor, televisão, condicionador de ar, entre outros; e

iii) A parte central do retângulo é reservada à exposição do nome do dispositivo, seguida da palavra reservada *START*, localizada logo abaixo do nome do dispositivo.

Os objetos intermediários são dispositivos que executam ações de acordo com a ordem de execução definida pelo fluxo. Tais objetos são dispostos entre um objeto de início de fluxo e outro objeto de fim de fluxo. Objetos intermediários possuem por padrão o disparo do tipo engatilhado, ou seja, a sua configuração requer que o usuário defina o método e as condições necessárias para a realização do disparo. Objetos intermediários possuem layout mais simples se comparado aos objetos de início de fluxo. O quadro localizado no centro do retângulo é reservado para expor o ícone que representa a classe do dispositivo e a parte central do retângulo fica reservada para expor o nome do dispositivo, seguida da palavra reservada *INTER*, localizada logo abaixo do seu nome.

Objetos de fim de fluxo são objetos que indicam o fim de um fluxo lógico, que pode ser finalizado com a execução da ação de um dispositivo ou, até mesmo, mesmo sem ação. Objetos de fim de fluxo que executam ações utilizam como padrão o disparo do tipo gatilho. Objetos de fim de fluxo possuem o mesmo layout do que os objetos de meio, diferenciados apenas pela palavra reservada *END*, localizada logo abaixo do nome do dispositivo.

É importante ressaltar que um modelo de comportamento no BDM4IoT, deve obrigatoriamente possuir um objeto de início de fluxo e um objeto de fim de fluxo. Objetos intermediários não são obrigatórios, visto que um comportamento pode ser definido em dois passos, por exemplo, ligar a televisão (início de fluxo) e desligar as luzes (fim de fluxo). A Figura 4.2, a seguir, ilustra com propriedade um exemplo de comportamento simples, utilizando apenas objetos de dispositivos:

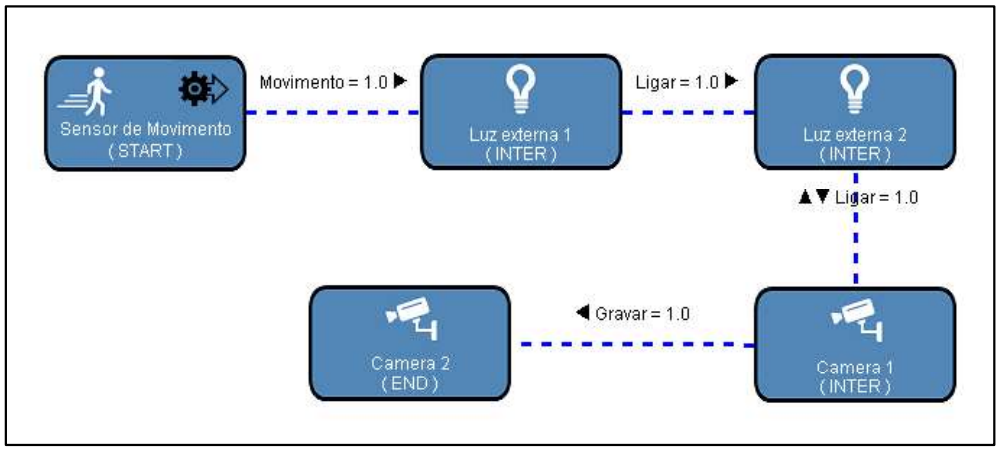


Figura 4.2. Modelo utilizando somente objetos de dispositivos.

No BDM4IoT os objetos que compõem o modelo são interligados por meio de uma linha tracejada, que indica o fluxo de ações a serem executadas. Além disso, o modelo de comportamento é também composto por setas indicativas para facilitar a leitura do modelo, bem como indicar a ordem das ações. As legendas das ligações, supramencionada, informam ao usuário as condições nas quais devem ser atendidas, para se executar a próxima ação. Por exemplo, a ação do objeto *Luz externa 1* só será executada se o sensor detectar um movimento, se, nesse caso, a função *movimento* retornar o valor 1 (verdadeiro). Após isso, as demais ações serão executadas até que um objeto de fim de fluxo seja encontrado. É importante observar que o objeto de início de fluxo utiliza o disparo do tipo gatilho para iniciar a execução do comportamento. A seção, a seguir, apresenta os objetos para definição das regras de negócio, que devem ser pontualmente obedecidas na execução do modelo.

4.3. Objetos de Regras de Negócio

Objetos de regras de negócio são criados para auxiliar o usuário no desenvolvimento das regras de negócio em questão, que deverão ser executadas quando a ação de um objeto de dispositivo, executada no ambiente, for detectada pela solução. Por exemplo, a existência de fumaça no ambiente foi detectada por seu sensor: com base nessa ação, o usuário poderá definir que tipo de comportamento pode ser executado, como o de executar a ação de um outro dispositivo presente no ambiente, enviar um e-mail, SMS (*Short Message Service*), mensagem, entre outras ações.

Objetos de regras de negócio possuem três classificações: objetos de controle de dados, objetos de controle de fluxo e objetos de controle estrutural. *Objetos para controle*

dados são responsáveis por manter o controle dos dados gerados e capturados pelos dispositivos presentes no ambiente, como exibição (*E-mail*, *SMS*, mensagem em tela ou Som) e localização de onde os dados serão salvos (arquivo de texto, nuvens ou em banco de dados). *Objetos para controle de fluxo* são responsáveis por permitir aos usuários definirem os diferentes fluxos de ações de um comportamento, como desvio, repetição, operadores e interrupção. *Objetos para controle estrutural* são utilizados para se definir a estrutura do ambiente a ser modelado, como salas, quartos, escritórios, entre outros. A Tabela 4.1, adiante, busca apresentar os objetos de regras de negócio, existentes no modelo BDM4IoT:

OBJETOS PARA CONTROLE DE DADOS	
	E-mail – objeto utilizado para enviar mensagens de e-mails após o término da execução da ação de um dispositivo.
	SMS (Short Message Service) – objeto utilizado para enviar uma mensagem SMS após o término da execução da ação de um dispositivo.
	MSG – objeto utilizado para enviar uma mensagem na tela após o término da execução da ação de um dispositivo.
	Sound – objeto utilizado para emitir um som após o término da execução da ação de um dispositivo, como, por exemplo, um som advindo de um arquivo em formato MP3 ou Mid.
	Calc – Objeto utilizado para permitir a realização de cálculo sobre os dados retornados de um objeto de dispositivo; por exemplo, o usuário deseja calcular qual o valor de energia consumido por um determinado eletroeletrônico, TV, Geladeira, Microondas, entre outros, e, com base no valor encontrado, tomar alguma ação.
	File – objeto utilizado para criar um arquivo texto com o resultado gerado pela execução da ação de um dispositivo.

	Database I/O – objetos utilizados quando se deseja salvar ou recuperar dados de um banco de dados, após o término da execução da ação de um dispositivo.
	Cloud I/O – objeto utilizado quando se deseja salvar ou recuperar dados de um servidor em nuvem, após o termino da execução da ação de um dispositivo.
OBJETOS PARA CONTROLE DE FLUXO	
	Desvio Condicional – utilizado quando se deseja incluir um fluxo condicional ao fluxo principal, permitindo que este possa seguir fluxos distintos, que serão definidos por meio do atendimento de uma condição verdadeira ou falsa.
	Execução Sequencial – Objeto utilizado quando se deseja realizar uma lista de ações de dispositivos de forma sequencial.
	Repetição Sequencial – Objeto utilizado quando se deseja repetir a execução de uma ação ou um bloco de ações de um ou mais dispositivos, mas com uma quantidade definida de vezes.
	Operador AND – objeto utilizado quando se deseja condicionar a execução de dois ou mais fluxos em um único fluxo contínuo, utilizando, para isso, o operador lógico AND.
	Operador OR – objeto utilizado quando se deseja condicionar a execução de dois ou mais fluxos em um único fluxo contínuo, utilizando, para isso, o operador lógico OR.
	*Pause* – objeto utilizado para interromper o fluxo por um período de tempo programado.

OBJETOS PARA CONTROLE ESTRUTURAL	
	Environment – objeto utilizado para representar lugares físicos do ambiente IoT, como, por exemplo, quartos, salas, cozinhas, banheiros e, até mesmo, locais mais complexos, como a linha de produção de um ambiente industrial. Environment são containers que armazenam qualquer tipo de objeto do modelo, inclusive outro objeto Environment.

Em um modelo de comportamento utilizando o BDM4IoT, não há obrigatoriedade do uso dos objetos de regra de negócio, conforme pode ser observado através da Figura 4.2. Isso possibilita que, mesmo o usuário não sabendo utilizar os objetos de regras de negócio, ainda assim ele conseguirá criar modelos de comportamento simples, utilizando apenas os objetos de dispositivos para tal. No entanto, objetos de regras de negócio permitem enriquecer o modelo possibilitando melhor controle dos dados e do fluxo de execução das ações. A fim de demonstrar o uso dos objetos de regras de negócio em um modelo de comportamento, a Figura 4.3, busca ilustrar um modelo de comportamento, combinando objetos de dispositivo e de regras de regras de negócio:

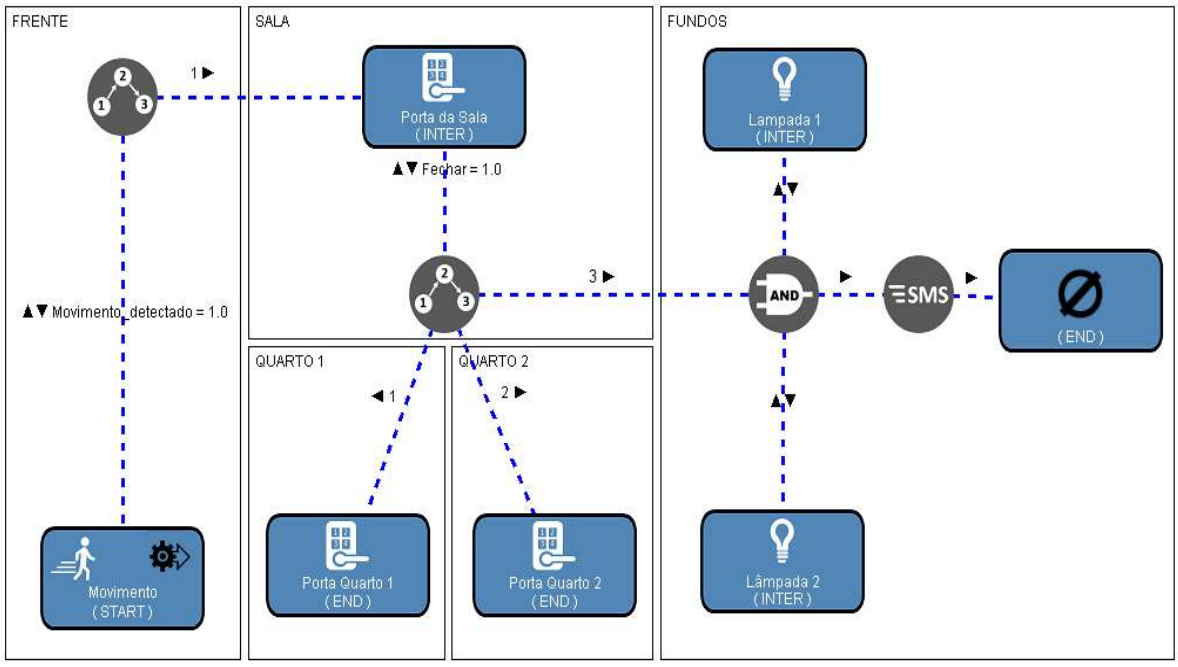


Figura 4.3. Combinação de objetos de dispositivos e de regras de negócio.

Pode-se observar que a Figura 4.3 apresenta um modelo de comportamento de segurança de uma casa. O comportamento é iniciado quando o sensor, localizado na parte da frente da casa, detecta um algum movimento. Após isso, é executado um conjunto de ações separadas, utilizando um objeto para execução sequencial. Cabe ressaltar que o primeiro objeto de execução sequencial possui apenas uma sequência de ações. A primeira ação a ser executa é trancar a porta da frente da casa; em seguida, outro objeto de execução sequencial é iniciado, possuindo três sequências de ações: a primeira ação executada é fechar a porta do quarto 1; a segunda, o fechamento da porta do quarto 2; e, então, a terceira ação é direcionada para um objeto *AND* para garantir que a próxima ação a ser enviar só será executada se, e somente se, as lâmpadas 1 e 2 forem ligadas. Após isso, um SMS é enviado e processo finalizado sem que qualquer execução seja realizada.

4.4. Utilizando o Modelo BDM4IoT

Para melhor esclarecer a utilização dos objetos do modelo BDM4IoT, esta seção destina-se a apresentar e explicar o uso correto dos objetos que compõem o modelo, bem como apresentar os parâmetros necessários para sua utilização. Conforme citado anteriormente, o modelo BDM4IoT é composto por duas classes de objetos: objetos de dispositivo e objetos de regras de negócio. A seguir, a subseção 4.4.1 apresenta as regras e principais parâmetros para utilização de objeto do tipo dispositivo.

4.4.1. Utilizando Objetos de Dispositivos

A utilização de objetos de dispositivo é simples, necessitando apenas de que o usuário defina o tipo de objeto e os parâmetros necessários para utilização. Primeiramente, o usuário deverá informar o tipo de dispositivo (“inicio de fluxo”, “intermediário” e “fim de fluxo”), e, em seguida, o usuário deverá configurar o tipo de disparo que irá inicializar o fluxo de execução do modelo (“engatilhado”, “temporizado” e “manual”). Além disso, é importante ressaltar que objetos intermediários e de fim de fluxo possuem, por padrão, o disparo do tipo “engatilhado”.

Para disparo do tipo “engatilhado”, o usuário deverá informar qual serviço deve ser executado e em quais condições deverão ser atendidas, para que o fluxo de execução prossiga para próxima ação. Para disparo do tipo “temporizado”, o usuário deve informar o

serviço a ser executado, além da data e hora para sua execução. Para disparo do tipo "manual", o usuário deverá informar somente o serviço a executar, uma vez que os disparos manuais são acionados somente com a intervenção do usuário. A Figura 4.4, abaixo, apresenta três exemplos, utilizando os diferentes tipos de objeto de dispositivo e disparos existentes:

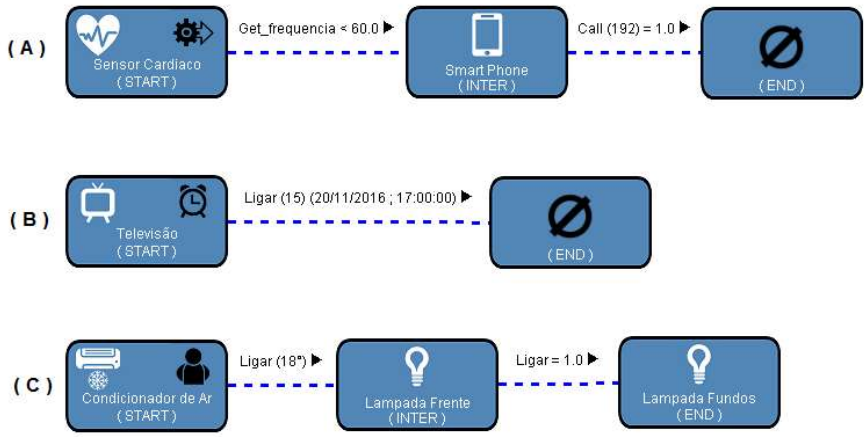


Figura 4.4. Utilizando objetos de dispositivos, seus diferentes tipos e gatilhos.

No BDM4IoT, um modelo de comportamento deve sempre ser iniciado e finalizado utilizando um objeto de dispositivo. No exemplo (A) da Figura 4.4, o modelo de comportamento utiliza um disparo do tipo engatilhado, ou seja, quando a execução do método "Get_Frequencia" retornar um valor menor que 60. Caso isso ocorra, a próxima ação a ser executada é uma chamada para o serviço de emergência (192), assim o fluxo finaliza sem nenhuma outra ação.

No exemplo (B) da Figura 4.4, o modelo de comportamento utiliza um disparo do tipo temporizado, ou seja, o método "Ligar" da televisão só será executado em uma data (20/11/2016) e hora (17:00:00) prevista. Nesse caso, o fluxo é finalizado sem execução de ação.

No exemplo (C) da Figura 4.4, o modelo de comportamento utiliza um disparo do tipo manual, ou seja, quando o método "Ligar" do condicionador de ar for remotamente acionado pelo usuário. O próximo evento do fluxo a ser executado é o ligar lâmpada da frente. E, por fim, o fluxo é finalizado somente após a execução de um método do objeto "Lâmpada Fundo".

4.4.2. Utilizando Objetos de Regras de Negócios

Diferente dos objetos de dispositivo, que podem ser utilizados de forma intuitiva, objetos de regras de negócio requer que o usuário adquira conhecimento prévio sobre a utilização correta de cada objeto e quais os parâmetros necessários para configuração dos mesmos. Além disso, alguns objetos de regras de negócios requerem dos usuários conhecimentos técnicos para sua utilização, principalmente com relação à utilização dos objetos para controle de fluxo, que possuem parâmetros pouco convencionais aos usuários sem qualquer conhecimento técnico. No entanto, objetos para controle de dados e controle estrutural são fáceis de serem utilizados, requerendo apenas que os usuários saibam informar corretamente os parâmetros exigidos para sua configuração.

É importante destacar também que objetos de regras de negócio podem ser utilizados em combinação com os diversos objetos do modelo BDM4IoT, tanto com objetos de dispositivo quanto objetos de regras de negócio. No entanto, objetos de regras de negócio não podem ser utilizados para iniciar e finalizar um fluxo de execução, mas somente como objetos intermediários, conforme pode se observar na Figura 4.5. A seguir, são apresentados os objetos de regras de negócios para controle de dados (Tabela 4.1).

a. Objetos de regras de negócio para controle de dados

Objetos de regras de negócio para controle de dados são os principais objetos do modelo BDM4IoT responsáveis por controlar os dados, como armazenamento e exibição, utilizando, para isso, diferentes meios, como e-mail, SMS, mensagens de tela e, até mesmo, mensagens sonoras, além de também possibilitar que os dados sejam armazenados localmente ou nas nuvens. A Figura 4.5 apresenta exemplos do uso dos objetos de regras de negócio para controle de dados:

A Figura 4.5(A) apresenta um modelo de comportamento que é iniciado quando o sensor detecta um movimento. Após isso, um *e-mail* é enviado e o processo finalizado sem qualquer outra ação. Para utilização do objeto e-mail, é necessário que o usuário informe alguns parâmetros, como: remetente, destinatário, assunto, entre outros.

A Figura 4.5(B) apresenta um modelo de comportamento que é iniciado após o sensor elétrico realizar a leitura da tensão da rede elétrica. Após isso, um cálculo é realizado sob os dados obtidos do sensor e o processo é finalizado sem que qualquer ação seja executada.

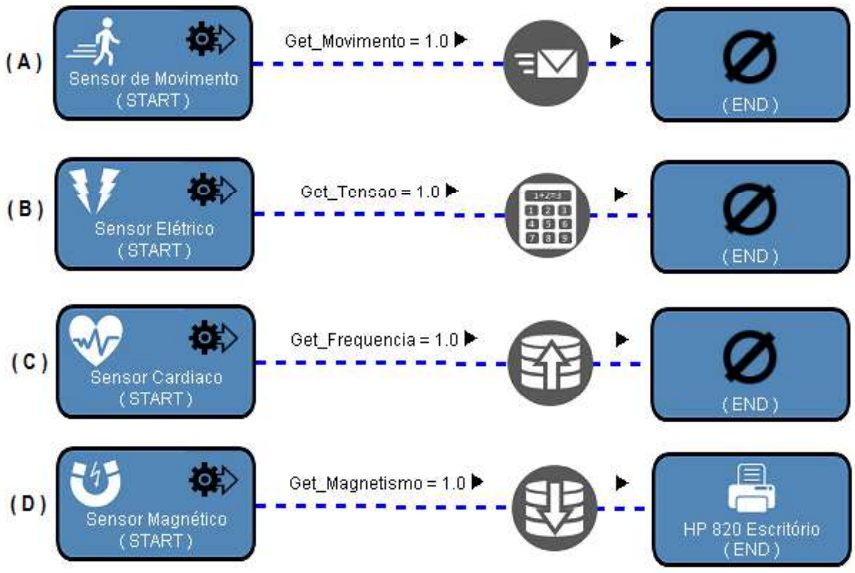


Figura 4.5. Utilizando objetos de regras de negócio para controle de dados.

A Figura 4.5(C) apresenta um modelo de comportamento que é iniciado após cada leitura realizada pelo sensor cardíaco. Depois disso, os dados gerados pelo sensor são enviados para um objeto *database* do tipo *input* para serem armazenados. É importante destacar que os dados são recebidos e armazenados em formato JSON (*Java Script Object Notation*).

A Figura 4.5(D) apresenta um modelo de comportamento iniciado após o sensor identificar e realizar a leitura de um campo magnético. Em função dessa leitura, é realizada uma consulta em uma base de dados utilizando, para tal, o objeto *database* do tipo *output*. Após a consulta, o fluxo de execução é finalizado com a impressão desses dados. É importante destacar que, para realização da busca dos dados, o usuário deverá informar a consulta a ser realizada, por meio do parâmetro *query*.

Os exemplos apresentados pela Figura 4.5 demonstram a utilização de quatro objeto de regras de negócio, os demais objetos seguem a mesma lógica de utilização, bastando apenas que o usuário informe corretamente os parâmetros de cada objeto, de acordo com as suas necessidades.

b. Objetos de regras de negócio para controle de fluxo

Objetos de regra de negócio para controle de fluxo são utilizados quando o usuário deseja inferir regras de negócio que venham a alterar o fluxo de execução padrão de um comportamento. No modelo BDM4IoT, objetos para controle de fluxo são: desvio condicional, repetição sequencial de fluxo, operador lógico *AND* e *OR* e *Pause*. Os

exemplos a seguir apresentam um modo de utilização desses objetos. A Figura 4.6 demonstra a utilização do objeto desvio condicional. Para facilitar a identificação dos objetos, uma seta em vermelho faz referência ao objeto em destaque, devido a alguns exemplos utilizarem mais de um objeto de regras de negócio, conforme o seguinte:

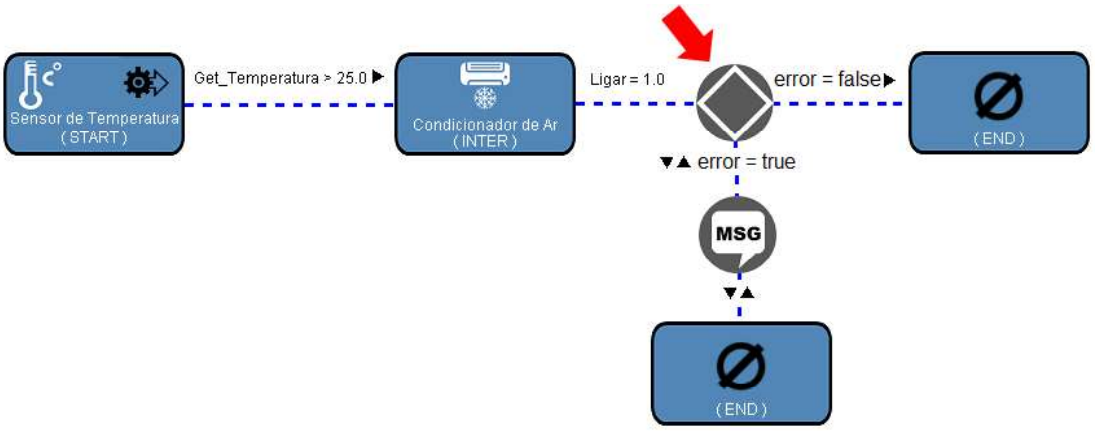


Figura 4.6. Usando o objeto desvio condicional.

O exemplo da Figura 4.6 é iniciado quando o sensor detecta uma temperatura maior que 25° graus. O condicionador é alterado para o estado de ligado e, em seguida, é verificado se houve algum erro ao tentar trocar o estado do condicionador de ar para ligado. Caso não haja erro, o fluxo será finalizado sem qualquer outra ação. Caso contrário, será apresentado uma mensagem para o usuário informando o erro ocorrido. A utilização do objeto desvio condicional requer que o usuário informe apenas os parâmetros de condição verdadeira e falsa e os caminhos a serem seguidos, caso as condições sejam atendidas. A Figura 4.7 apresenta um exemplo utilizando o objeto execução sequencial:

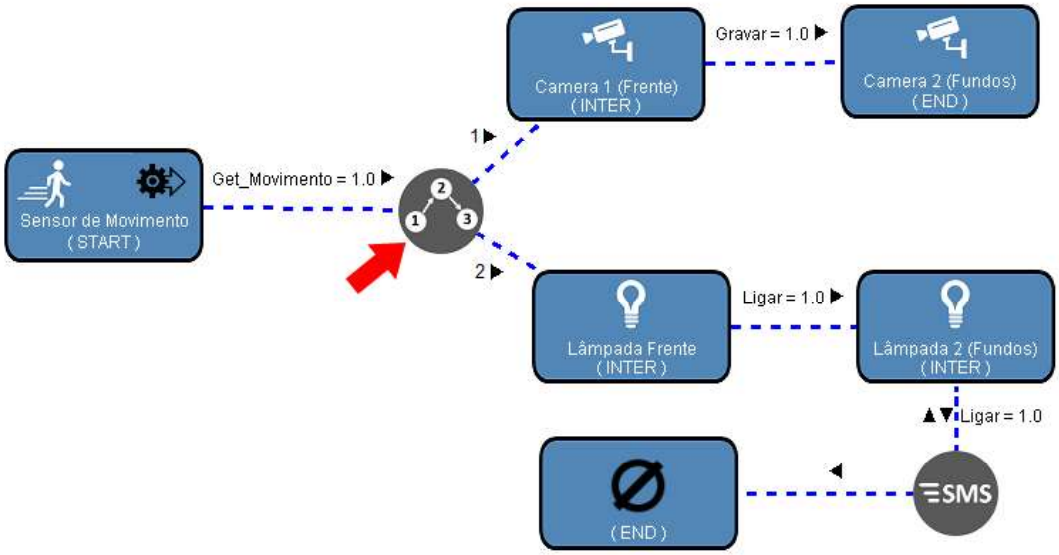


Figura 4.7. Usando o objeto execução sequencial.

O exemplo da Figura 4.7 é iniciado quando o sensor detecta um movimento no ambiente. Com auxílio do objeto de execução sequencial, foram criados dois fluxos de execuções. No primeiro fluxo, as câmeras 1 e 2 são colocadas em modo de gravação e o fluxo 1 finalizado. No segundo fluxo, as lâmpadas 1 e 2 são acionadas e um *SMS* é enviado, um comportamento utilizando uma execução sequencial só é finalizado quando todos os comportamentos forem executados. A Figura 4.8 apresenta o uso do objeto de repetição sequencial:

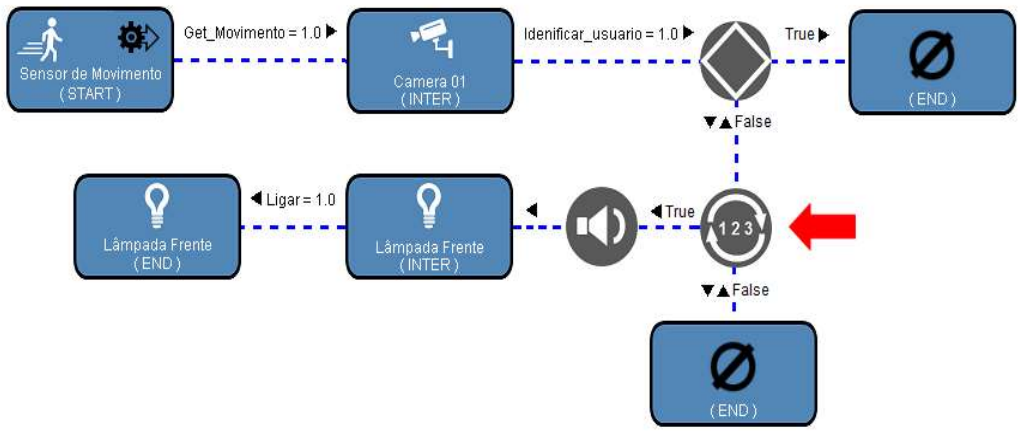


Figura 4.8. Usando o objeto Repetição Sequencial.

O exemplo da Figura 4.8 é iniciado após o sensor detectar um movimento no ambiente. Partindo da premissa de que a câmera utiliza um algoritmo para reconhecimento facial, sendo que, após a câmera 01 verificar se a pessoa detectada é um usuário autorizado para estar no ambiente, o fluxo de execução é finalizado; do contrário, com o uso de um objeto repetição sequencial, um alarme sonoro é disparado N vezes seguidas e a lâmpada da frente ira ficar ligando e desligando enquanto a repetição estiver em execução. Após isso, a execução do comportamento é finalizada. Para utilização do objeto repetição sequencial, é necessário informar o número de repetições e os fluxos verdadeiros e falsos que deverão ser seguidos quando a condição for ou não atendida. A Figura 4.9 apresenta o uso do objeto operador lógico *AND*:

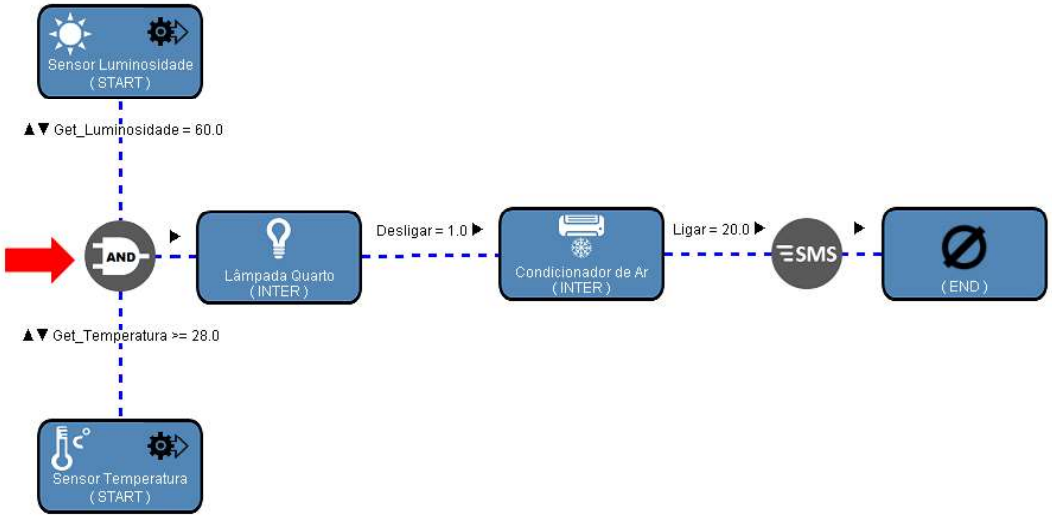


Figura 4.9. Usando o objeto operador AND

O exemplo da Figura 4.9 é iniciado quando duas ações são executadas e as condições atendidas. Para condicionar o fluxo a atender as condições de execução dos objetos de dispositivo sensor de luminosidade e sensor de temperatura, foi utilizado o operador lógico *AND.* O fluxo principal só será executado se as condições de execução de ambas as ações forem atendidas. Caso positivo, o fluxo inicia desligando a lâmpada do quarto, ligando o condicionador de ar e enviando um *SMS.* Após isso, o fluxo de execução é finalizado. Um objeto AND pode receber N condições, ou seja, o comportamento só será executado se todas as condições forem atendidas. A Figura 4.10 apresenta o uso do objeto operador lógico *OR*:

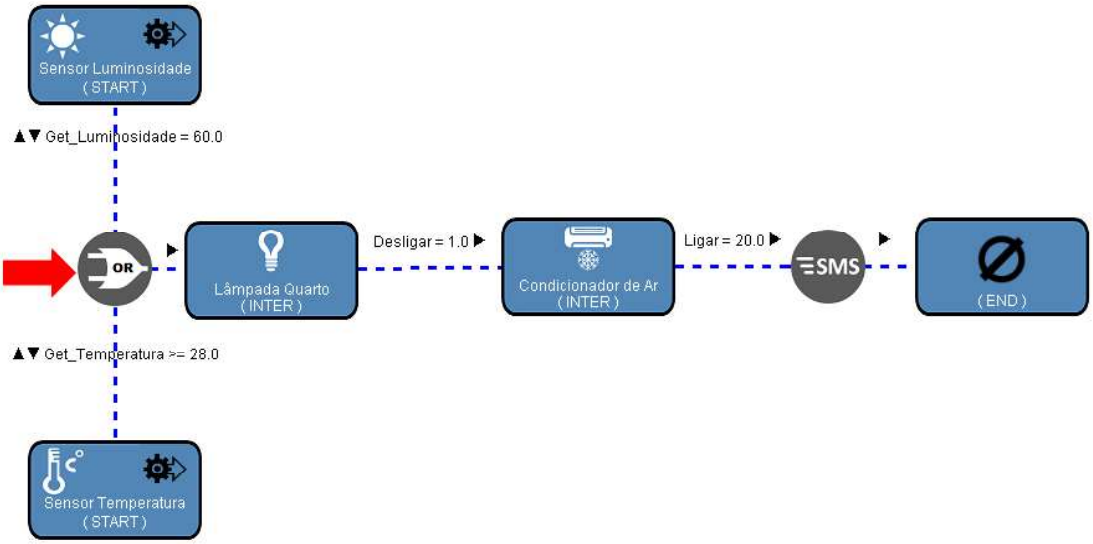


Figura 4.10. Usando o objeto operador OR.

O exemplo da Figura 4.10 é idêntico ao exemplo apresentado pela Figura 4.9, mas utilizando o objeto operador lógico *OR*, ou seja, o fluxo de execução só será iniciado quando umas das condições forem atendidas. A Figura 4.11 apresenta o uso do objeto *pause*:

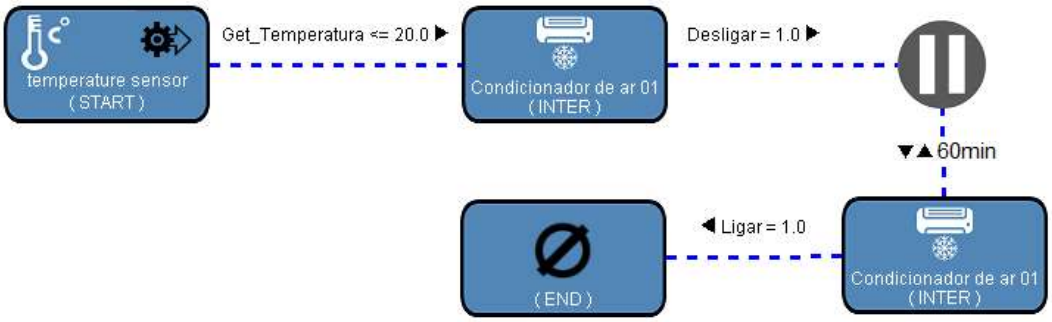


Figura 4.11. Usando o objeto pause.

O exemplo da Figura 4.11 é iniciado quando o sensor detecta uma temperatura menor ou igual a 20° graus. Então, o condicionador de ar 01 é desligado e o processo é interrompido por 60 min. Após o término desse tempo, o condicionador de ar 01 é ligado e o processo finalizado. Para utilização do objeto pause, o usuário deverá informar somente o tempo que o fluxo de execução ficará interrompido.

c. Objeto de regra de negócio para controle estrutural

O modelo BDM4IoT possui apenas um objeto para definição e controle estrutural, o objeto ENV (do inglês, *Environment*), responsável por permitir que os usuários possam definir estruturalmente o ambiente que se deseja monitorar. Com o ENV, o usuário pode criar diferentes áreas do ambiente, como salas, quartos, banheiros, quintal, entre outros, permitindo assim definir o exato local onde um dispositivo se encontra no ambiente. O objeto ENV é fácil de utilizar, exigindo apenas que o usuário informe o nome da área a ser criada. Propriedades, como tamanho altura e largura do objeto, são definidas por meio da ferramenta Êxodo GUI. A Figura 4.12 apresenta um exemplo de modelo de comportamento utilizando o objeto ENV:

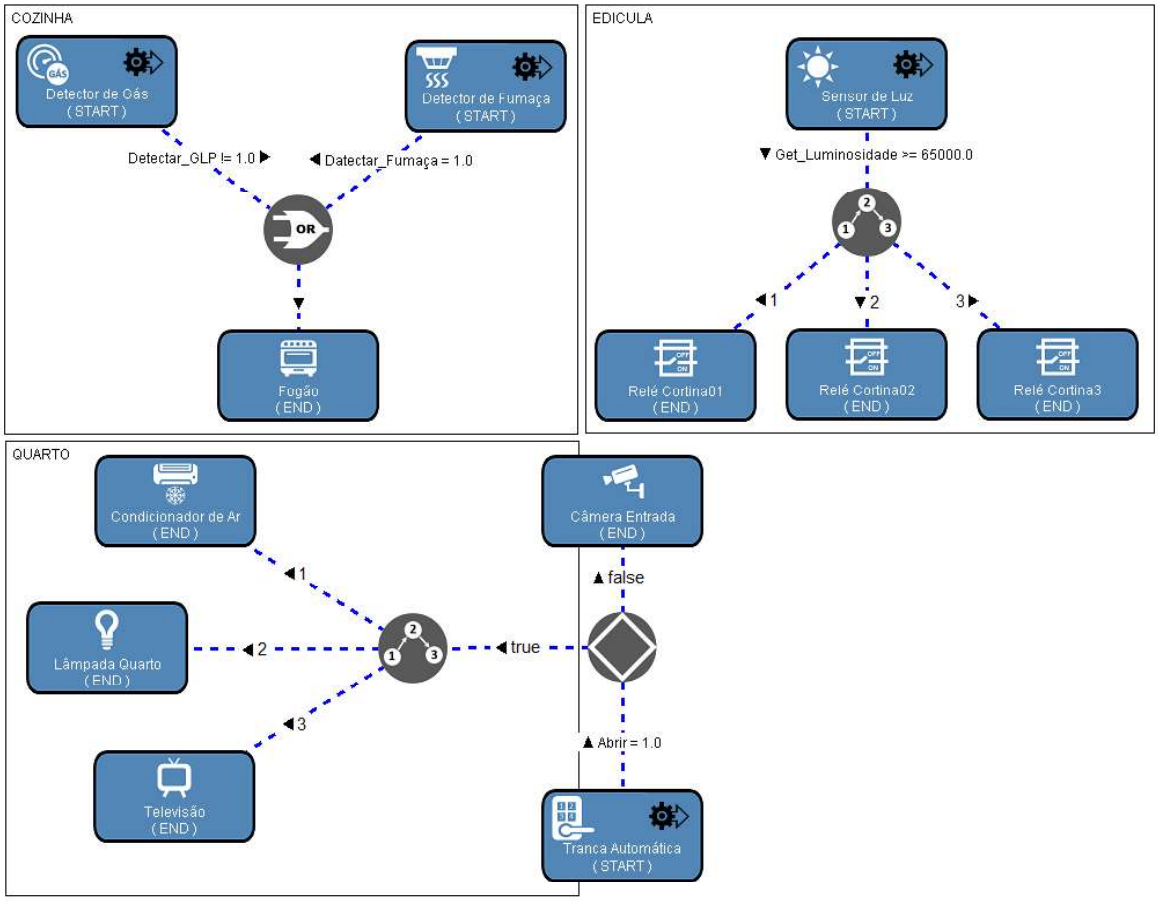


Figura 4.12. Exemplo utilizando o objeto ENV.

O comportamento modelado para cozinha é responsável por interromper o fornecimento de gás do fogão, caso seja detectado algum vazamento de GLP ou fumaça no ambiente. Nesse comportamento, foi utilizado o objeto *OR* para possibilitar que o comportamento seja iniciado utilizando mais de um objeto para início de fluxo.

O comportamento modelado para a edícula é responsável por fechar as cortinas automaticamente, caso o sensor de luminosidade identifique uma intensidade muito alta de luz solar. Se for identificado, os relés responsáveis por abrir e fechar as cortinas são acionados.

O comportamento modelado para o quarto é responsável por realizar o controle de acesso ao ambiente. Esse é iniciado quando um usuário digita a senha para acesso ao ambiente em uma fechadura eletrônica. Assim que esse procedimento é realizado, imediatamente é verificado se a senha informada está correta. Caso não esteja, a câmera inicia a filmagem do usuário que está tentando acesso ao ambiente e o comportamento é finalizado. Caso a senha digitada esteja correta, a porta é aberta e mais três ações são

realizadas: o condicionador de ar é ligado, a lâmpada do quarto acionada e a televisão ligada no canal preferido do usuário.

Apesar da não obrigatoriedade do uso do objeto ENV, é importante observar que sua utilização é imprescindível para tornar o modelo mais claro. E, permitindo posicionar os objetos de dispositivos utilizados no modelo de comportamento, no local exatamente onde esses objetos se encontram, facilita a sua localização, caso seja necessário realizar manutenções no dispositivo.

Destaca-se que os exemplos apresentados são todos destinados a atender a ambientes domésticos. Isso se deve ao fato do modelo inicialmente possuir um conjunto de objetos para atender a ambientes domésticos, como televisão, câmeras, fogão, lâmpadas, fechaduras eletrônicas, sensores de movimento, temperatura, fumaça, entre outros. No entanto, o modelo BDM4IoT foi projetado para permitir que dispositivos para atender a diferentes ambientes sejam adicionados ao modelo. Por esse motivo, o modelo permite a inclusão de novos objetos de regras de negócio. O capítulo a seguir apresenta a linguagem de marcação BDML, criada para permitir que o modelo seja estendido e adaptado, de acordo com as necessidades dos usuários e do ambiente:

4.5. Considerações Finais

Esse capítulo apresentou o BDM4IoT (*Behavior Definition Model For Internet of Things Ecosystems*), um modelo conceitual projetado para possibilitar a inclusão dos usuários não especialistas no processo criativo e gerencial de ambientes para IoT. O modelo BDM4IoT, por meio de um conjunto de objetos, possibilita que os usuários modelem o comportamento de seus próprios ambientes para IoT, sem a necessidade de adquirir conhecimentos profundos sobre as diversas tecnologias atualmente utilizadas para habilitar o paradigma da IoT.

O BDM4IoT é um modelo composto por um conjunto de objetos classificados, conforme se segue: i) Objetos de dispositivos, criados para representar virtualmente os dispositivos existentes no mundo real, além de suas propriedades e serviços disponíveis e ii) objetos de regras de negócio, utilizados no modelo para permitir a inclusão de regras de negócio, que deverão refletir diretamente sobre os dados e/ou sobre o fluxo de execução. Objetos de regras de negócio possuem a seguinte classificação: a) objetos para controle de

dados, responsáveis por salvar, restaurar e exibir os dados de acordo com as necessidades do usuário; b) objetos para controle de fluxo, responsáveis por inferir regras ao fluxo de execução, como desvios condicionais, repetições, condições lógicas de execução, entre outros; e c) objetos para controle estrutural, responsáveis por representar os espações físicos existentes no ecossistema, como salas, quartos, banheiros, cozinhas, entre outros.

Capítulo 5 – BDML (*Behavior Definition Markup Language*)

Para implementação do modelo conceitual BDM4IoT, inclusive com vista à possibilidade de extensão e modificação do modelo (de acordo com as necessidades do usuário e do ambiente), foi criada uma linguagem de marcação em XML, denominada como Linguagem de Marcação para Definição de Comportamento ou, simplesmente, BDML (do inglês, *Behavior Definition Markup Language*). A BDML é composta por todos os objetos que compõem o modelo BDM4IoT, assim como as regras necessárias para validação dos modelos de comportamento em formato BDML. Os principais objetivos da linguagem BDML são: i) armazenar as regras de validação do modelo; ii) manter todos os objetos existentes no modelo BDM4IoT, bem como suas características e propriedades; iii) manter as informações de um modelo salvos em formato BDML, permitindo sua portabilidade a qualquer aplicação que faça uso desse padrão; iv) armazenar as ligações lógicas que compõem os comportamentos do modelo; e v) realizar a execução dos comportamentos de acordo com dados e ligações logicas existentes no modelo em formato BDML.

Este capítulo tem por objetivo apresentar a estrutura da linguagem de marcação BDML e os principais elementos que compõem essa linguagem. Primeiramente, serão apresentados os três principais elementos que compõem a estrutura principal da linguagem e, em seguida, os demais elementos serão apresentados.

5.1. A Estrutura Principal da Linguagem BDML

A estrutura da BDML é composta por três elementos principais, sendo o elemento <BDML> a estrutura raiz do documento, no qual se encontram todas as informações, representadas hierarquicamente pelos demais elementos que compõem a estrutura do documento BDML, conforme apresentado pela Figura 5.1.

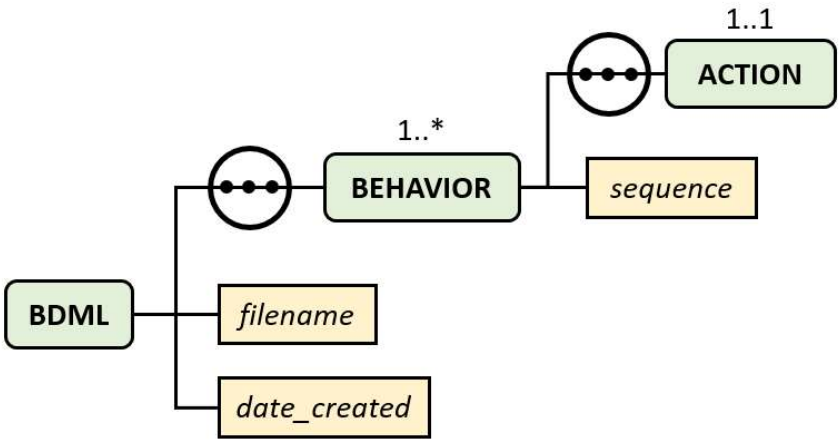


Figura 5.1. Estrutura principal do modelo BDML.

A estrutura do elemento <BDML> é composta pelos atributos *filename*, nome do arquivo BDML e *date_created*, data de criação do documento BDML. O elemento <BDML> é composto por um único elemento, o <BEHAVIOR>, responsável por manter todos os comportamentos modelados. Esse elemento é composto pelo atributo "sequence", que controla a ordem dos comportamentos modelados, e pelo elemento <ACTION>, que são as ações que deverão ser executadas no fluxo de execução do modelo de comportamento, conforme apresentado pelo exemplo a seguir:

```
<BDML filename="Modelo01" date_created="01/06/2017">
  <BEHAVIOR sequence=1>
    <ACTION> . . . </ACTION>
  </BEHAVIOR>
  <BEHAVIOR sequence=2>
    <ACTION> . . . </ACTION>
  </BEHAVIOR>
  <BEHAVIOR sequence=3>
    <ACTION> . . . </ACTION>
  </BEHAVIOR>
</BDML>
```

Na estrutura da BDML, o elemento <BEHAVIOR> pode aparecer N vezes na estrutura de um documento no formato BDML, devido ao fato de que um arquivo BDML pode conter mais de um comportamento em sua estrutura. No entanto, o elemento <BEHAVIOR> pode conter somente um elemento <ACTION>, onde são encontrados todos os objetos e relações que compõem um comportamento. A seguir, a estrutura do elemento <ACTION> será apresentada.

5.2 O Elemento <ACTION>

O <ACTION> é o principal elemento da estrutura da linguagem BDML, pois é nesse elemento que são mantidas todas as principais informações sobre o modelo de comportamento, como, por exemplo, os objetos e relações e as regras que compõem um comportamento. A Figura 5.2 apresenta os elementos que compõem a estrutura do elemento <ACTION>:

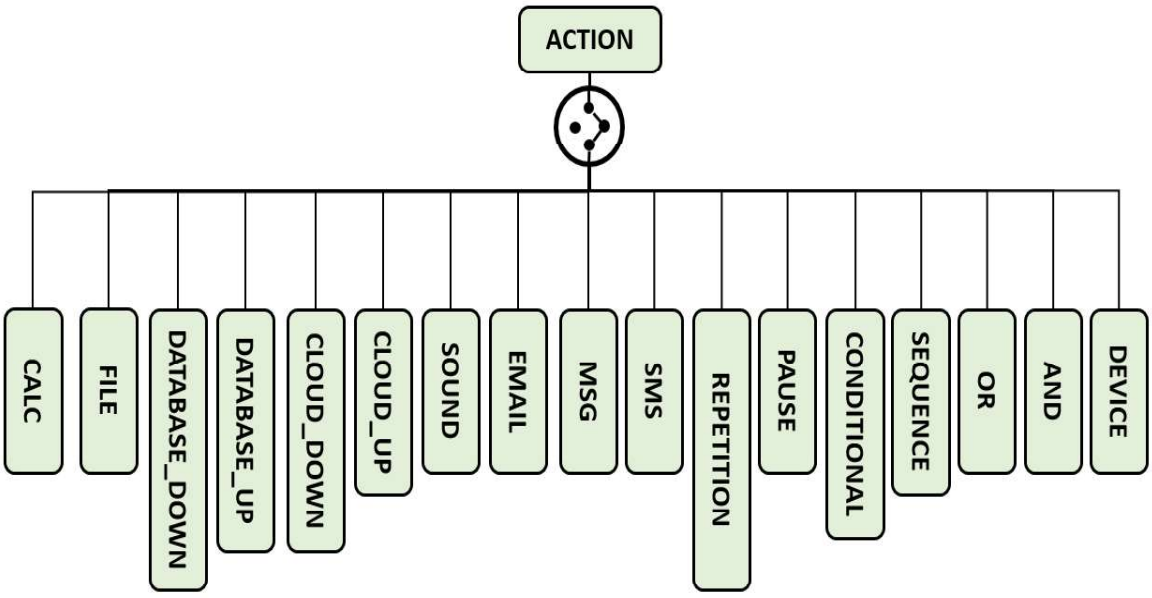


Figura 5.2. Estrutura do elemento <ACTION>.

Conforme se pode observar, o elemento <ACTION> é composto por 17 elementos, que são os objetos de dispositivo, representado pelo elemento <DEVICES>, e objetos de regras de negócio, representado pelos demais elementos. É importante destacar que, em um elemento <ACTION>, deve conter pelo menos um elemento de quaisquer dos tipos apresentados (Figura 5.2) em sua estrutura. Caso contrário, ocorrerá um erro de validação na estrutura do elemento <ACTION>. A seguir, serão apresentados os elementos que compõem o elemento <ACTION>, na ordem da direita para esquerda, conforme Figura 5.2.

5.2.1. O Elemento <DEVICE>

O elemento <DEVICE>, apresentado pela Figura 5.3, é o primeiro elemento pertencente à estrutura do <ACTION>, sua principal função no modelo é possibilitar que dispositivos do mundo físico possam ser representados virtualmente, capturar ações executadas pelos

dispositivos no mundo real e serem monitorados e acessados remotamente. A estrutura do elemento <DEVICE> é composta pelos atributos *ID*, campo que identifica um dispositivo no modelo como único, *name*, nome do dispositivo atribuído pelo usuário, *model*, o tipo de modelo do dispositivo (celular, tablet, Datashow, condicionador de ar, entre outros), *manufactured*, fabricante do dispositivo, *URI*, link de acesso aos dispositivos e serviços por ele providos, e os atributos de coordenadas gráficas *width*, *height*, *posX* e *posY*, conforme o seguinte:

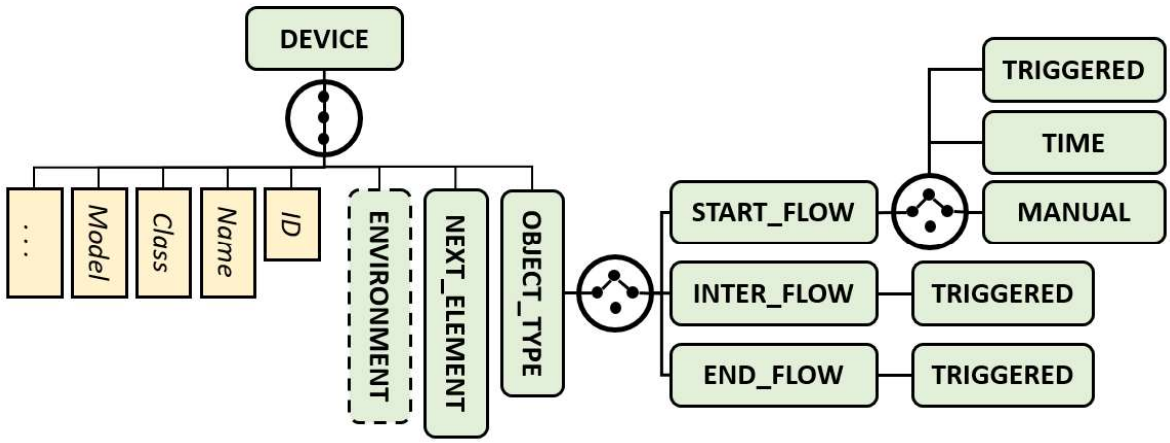


Figura 5.3. Estrutura do elemento <DEVICE>.

O elemento <DEVICE> é composto por vários elementos. O elemento <OBJECT_TYPE> define o tipo do objeto de dispositivo, que são: i) objeto de início de fluxo, definido pelo elemento <START_FLOW>; ii) objeto intermediário, definido pelo elemento <INTER_FLOW>; e iii) objeto de fim de fluxo, definido pelo elemento <END_FLOW>. Conforme apresentado no capítulo 4, para possibilitar que um evento de um dispositivo do mundo seja monitorado, é necessário definir o método adequado por meio do qual deverá ser monitorado e quais parâmetros serem atendidos para a inicialização do fluxo de execução do modelo de comportamento. Dispositivos de início de fluxo <START_FLOW> podem ser configurados com três tipos diferentes de disparo: o disparo engatilhado, temporizado e manual, representados, respectivamente, pelos elementos <TRIGGERED>, <TIMER> e <MANUAL>. É importante destacar que dispositivos intermediários e de fim de fluxo são, por padrão, configurados com o disparo do tipo <TRIGGERED>.

O elemento <TRIGGERED> é um gatilho utilizado para analisar o resultado da ação de um dispositivo e verificar se o fluxo deve iniciar, continuar ou interromper sua execução. A estrutura do elemento <TRIGGERED> é composta pelos seguintes atributos: “method”, método que deverá ser monitorada e analisada cada execução; “value”, valor para condição de disparo; “minvalue”, valor mínimo para condição de disparo; “maxvalue”, valor máximo para condição de disparo; e “condiction”, que são as devidas condições para ativação do gatilho (maior, menor, maior igual, menor igual, diferente e valores entre).

O elemento <TIMER> é um gatilho que foi disparado em uma data e/ou hora determinada pelo usuário. A estrutura do elemento <TIMER> é composta dos seguintes atributos: “method”, método a ser disparado na data e/ou hora determinada; “data”, data de disparo do evento; “time”, hora de disparo do evento; e “repeat”, parâmetro que define se o evento irá ou não se repetir. Cabe ressaltar que, para uso desse parâmetro, é importante que não seja informada a data no seu devido campo. Dessa forma, o evento será disparado todos os dias, no mesmo horário.

O elemento <MANUAL> é um evento que será disparado manualmente pelo usuário por meio de uma interface, um computador, tablet, celular ou qualquer outro dispositivo, para entrada de dados. A estrutura do elemento <MANUAL> é composta somente pelo atributo “method”, que significa o método a ser disparado manualmente pelo usuário.

O <NEXT_ELEMENT> é outro importante elemento pertencente à estrutura <DEVICE>, que é também comum a todos os demais elementos pertencentes ao <ACTION>. O principal objetivo do <NEXT_ELEMENT> é informar o próximo elemento do fluxo de execução a ser visitado. Por fim, o último elemento que compõe a estrutura do elemento <DEVICE> é o <ENVIRONMENT>, o qual é responsável por informar o local físico onde o objeto se encontra (sala, quarto, cozinha, banheiro, entre outros). A sua estrutura é composta pelos seguintes atributos: “ID”, número de identificação do ambiente e/ou localidade; “name”, nome do ambiente e/ou localidade; e pelos atributos de coordenadas gráficas “height”, “width”, “posX” e “posY”. Eis, a seguir, uma apresentação de uso do elemento <DEVICE>, como um exemplo:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
```

```
<ACTION>
<DEVICE id=1 name='STEMP1' class='sensor' width=30
      Height=10 posX=150 posY=400>
 <OBJECT_TYPE>
  <START_FLOW>
   <TRIGGERED method='get_temp' value=20 minvalue=16
          Maxvalue=30 condiction=5/>
  </START_FLOW>
 </OBJECT_TYPE>
 <ENVIRONMENT id=2 name='SALA' width=200 height=200 posX=2
          PosY=600/>
 <NEXT_ELEMENT>
  <DEVICE id=3 name='AC' class='Ar-Condicionado' width=30
        Height=10 posX=190 posY=400>
   <OBJECT_TYPE>
    <END_FLOW>
     <TRIGGERED method='desligar' value=1 minvalue=0
            Maxvalue=1 condiction=0/>
    </END_FLOW>
   </OBJECT_TYPE>
   <ENVIRONMENT id=2 name='SALA' width=200 height=900
           PosX=2 posY=2/>
  </DEVICE>
 </NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

No exemplo supracitado, referindo ao uso do elemento <DEVICE>, foi criado um comportamento onde um sensor, localizado na sala, ao detectar que a temperatura do ambiente esteja menor ou igual a 20°, o ar condicionado é desligado e o comportamento finalizado. Cabe uma importante observação: o sensor é um objeto do tipo <START_FLOW>, que utiliza o disparo do tipo <TRIGGERED> para iniciar o fluxo de execução do modelo de comportamento.

5.2.2. O Elemento <AND>

O elemento <AND>, apresentado pela Figura 5.4, é o segundo elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é inferir uma condição lógica AND ao fluxo de execução. Basta imaginar que o usuário queira condicionar a execução do fluxo à execução de duas ou mais ações; assim, o fluxo só será executado se, e somente se, todas as ações forem executadas com sucesso, consoante o seguinte:

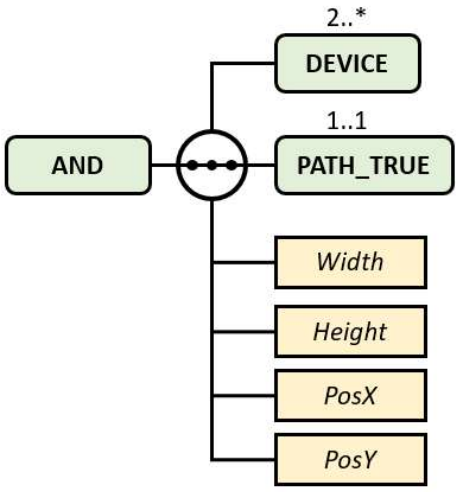


Figura 5.4. Estrutura do elemento <AND>.

A estrutura do elemento <AND> é composta somente pelos atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. Os elementos que o compõem são: <DEVICE>, dispositivos que executaram as ações que deverão ser comparadas (um elemento <AND> pode conter em sua estrutura dois (2) ou mais elementos do tipo <DEVICE>; e o <PATH_TRUE>, elemento do fluxo de execução a ser visitado, após condição AND ser atendida. A seguir, é apresentado um exemplo de uso do elemento <AND>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
 <AND>
   <DEVICE id=1 name='STEMP1' class='sensor'>
   <OBJECT_TYPE>
    <START_FLOW>
          <TRIGGERED method='get_temp' value=20 minvalue=16
          Maxvalue=30 condiction=5/>
    </START_FLOW>
   </OBJECT_TYPE>
   </DEVICE>
        <DEVICE id=3 name='AC' class='Ar-Condicionado'>
        <OBJECT_TYPE>
         <START_FLOW>
               <TRIGGERED method='ligado' value=1 condiction=0/>
         </START_FLOW>
        </OBJECT_TYPE>
        </DEVICE>
        <PATH_TRUE>
               <DEVICE id=3 name='AC' class='Ar-Condicionado'>
               <OBJECT_TYPE>
```

```
                    <START_FLOW>
                        <TRIGGERED method='desligado' value=1
                        Condiction=0/>
                    </START_FLOW>
                    </OBJECT_TYPE>
                    </DEVICE>
            </PATH_TRUE>
    </AND>
    </ACTION>
    </BEHAVIOR>
    </BDML>
```

Como exemplo de apresentação do comportamento, quanto ao uso do elemento <AND>, pode-se pressupor a seguinte situação: se o sensor detectar uma temperatura menor ou igual a 20º e o condicionador de ar estiver ligado, o condicionador de ar será desligado imediatamente.

5.2.3. O Elemento <OR>

O elemento <OR>, apresentado pela Figura 5.5, é o terceiro elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é inferir uma condição lógica OR ao fluxo de execução. Imagine que o usuário queira condicionar a execução do fluxo à execução de duas ou mais ações, ou seja, o fluxo só será executado se, e somente se, uma das ações for executada com sucesso. Eis, a seguir, a ideia:

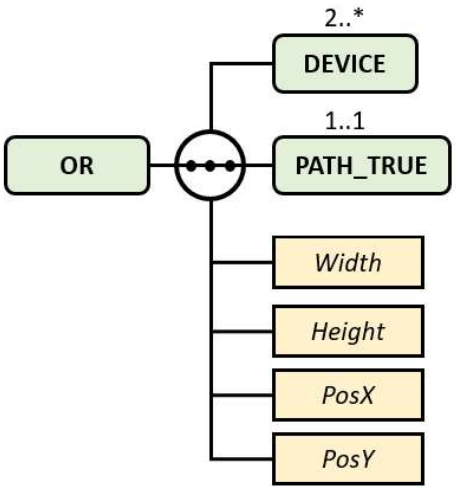


Figura 5.5. Estrutura do elemento <OR>.

A estrutura do elemento <OR> é composta somente pelos atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. Os elementos que o compõem são: <DEVICE>, dispositivos que executaram as ações que deverão ser comparadas (um

elemento <OR> pode conter em sua estrutura 2 ou mais elementos do tipo <DEVICE>); e o <PATH_TRUE>, elemento do fluxo de execução a ser visitado após condição OR ser atendida. A seguir, é apresentado um exemplo de uso do elemento <OR>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
 <OR>
   <DEVICE id=1 name='STEMP1' class='sensor'>
   <OBJECT_TYPE>
    <START_FLOW>
          <TRIGGERED method='get_temp' value=20 minvalue=16
          Maxvalue=30 condiction=5/>
    </START_FLOW>
   </OBJECT_TYPE>
   </DEVICE>
        <DEVICE id=3 name='AC' class='Ar-Condicionado'>
        <OBJECT_TYPE>
         <START_FLOW>
               <TRIGGERED method='ligado' value=1 condiction=0/>
         </START_FLOW>
        </OBJECT_TYPE>
        </DEVICE>
        <PATH_TRUE>
               <DEVICE id=3 name='AC' class='Ar-Condicionado'>
               <OBJECT_TYPE>
                <START_FLOW>
                      <TRIGGERED method='desligado' value=1
               Condiction=0/>
                </START_FLOW>
               </OBJECT_TYPE>
               </DEVICE>
        </PATH_TRUE>
 </OR>
</ACTION>
</BEHAVIOR>
</BDML>
```

O exemplo utilizado para demonstrar o uso do elemento <OR> utilizou o mesmo exemplo utilizado para o elemento <AND>, mas o condicionador de ar será desligado somente quando uma das situações do OR for verdadeira, ou seja, se o sensor detectar temperatura menor ou igual a 20° ou se o condicionador de ar estiver ligado.

5.2.4. O Elemento <SEQUENCE>

O elemento <SEQUENCE>, apresentado pela Figura 5.6, é o quarto elemento pertencente à estrutura do <ACTION>, tendo como sua principal função no modelo de permitir a execução de diferentes sequencias de ações em ordem sequencial, no fluxo de execução de um modelo de comportamento. A estrutura do elemento <SEQUENCE> é composta somente pelos atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”, conforme o seguinte:

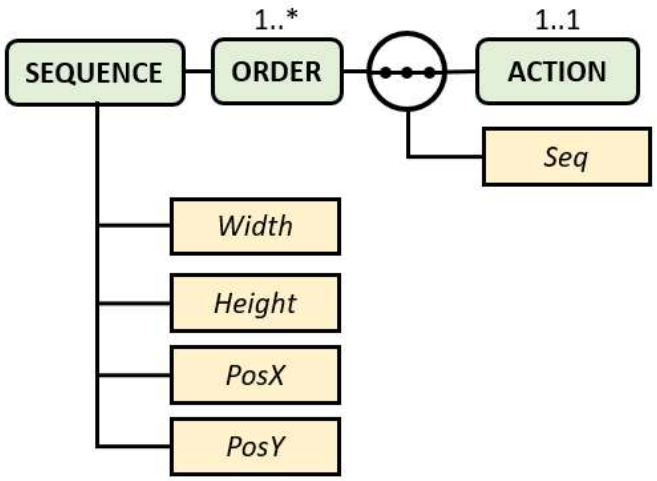


Figura 5.6. Estrutura do elemento <SEQUENCE>.

Entre todos os elementos que compõem, em particular, o elemento <SEQUENCE>, o elemento <ORDER>, que contém a ordem sequencial das ações que deverão ser realizadas, pode conter em sua estrutura uma ou mais ordens de sequência. Logo, o elemento <SEQUENCE> possui um único elemento em sua estrutura, inclusive o elemento <ACTION>, que são as ações a serem executadas. A estrutura do elemento <ORDER> é composta pelo atributo “Seq”, o qual enumera a ordem das ações a serem realizadas. A seguir, é apresentado um exemplo de uso do elemento <SEQUENCE>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='STEMP1' class='sensor'>
<OBJECT_TYPE>
<START_FLOW>
<TRIGGERED method='get_temp' value=20 minvalue=16
        Maxvalue=30 condiction=5/>
</START_FLOW>
<NEXT_ELEMENT>
```

```
<SEQUENCE>
 <ORDER seq=1>
   <ACTION><DEVICE id=2></DEVICE></ACTION>
 </ORDER>
        <ORDER seq=3>
          <ACTION>
     <DEVICE id=4><OBJECT_TYPE><END_FLOW/></OBJECT_TYPE>
           </DEVICE>
          </ACTION>
        </ORDER>
</SEQUENCE>
</NEXT_ELEMENT>
</OBJECT_TYPE>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

No exemplo do uso do elemento <SEQUENCE>, foi criada uma sequência com duas ações distintas, que serão executadas quando o sensor de temperatura marcar um valor menor ou igual a 20°. E finalizadas somente quando um objeto <END_FLOW> for encontrado.

5.2.5. O Elemento <CONDITIONAL>

O elemento <CONDITIONAL>, apresentado pela Figura 5.7, abaixo, é o quinto elemento pertencente à estrutura do <ACTION>, sendo que a sua função precípua no modelo é a de realizar o controle condicional do fluxo de execução de um modelo de comportamento:

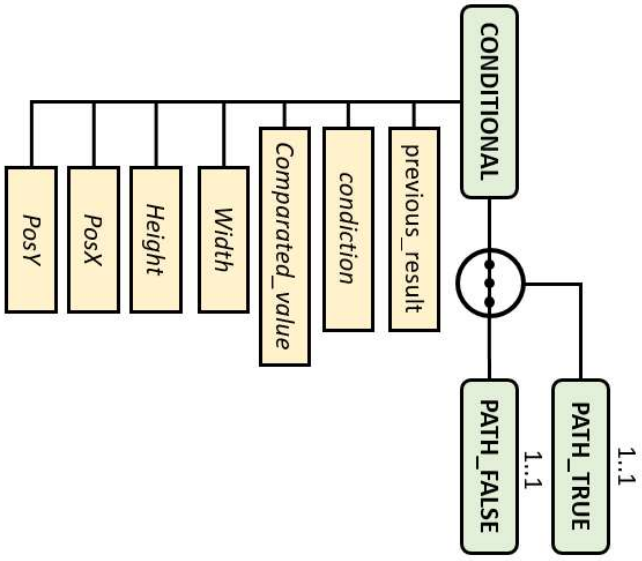


Figura 5.7. Estrutura do elemento <CONDITIONAL>.

A estrutura do elemento <CONDITIONAL> é composta pelos atributos: “previous_result”, variável que recebe o resultado de uma ação executada anteriormente para fins de comparação; “condiction”, condição de comparação; “comparated_value”, valor de comparação; e os atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. Já compondo a estrutura do elemento <CONDITIONAL>, o elemento <NEXT_ELEMENT> é o próximo elemento do fluxo de execução a ser visitado. A seguir, é apresentado um exemplo de uso do elemento <CONDITIONAL>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='STEMP1' class='sensor'>
<OBJECT_TYPE>
<START_FLOW>
<TRIGGERED method='get_temp' value=20 condiction=”>=”/>
</START_FLOW>
<NEXT_ELEMENT>
<CONDICTIONAL previous_result="20"
          Condiction=">=" comparated_value="30" >
 <PATH_TRUE>
  <DEVICE id=1 name="ac" class="Ar Condicionado">
  <OBJECT_TYPE>
  <INTER_FLOW>
   <TRIGGERED method='ligar' value=1
                Minvalue=0 maxvalue=1 condiction=0/>
  </INTER_FLOW>
  </OBJECT_TYPE>
  <NEXT_ELEMENT><DEVICE id=2> </DEVICE>. . </NEXT_ELEMENT>
  </DEVICE>
 <PATH_TRUE>
 <PATH_FALSE>
  <DEVICE id=3>
        <OBJECT_TYPE>
          <END_FLOW/>
        </OBJECT_TYPE>
  </DEVICE>
 </PATH_FALSE>
</CONDICTIONAL>
</NEXT_ELEMENT>
</OBJECT_TYPE>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

No exemplo do elemento <CONDICTION>, foi elaborado o seguinte comportamento: quando o sensor de temperatura marcar uma temperatura maior ou igual a 20, imediatamente após isso, é verificado junto ao elemento <CONDICTION> se a temperatura é maior ou igual a 30°; se sim, segue para execução do elemento <PATH_TRUE>; caso contrário, segue-se para a execução do elemento <PATH_FALSE>.

5.2.6. O Elemento <PAUSE>

O elemento <PAUSE>, apresentado pela Figura 5.8, é o sexto elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é realizar uma pausa temporária no fluxo de execução do modelo de comportamento, conforme o seguinte:

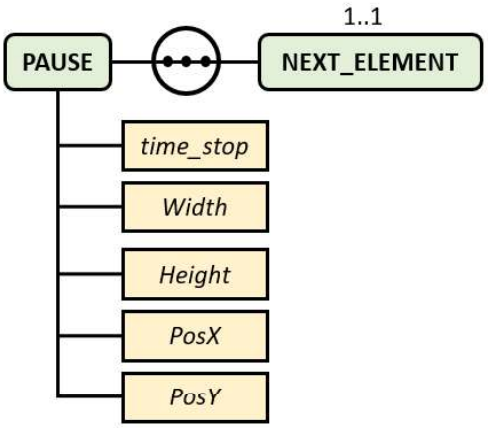


Figura 5.8. Estrutura do elemento <PAUSE>.

A estrutura do elemento <PAUSE> é composta pelos atributos: “time_stop”, responsável por manter o tempo de parada do fluxo de execução; e pelos atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. O elemento que compõe a estrutura do elemento <PAUSE> é o <NEXT_ELEMENT>, o qual é o próximo elemento do fluxo de execução a ser visitado. A seguir, apresenta-se um exemplo de uso do elemento <PAUSE>, conforme se segue:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='STEMP1' class='sensor'>
<OBJECT_TYPE>
<START_FLOW>
<TRIGGERED method='get_temp' value=20 minvalue=16
        Maxvalue=30 condiction=5/>
```

```
</START_FLOW>
</OBJECT_TYPE>
<NEXT_ELEMENT>
<PAUSE time_stop=60>
<NEXT_ELEMENT>
 <DEVICE id=1 name="ac" class="Ar Condicionado">
  <OBJECT_TYPE>
        <INTER_FLOW>
          <TRIGGERED method='ligar' value=1
                    Minvalue=0 maxvalue=1 condiction=0/>
        </INTER_FLOW>
  </OBJECT_TYPE>
  <NEXT_ELEMENT>
    <DEVICE id=2> . . .  </DEVICE>
  </NEXT_ELEMENT>
 </DEVICE>
</NEXT_ELEMENT>
</PAUSE>
</NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

No exemplo do uso do elemento <PAUSE>, foi criado o seguinte comportamento: um sensor, ao detectar uma temperatura menor ou igual a 20°, antes de solicitar o desligamento do aparelho de ar condicionado, faz uma parada no fluxo de execução de 60 minutos; após o termino desse tempo, o condicionador de ar é acionado novamente.

5.2.7. O Elemento <REPETITION>

O elemento <REPETITION>, apresentado pela Figura 5.9, é o sétimo elemento pertencente à estrutura do <ACTION>, tendo como principal função no modelo de realizar repetições no fluxo de execução. Numa abordagem mais simples, imagine que seja necessário realizar N repetições de uma ação ou um bloco de ações pertencentes ao fluxo de execução de um modelo de comportamento: para tal finalidade, faz-se uso do elemento <REPETITION>. A seguir, há uma ilustração sobre o elemento <REPETITION>:

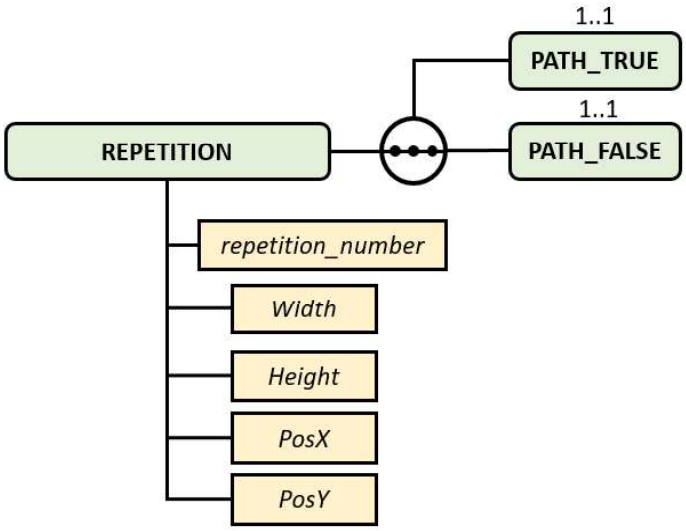


Figura 5.9. Estrutura do elemento <REPETITION>.

A estrutura do elemento <REPETITION> é composta pelos atributos: “repetition_number”, que mantém a quantidade de vezes em que uma ou mais ações devem se repetir; e os atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. Os elementos que compõe a estrutura do elemento <REPETITION> são: o elemento <PATH_TRUE>, que é o bloco de ações que serão repetidos N vezes, e <PATH_FALSE>, caminho que deve ser seguido quando o número de repetições for alcançado. A seguir, é apresentado um exemplo de uso do elemento <REPETITION>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='porta1' class='lock'>
<OBJECT_TYPE>
<START_FLOW>
  <TRIGGERED method='open' value=1 condiction="="/>
</START_FLOW>
</OBJECT_TYPE>
<NEXT_ELEMENT>
<REPETITION repetition_number=20>
  <PATH_TRUE>
        /* Bloco de ações que deverão repetir */
  </PATH_TRUE>
  <PATH_FALSE>
        /* Ações que deverão ser executadas */
       /* após o termino das repetições */
  </PATH_FALSE>
 </REPETITION>
</NEXT_ELEMENT>
```

</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>

O comportamento modelado é iniciado após a fechadura eletrônica identificar a abertura da porta; em seguida, é iniciada a execução do elemento <REPETITION>, que executará 20 vezes as ações existentes dentro do elemento <PATH_TRUE>; e, após finalizadas 20 repetições, as ações existentes no elemento <PATH_FALSE> são iniciadas e o fluxo prossegue com as execuções previstas pelo modelo de comportamento.

5.2.8. O Elemento <SMS>

O elemento <SMS>, apresentado pela Figura 5.10, é o oitavo elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é enviar um SMS a um número previamente especificado. Imagine que o usuário queira receber um SMS sempre que um fluxo de execução for finalizado ou quando a ação de um dispositivo crítico for executada, como, por exemplo, uma detecção de fumaça por um sensor. A seguir, uma ilustração a respeito do elemento <SMS>:

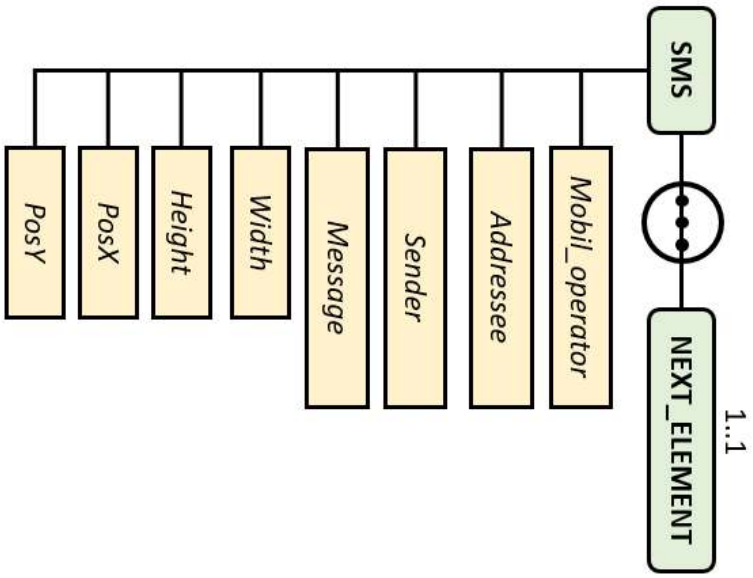


Figura 5.10. Estrutura do elemento <SMS>.

A estrutura do elemento <SMS> é composta pelos atributos: “mobil_operator”, operadora que se deseja utilizar para o envio da mensagem; “Addressee”, destinatário, ou seja, o que receberá a mensagem; “sender”, remetente da mensagem, isto é, o emissor; “message”, que é o corpo da mensagem; e os atributos de coordenadas gráficas “width”,

“height”, “posX” e “posY”. O elemento que compõe a estrutura do elemento <SMS> é o elemento <NEXT_ELEMENT>, que é o próximo elemento do fluxo de execução a ser visitado. A seguir, é apresentado um exemplo de uso do elemento <SMS>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='porta1' class='lock'>
<OBJECT_TYPE>
<START_FLOW>
  <TRIGGERED method='open' value=1 condiction="="/>
</START_FLOW>
</OBJECT_TYPE>
<NEXT_ELEMENT>
  <SMS mobil_operator="zap.vivo.com.br"
      Addressee="5592981427852"
      Sender="559281237845"
      Message="porta da frente foi aberta inesperadamente">
    <NEXT_ELEMENT> . . . </NEXT_ELEMENT>
  </SMS>
</NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

O exemplo do elemento <SMS> descreve o seguinte comportamento: se a fechadura eletrônica da porta principal for aberta, será enviado um SMS para o celular do usuário informando o incidente ocorrido.

5.2.9. O Elemento <MSG>

O elemento <MSG>, apresentado pela Figura 5.11, é o nono elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é apresentar uma mensagem na tela de um dispositivo (computador, tablet, celular, etc.) ao usuário, conforme o seguinte:

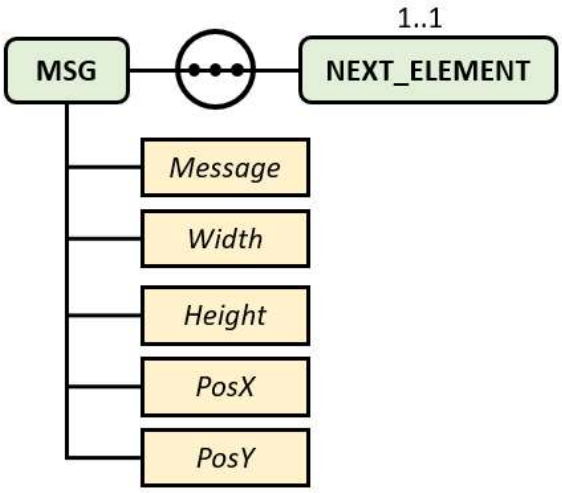


Figura 5.11. Estrutura do elemento <MSG>.

A estrutura do elemento <MSG> é composta pelos atributos: “message”, que é a mensagem de texto a ser apresentada ao usuário; e pelos atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. O elemento que compõem a estrutura do elemento <MSG> é o elemento <NEXT_ELEMENT>, que é o próximo elemento do fluxo de execução a ser visitado. A seguir, é apresentado um exemplo de uso do elemento <MSG>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='porta1' class='lock'>
<OBJECT_TYPE>
<START_FLOW>
  <TRIGGERED method='open' value=1 condiction="="/>
</START_FLOW>
</OBJECT_TYPE>
<NEXT_ELEMENT>
  <MSG message="porta da frente foi aberta inesperadamente">
    <NEXT_ELEMENT> . . . </NEXT_ELEMENT>
  </MSG>
</NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

O exemplo de uso do elemento <MSG> descreve o seguinte comportamento: se a fechadura eletrônica da porta principal for aberta, é enviada uma mensagem ao usuário, a qual poderá ser visualizada em qualquer dispositivo eletrônico, como celular, tablet, notebook, computador pessoal, entre outros.

5.2.10. O Elemento <EMAIL>

O elemento <EMAIL>, apresentado pela Figura 5.12, é o decimo elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é enviar uma mensagem de e-mail ao usuário, segundo pode-se observar a figura a seguir:

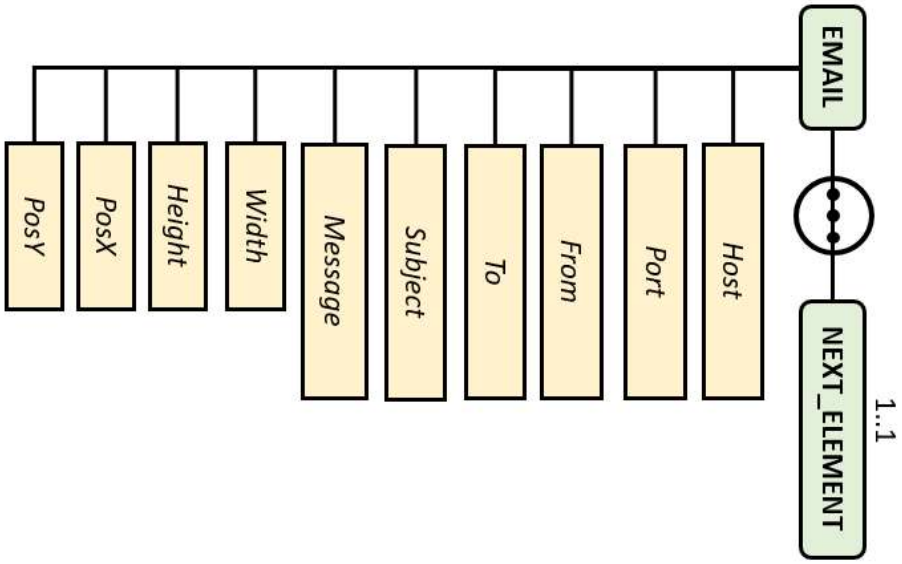


Figura 5.12. Estrutura do elemento <EMAIL>.

A estrutura do elemento <EMAIL> é composta pelos atributos: “host”, o endereço do provedor que enviará o e-mail; “Porta”, que é a porta a ser utilizada pelo host para envio do e-mail; “from”, o endereço de e-mail de quem está enviando a mensagem; “To”, endereço de e-mail de quem receberá a mensagem; “Subject”, assunto da mensagem; “message”, mensagem de texto a ser enviada; e os atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. O elemento que compõe a estrutura do elemento <EMAIL> é o elemento <NEXT_ELEMENT>, que é o próximo elemento do fluxo de execução a ser visitado. A seguir, é apresentado um exemplo de uso do elemento <EMAIL>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='porta1' class='lock'>
<OBJECT_TYPE>
<START_FLOW>
  <TRIGERRED method='open' value=1 condiction="="/>
</START_FLOW>
</OBJECT_TYPE>
```

```
<NEXT_ELEMENT>
  <EMAIL Host="imap.gmail.com"
                 Port=993
                 From="mail@gmail.com"
                 To="usuario@gmail.com"
                 Subject="Avisos"
                 Message="porta foi aberta inesperadamente">
    <NEXT_ELEMENT> . . . </NEXT_ELEMENT>
  </EMAIL>
</NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

O exemplo do elemento <EMAIL> descreve o seguinte comportamento: se a fechadura eletrônica da porta principal for aberta, será enviado um E-Mail para o usuário informando o incidente ocorrido.

5.2.11. O Elemento <SOUND>

O elemento <SOUND>, apresentado pela Figura 5.13, a seguir, é o décimo primeiro elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é emitir um som de alerta ao usuário por meio de um dispositivo (computador, celular, tablet, etc.).

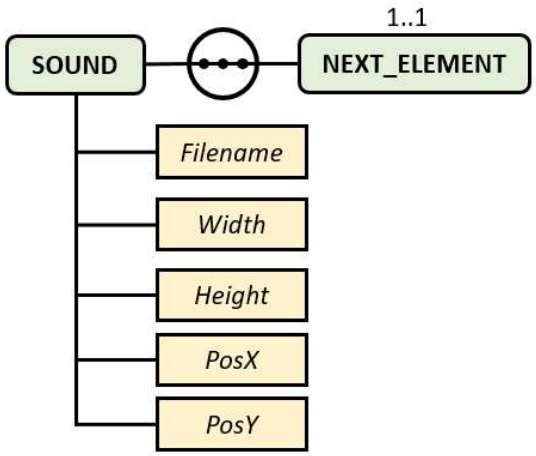


Figura 5.13. Estrutura do elemento <SOUND>.

A estrutura do elemento <SOUND> é composta pelos atributos: “filename”, local onde o arquivo de som a ser emitido está localizado; e pelos atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. O elemento que compõe a estrutura do elemento <SOUND> é o elemento <NEXT_ELEMENT>, que é o próximo elemento do

fluxo de execução a ser visitado. A seguir, é apresentado um exemplo de uso do elemento <SOUND>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='porta1' class='lock'>
<OBJECT_TYPE>
<START_FLOW>
  <TRIGGERED method='open' value=1 condiction="="/>
</START_FLOW>
</OBJECT_TYPE>
<NEXT_ELEMENT>
  <SOUND filename="alerta.mid">
    <NEXT_ELEMENT> . . . </NEXT_ELEMENT>
  </SOUND>
</NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

O exemplo do elemento <SOUND> descreve o seguinte comportamento: se a fechadura eletrônica da porta principal for aberta, será acionado imediatamente um alarme sonoro em um dispositivo do usuário, como celular, smart TV, computador, etc.

5.2.12. O Elemento <CLOUD_UP>

O elemento <CLOUND_UP>, apresentado pela Figura 5.14, a seguir, é o décimo segundo elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é salvar do resultado de uma ação de um dispositivo em um servidor nas nuvens. Imagine que em algum momento do fluxo de execução seja necessário realizar a gravação em vídeo de uma determinada situação que tenha ocorrido no ambiente: para manter o vídeo salvo em um local seguro, o usuário pode utilizar o elemento <CLOUD_UP> para tal situação.

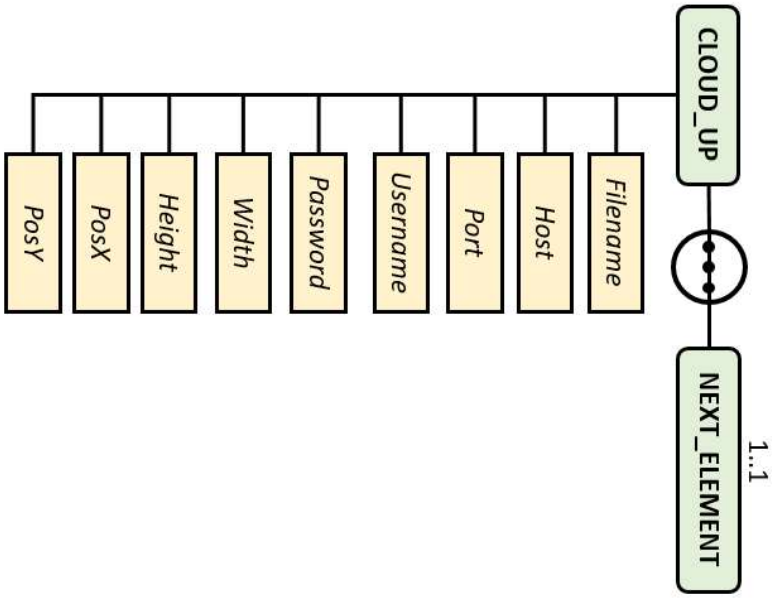


Figura 5.14. Estrutura do elemento <CLOUD_UP>.

A estrutura do elemento <CLOUD_UP> é composta pelos atributos: “host” endereço da nuvem onde o arquivo será enviado; “port”, porta a ser utilizada para envio; “username” e “password”, nome do usuário e senha para autenticação no servidor; e os atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. O elemento que compõe a estrutura do elemento <CLOUD_UP> é o elemento <NEXT_ELEMENT>, que é o próximo elemento do fluxo de execução a ser visitado. A seguir, é apresentado um exemplo de uso do elemento <CLOUND_UP>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='Cam01' class='Camera'>
<OBJECT_TYPE>
<START_FLOW>
  <TRIGGERED method='get_moviment' value=1 condiction="="/>
</START_FLOW>
</OBJECT_TYPE>
<NEXT_ELEMENT>
  <DEVICE id=1 name='Cam01' class='Camera'>
  <OBJECT_TYPE>
  <START_FLOW>
    <TRIGGERED method='set_record' value=1 condiction="="/>
  </START_FLOW>
  </OBJECT_TYPE>
  <NEXT_ELEMENT>
    <CLOUD_UP host="10.0.0.1"
                  Port=8080
```

```
                Username="usuario1"
                    Password="****">
            <NEXT_ELEMENT>
                /* Próxima ação a ser executada */
      </NEXT_ELEMENT>
     </CLOUD_UP>
   </NEXT_ELEMENT>
   </DEVICE>
</NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

O exemplo do elemento <CLOUD_UP> expressa o seguinte comportamento: uma câmera que, ao perceber um movimento, entre imediatamente em modo de gravação, enviando o arquivo de vídeo para um servidor nas nuvens, utilizando o elemento <CLOUD_UP>.

5.2.13. O Elemento <CLOUD_DOWN>

O elemento <CLOUND_DOWN>, apresentado pela Figura 5.15, a seguir, é o décimo terceiro elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é fazer o download de um arquivo armazenado em um servidor nas nuvens. Imagine que o usuário queira realizar a validação de uma imagem armazenada nas nuvens, para, então, seguir com o fluxo de execução. Para isso, faz-se necessário o uso do elemento <CLOUD_DOWN>.

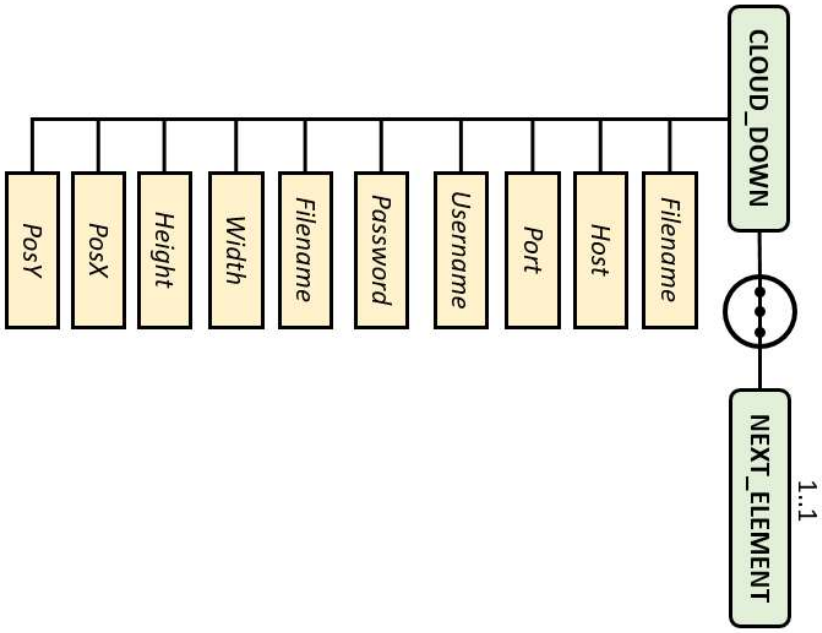


Figura 5.15. Estrutura do elemento <CLOUD_DOWN>.

A estrutura do elemento <CLOUD_DOWN> é composta pelos mesmos atributos utilizados pelo elemento <CLOUD_UP> para conexão e autenticação com o servidor, com exceção do atributo “filename”, nome do arquivo a ser realizado o download, além dos atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. O elemento que compõe a estrutura do <CLOUD_DOWN> é o elemento <NEXT_ELEMENT>, que é o próximo elemento do fluxo de execução a ser visitado. A seguir, é apresentado um exemplo de uso do elemento <CLOUD_DOWN>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='Geladeira' class='Frezzer'>
<OBJECT_TYPE>
<START_FLOW>
  <TRIGGERED method='get_inventary'
          Value=5 condiction="<="/>
</START_FLOW>
</OBJECT_TYPE>
<NEXT_ELEMENT>
  <CLOUD_DOWN host="10.0.0.1"
           Porta=8080
           Username="usuario1"
           Password="******"
           Filename="listacompra.xml">
         <NEXT_ELEMENT>
           <EMAIL>
```

```
            /* Envia E-mail para fornecedor
                  Com a lista de compra em anexo */
               <NEXT_ELEMENT>
                 /* outras ações a serem executadas */
               </NEXT_ELEMENT>
          </EMAIL>
    </NEXT_ELEMENT>
  </CLOUD_DOWN>
</NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

O exemplo do uso do elemento <CLOUD_DOWN> apresenta o seguinte comportamento: uma geladeira inteligente que, ao realizar o inventário dos produtos existentes, verifica que a quantidade de produtos definida pelo usuário está abaixo da meta; logo, é realizado o download da lista de compras, localizada em um servidor nas nuvens e, em seguida, essa lista é enviada ao um fornecedor credenciado, que fornecerá os produtos em falta.

5.2.14. O Elemento <DATABASE_UP>

O elemento <DATABASE_UP>, apresentado pela Figura 16, a seguir, é o décimo quarto elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é salvar o resultado de uma ação em um banco de dados.

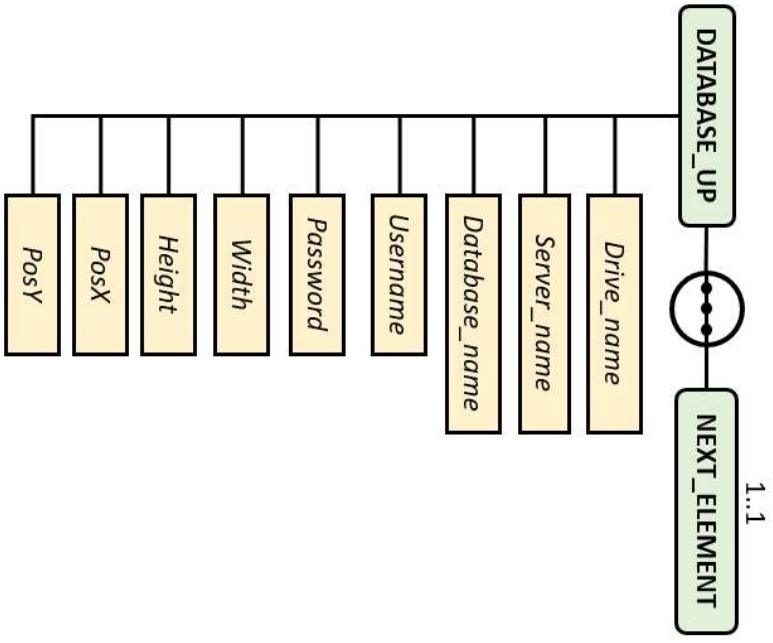


Figura 5.16. Estrutura do elemento <DATABASE_UP>.

A estrutura do elemento <DATABASE_UP> é composta pelos atributos: “drive_name”, drive a ser utilizado para conexão com o banco de dados; “server_name”, endereço do servidor que será utilizado para salvar os dados; “database_name”, nome da base de dados onde os dados serão salvos; “username” e “password” usuário e senha para autenticação e acesso ao banco de dados; e os atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. O elemento que compõe a estrutura do elemento <DATABASE_UP> é o elemento <NEXT_ELEMENT>, que é o próximo elemento do fluxo de execução a ser visitado. A seguir, é apresentado um exemplo de uso do elemento <DATABASE_UP>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='Voltagem' class='Sensor'>
<OBJECT_TYPE>
<START_FLOW>
  <MANUAL method='get_consumo_atual'/>
</START_FLOW>
</OBJECT_TYPE>
<NEXT_ELEMENT>
  <DATABASE_UP drive_name="com.mysql.jdbc.Driver"
            server_name="localhost"
            database_name="log"
            Username="root"
            Password="senha">
    <NEXT_ELEMENT>
                /* outras ações a serem executadas */
    </NEXT_ELEMENT>
  </DATABASE_UP>
</NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

No exemplo de uso o elemento <DATABASE_UP>, foi modelado o seguinte comportamento: o usuário, ao solicitar de um sensor de voltagem, a leitura do consumo atual de um determinado dispositivo eletrônico, enviará essas informações para serem salvas em um banco de dados, que, posteriormente, poderão ser utilizadas para outras finalidades.

5.2.15. O Elemento <DATABASE_DOWN>

O elemento <DATABASE_DOWN>, apresentado pela Figura 5.17, a seguir, é o décimo quinto elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é buscar o resultado de uma ação salva em um banco de dados.

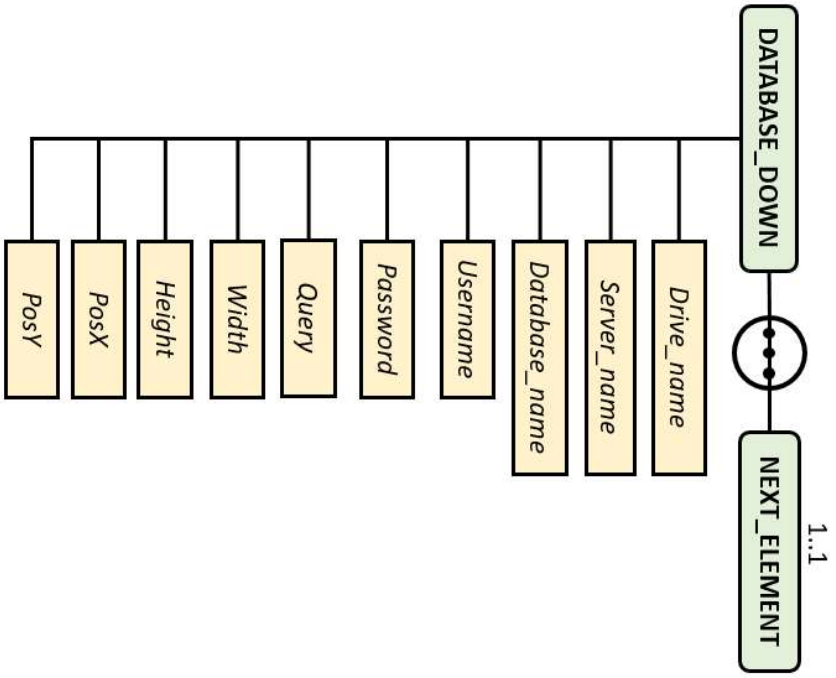


Figura 5.17. Estrutura do elemento <DATABASE_DOWN>.

A estrutura do elemento <DATABASE_DOWN> é composta pelos mesmos atributos utilizados para conexão e autenticação no banco de dados, com exceção do atributo “Query”, consulta a ser realizada para retorno dos dados, e pelos atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. O elemento que compõe a estrutura do elemento <DATABASE_DOWN> é o elemento <NEXT_ELEMENT>, que é o próximo elemento do fluxo de execução a ser visitado. A seguir, é apresentado um exemplo de uso do elemento <DATABASE_DOWN>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='voltagem' class='Sensor'>
<OBJECT_TYPE>
<START_FLOW>
  <MANUAL method='get_consumo_atual'/>
</START_FLOW>
</OBJECT_TYPE>
```

```
<NEXT_ELEMENT>
  <DATABASE_DOWN
   drive_name="com.mysql.jdbc.Driver"
   server_name="localhost"
            database_name="log"
            Username="root"
            Password="senha"
            Query="select consumo from eletronicos where mes = 3">
            <NEXT_ELEMENT>
              /* Realiza processamentos para saber
         O consumo mensal */
   </NEXT_ELEMENT>
  </DATABASE_DOWN>
</NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

No exemplo de uso do elemento <DATABASE_DOWN>, imagine que o usuário queira pegar o consumo atual de um eletroeletrônico e saber o percentual de consumo daquele dispositivo no mês; dessa forma, após buscar o consumo de um dispositivo, o comportamento busca imediatamente todo consumo do mês, a fim de realizar o cálculo que se pede.

5.2.16. O Elemento <FILE>

O elemento <FILE>, apresentado pela Figura 5.18, a seguir, é o décimo sexto elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é salvar o resultado de uma ação executada no fluxo de execução em um arquivo local.

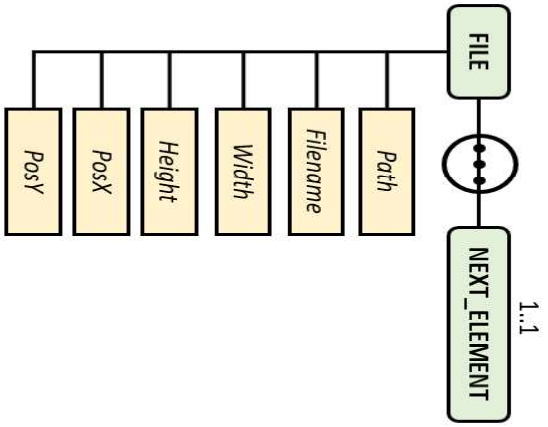


Figura 5.18. Estrutura do elemento <FILE>.

A estrutura do elemento <FILE> é composta pelos atributos: “path”, caminho onde o arquivo deverá ser salvo localmente; “filename”, nome do arquivo a ser salvo; e pelos atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. O elemento que compõem a estrutura do elemento <FILE> é o elemento <NEXT_ELEMENT>, que é o próximo elemento do fluxo de execução a ser visitado. A seguir, é apresentado um exemplo do uso do elemento <FILE>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='voltagem' class='Sensor'>
<OBJECT_TYPE>
<START_FLOW>
  <MANUAL method='get_consumo_atual'/>
</START_FLOW>
</OBJECT_TYPE>
<NEXT_ELEMENT>
  <FILE path="/home/usuario/Documentos"
       Filename="consumo_março.csv">
   <NEXT_ELEMENT>
            /* Realiza processamentos para saber
       O consumo mensal */
   </NEXT_ELEMENT>
  </FILE>
</NEXT_ELEMENT>
</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>
```

No exemplo de uso do elemento <FILE>, imagine que o usuário queira saber o percentual de consumo de um eletroeletrônico em um determinado mês. Contudo, os dados do mês estão armazenados em um arquivo localmente; dessa forma, após buscar o consumo atual do dispositivo por meio do sensor, as demais informações para realização do cálculo são buscadas em arquivo utilizando o elemento <FILE>.

5.2.17. O Elemento <CALC>

O elemento <CALC>, apresentado pela Figura 19, a seguir, é o décimo sétimo elemento pertencente à estrutura do <ACTION>. Sua principal função no modelo é realizar cálculos utilizando o resultado de uma ação executada anteriormente. Imagine que o usuário recebe de um sensor a consumo de energia de um determinado aparelho e o usuário queira saber

quanto, em Reais, ele deverá pagar pelo consumo desse aparelho; para tal, deve-se usar o elemento <CALC>.

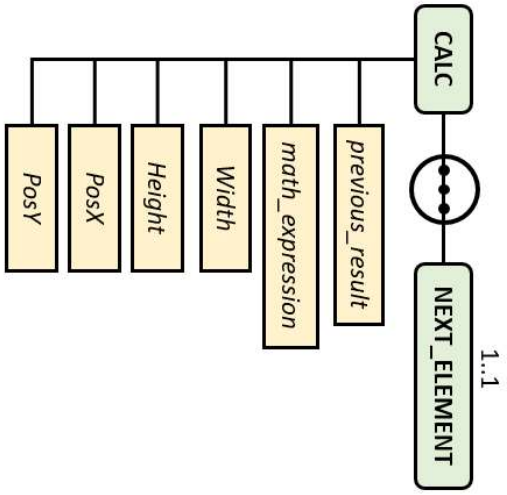


Figura 5.19. Estrutura do elemento <CALC>.

A estrutura do elemento <CALC> é composta pelos atributos: “previous_result”, valor do resultado de uma ação executada anteriormente ao uso do elemento <CALC>; “math_expression”, a expressão matemática utilizada para realização do cálculo que se deseja realizar; e pelos atributos de coordenadas gráficas “width”, “height”, “posX” e “posY”. O elemento que compõem a estrutura do elemento <CALC> é o elemento <NEXT_ELEMENT>, que é o próximo elemento do fluxo de execução a ser visitado. A seguir, é apresentado um exemplo de uso do elemento <CALC>:

```
<BDML filename='modelo-01' data_created='01/06/2017'>
<BEHAVIOR sequence=1>
<ACTION>
<DEVICE id=1 name='voltagem' class='Sensor'>
<OBJECT_TYPE>
<START_FLOW>
  <MANUAL method='get_consumo_atual'/>
</START_FLOW>
</OBJECT_TYPE>
<NEXT_ELEMENT>
  <CALC previous_result="0.5"
       math_expression="$previous_result*0.722173">
   <NEXT_ELEMENT>
            /* executa alguma ação com o resultado
       Obtido, como enviar por e-mail, sms,
       Salvar em banco, nas nuvens, etc. */
   </NEXT_ELEMENT>
  </DATABASE_DOWN>
</NEXT_ELEMENT>
```

</DEVICE>
</ACTION>
</BEHAVIOR>
</BDML>

No exemplo de uso do elemento <CALC>, o usuário gostaria de saber qual o valor a ser pago pelo consumo atual de um dispositivo; dessa forma, esse usuário, ao solicitar a leitura do sensor, que envia a leitura atual para um elemento <CALC>, o qual, por sua vez, possui o cálculo para conversão do consumo em valor monetário.

5.3. Considerações Finais

Devido à falta de modelos conceituais para modelagem de comportamento de ambientes para IoT e, ainda, padrões/linguagens para permitir a criação de modelos conceituais com esse propósito, A BDML surge como linguagem alternativa de se tentar impulsionar o desenvolvimento de modelos conceituais para IoT, por meio dos quais os usuários, com pouco ou quase nenhum conhecimento em computação, possam modelar e/ou modificar todo comportamento do ambiente, automatizando atividades antes executadas manualmente. Além disso, com a criação de modelos conceituais de modelagem de comportamento no ambiente, seria possível possibilitar a integração de ambientes heterogêneos com maior facilidade. Isso se daria por meio da criação de um padrão de comunicação entre os ambientes e a criação de um objeto responsável por permitir essa integração. Na BDML, é possível adicionar novos objetos de regras de negócio, devendo o usuário apenas definir o novo objeto no *XML Schema* da BDML, suas características e comportamentos.

Cabe ressaltar que a BDML é uma linguagem da criação de modelos conceituais de comportamento, que auxilia os usuários especialistas no sentido de criar um modelo conceitual visual e, até mesmo, novos objetos à linguagem, a fim de se atender às suas necessidades do ambiente ao qual será adaptada. A BDML não foi criada para atender a um ambiente específico, pois todas as entidades atualmente existentes no modelo podem ser utilizadas para se modelar qualquer ambiente para IoT. Contudo, é possível que um usuário queira adaptar a linguagem BDML, a fim de se atender a um ambiente especifico.

Por intermédio da BDML, é possível vislumbrar que, futuramente, a IoT esteja disponível a qualquer usuário, favorecendo-os a se tornarem peça fundamental na IoT, ou

seja, poderão emergir usuários que não faça apenas o uso de aplicações de terceiros, mas como projetistas de seus próprios ambientes para IoT.

Capítulo 6 - O Framework Êxodo

A palavra *Êxodo* tem origem do latim *Exodus*, que significa a emigração de um grupo de pessoas de uma região para outra, ocorrendo, normalmente, em busca de melhores condições de vida. Nesse contexto, o nome *Êxodo* sugere a saída dos usuários não especialistas de uma situação de difícil acesso para uma abordagem mais intuitiva e expressiva, representada pelo modelo BDM4IoT e pela interface gráfica *Êxodo GUI*, que, juntos, o possibilitam a modelar e manipular o ambiente de acordo com as suas necessidades.

Em virtude dessa ideia, emergiu *O framework*, criado com o propósito de prover uma solução para IoT de modo a permitir que usuários com diferentes níveis de conhecimento possam realizar o seguinte: i) criar/modelar seus próprios ambientes para IoT; ii) gerenciar os dispositivos presentes no ambiente; e iii) manipular as informações geradas pelos dispositivos.

Cabe destacar que o *framework Êxodo* é uma solução para IoT centrada no usuário, cujas funcionalidades implementadas são disponibilizadas na forma de serviços. Em função disso, foi adotada uma abordagem SOA (*Service-Oriented Architecture - SOA*), facilitando a comunicação, de forma a que as soluções se comunicar, a fim de trocar informações e serviços entre si.

Quanto o *framework Êxodo*, suas aplicações que o compõem foram desenvolvidas com foco em flexibilidade, facilitando sua extensão e adaptabilidade em relação às necessidades dos usuários e do ambiente em si. Para alcançar esse objetivo, foi desenvolvido um conjunto de classes extensíveis, com base em padrões de projeto, a fim de que os usuários especialistas possam: i) criar e integrar novas aplicações ao *framework*; ii) adaptar a solução para atender a diferentes domínios; iii) inclusão de novos protocolos e dispositivos; e iv) criar e registrar novos serviços a arquitetura da solução.

Dessa forma, este Capítulo apresenta o *framework Êxodo*, sua arquitetura e principais aplicações, estando organizado da seguinte forma: a Seção 6.1 apresenta os requisitos funcionais e não funcionais a serem alcançados pela solução proposta; a Seção 6.2 fornece uma visão geral do *framework Êxodo*, os seus componentes e as técnicas

utilizadas para permitir a inclusão de novas soluções ou extensão das soluções existentes; e a Seção 6.3 discute os aspectos abordados na concepção do *framework*.

6.1. Requisitos da Solução

Para alcançar os objetivos propostos, um conjunto de requisitos, a princípio, deve ser atendido, a respeito das tipologias de gerenciamento, abaixo descritas:

- **Gerenciamento de dispositivos –** O *framework* deve gerenciar os dispositivos de forma individual, permitindo que cada um esteja acessível aos demais dispositivos gerenciados, a fim de possibilitar a troca de informações e serviços entre eles;
- **Gerenciamento de dados –** O *framework* deve permitir que os dados gerados pelos dispositivos e capturados do ambiente sejam armazenados e disponíveis para os demais dispositivos, componentes do *framework* e usuários. Deve, também, facilitar sua manipulação e transformação, especialmente para gerar relatórios e/ou informações estatísticas, como, por exemplo, o consumo de energia gasto por um ou mais dispositivos, em um determinado período de tempo;
- **Gerenciamento de eventos de contexto –** O *framework* deve, constantemente, prover meios para monitoramento do ambiente, a fim de se identificar os eventos disparados, além de verificar se existe algum comportamento relacionado ao evento que, porventura, precise ser executado. Por exemplo, quando um sensor de movimento for disparado em uma área restrita, o *framework* deverá verificar se existe algum modelo de comportamento relacionado ao evento detectado. Caso exista, o modelo de comportamento é executado conforme as definições do usuário;
- **Modelagem de comportamento –** O *framework* deve permitir que usuários modelem os diferentes comportamentos do ambiente;
- **Interoperabilidade –** O *framework* deve permitir que os dispositivos heterogêneos presentes no ambiente se comuniquem, mesmo utilizando diferentes interfaces de comunicação;
- **Usabilidade –** O *framework* deve permitir que usuários especialistas consigam estender as principais funcionalidades da solução, sem que, para isso, tenham

conhecimentos aprofundados sobre os componentes internos da arquitetura. Além disso, possibilitam que usuários não especialistas possam, por meio de modelos e interfaces gráfica, modelar os diferentes comportamentos do ambiente para IoT; e

- **Flexibilidade** - O *framework* deve permitir que seus componentes sejam flexíveis o suficiente para inclusão de novos serviços e, até mesmo, estendidos e reutilizados por outras soluções.

6.2. Arquitetura do *Framework Êxodo*

A arquitetura do framework Êxodo é composta por três camadas principais, que exercem um papel mais relevante no que se refere ao bom funcionamento da solução, quais sejam: i) a camada de aplicações e dispositivos, responsável por fornecer meios para possibilitar que aplicações externas e dispositivos sejam gerenciados pelo *framework*; ii) o barramento de serviços denominado como ESB (do inglês, *Êxodo Service Bus*), responsável por realizar a troca de informações e serviços entre a camadas superior (aplicações e dispositivos) e inferior (*middleware*); e iii) a camada de middleware, responsável por realizar os principais gerenciamentos e processamentos fornecidos pelo *framework*. A Figura 6.1, a seguir, fornece uma visão geral da arquitetura do *framework* Êxodo, suas principais aplicações, componentes e interfaces:

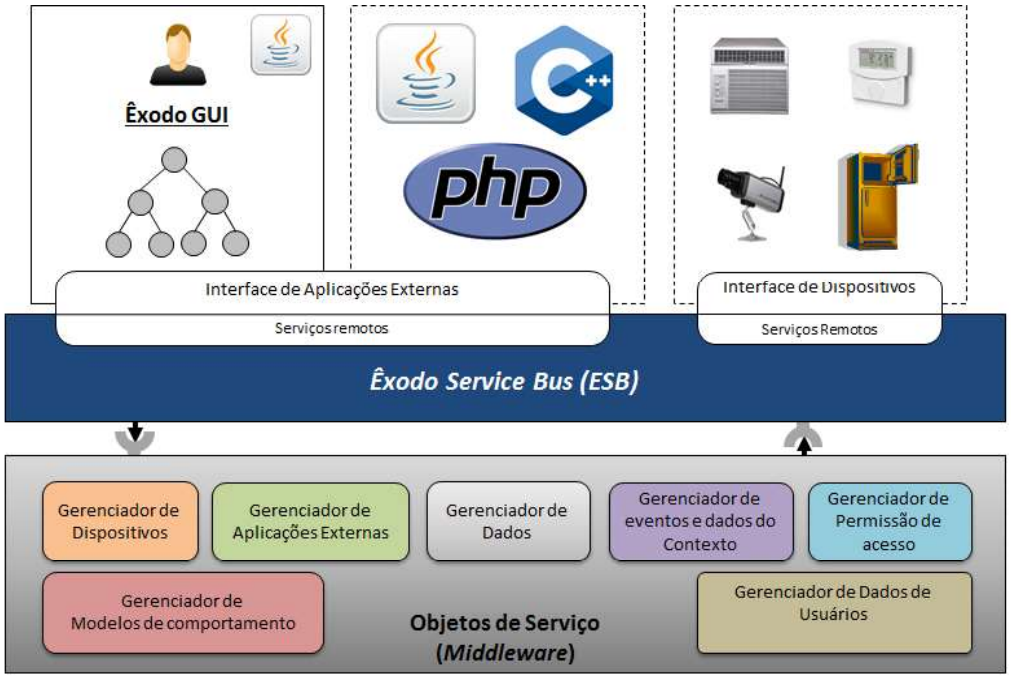


Figura 6.1. Visão integrada do *Framework Êxodo*, seus componentes e interfaces.

As seções, a seguir, destinam-se a apresentar, detalhadamente, as principais camadas que compõem a arquitetura do *framework* Êxodo: a seção 6.2.1 apresenta a camada de aplicação *middleware* e seus principais componentes; já a seção 6.2.2 vai apresentar o barramento de serviços ESB, os principais serviços e interfaces projetados na integração de novas soluções ao *framework*; e, para concluir, a seção 6.2.3 apresenta a camada de aplicações e dispositivos.

6.2.1. A Camada de *Middleware*

A camada de *middleware* agrupa as principais funcionalidades e processamentos executados pelo *framework*, tais como: i) o gerenciamento dos dispositivos; ii) gerenciamento de aplicações externas; iii) gerenciamento dos dados; iv) gerenciamento dos eventos de contexto; v) gerenciamento dos modelos de comportamento; vi) gerenciamento das permissões de acesso aos serviços; e vii) gerenciamento dos usuários. A Figura 6.2 ilustra, judiciosamente, a camada de *middleware* e seus componentes principais, a seguir:

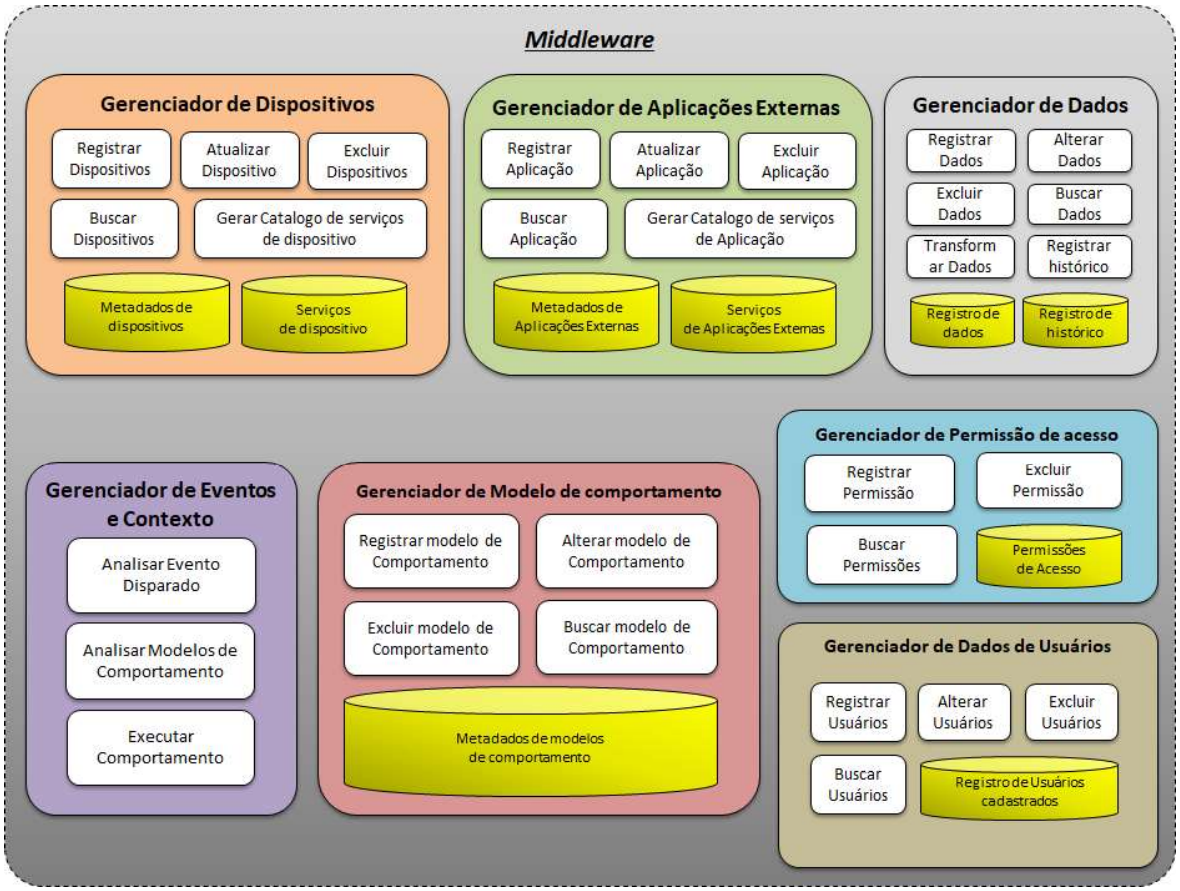


Figura 6.2. O componente *Middleware* e seus principais serviços.

O componente “gerenciador de dispositivos” é o responsável por controlar os dispositivos e serviços ativos no *framework*. Os seus principais serviços executados são: i) registrar dispositivos; ii) atualizar dispositivos; iii) excluir dispositivos; iv) buscar dispositivos; e v) gerar catalogo de serviços de dispositivo.

O serviço “registrar dispositivos” é responsável por adicionar novos dispositivos na base de dados do *framework Êxodo*, sendo que, para isso, faz-se necessário passar as seguintes informações: URI (*Uniform Resource Identifier*) do dispositivo, que é o endereço por meio do qual se dá o acesso às informações e serviços fornecidos pelo dispositivo, nome, classe, modelo, fornecedor, MAC, status e localização. Status é um importante parâmetro utilizado pelo *framework* para saber se um dispositivo está ou não disponível ao acesso. A Figura 6.3, abaixo, apresenta o esquema de dados das informações dos dispositivos, serviços e dados gerados por esses dispositivos:

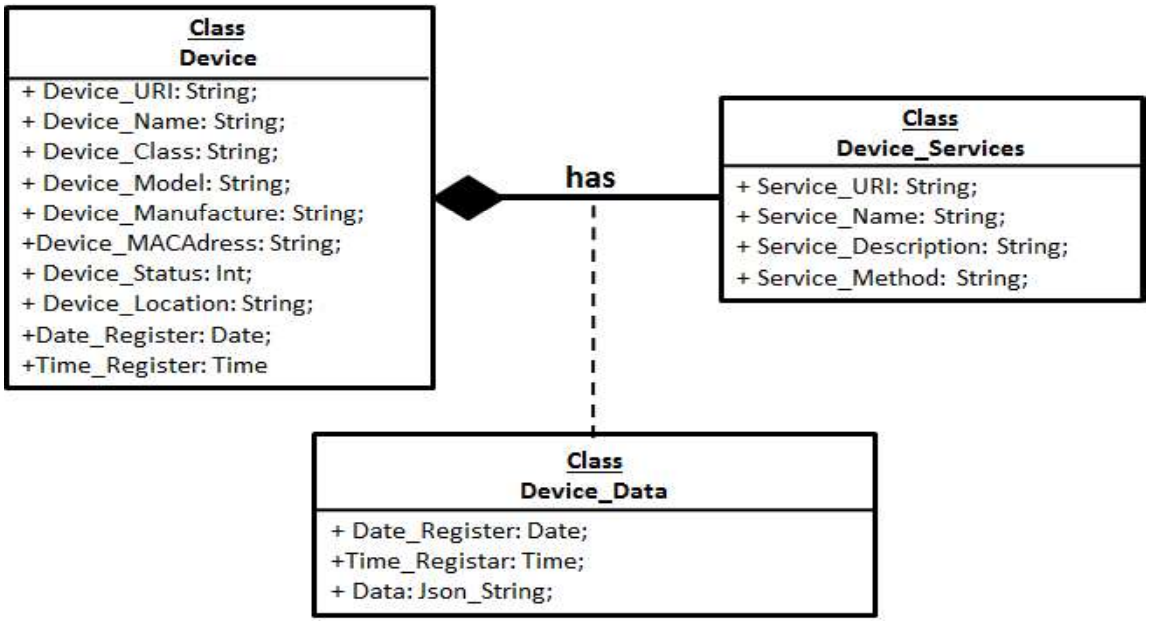


Figura 6.3. Esquema de dados de dispositivos.

A classe “Device” mantém as informações referentes ao dispositivo como nome, fornecedor, classe, entre outros. A classe “Device_Services” mantém as informações referentes aos serviços providos pelo dispositivo e, para cada um desses, existe uma URI que possibilita o acesso e execução dos serviços. A classe “Device_Data” é composta pela data e hora em que uma informação foi registrada e pelo JSON, que mantém a informação gerada pelo dispositivo.

O serviço “atualizar dispositivo” é responsável por atualizar as informações de um dispositivo já cadastrado na base de dados do *framework*. Para sua utilização, o usuário deverá informar uma classe do tipo “Device”, contendo todas as informações do dispositivo que se deseja alterar.

O serviço “excluir dispositivos” utiliza a URI do dispositivo para fins de exclusão da base de dados do *framework*, bastando ao usuário informar apenas a URI do dispositivo que se deseja excluir.

O serviço “buscar dispositivos” retorna todas as informações do dispositivo, bem como dos seus serviços providos. Por fim, o componente “gerenciador de dispositivo” também fornece um tipo de serviço responsável por gerar o catálogo de serviços gerenciados pelo *framework*, que, ainda, fornece ao usuário uma lista com todos os dispositivos e serviços fornecidos.

O componente “gerenciador de aplicações externas” tem a finalidade bem parecida com a do componente “Gerenciador de Dispositivos”, mas voltado para o gerenciamento das aplicações externas incorporadas ao *framework*. O componente “Gerenciador de Aplicações Externas” é responsável por: i) registrar novas aplicações; ii) atualizar informações; iii) excluir aplicações; iv) buscar aplicações; e v) gerar catálogo de serviços de aplicações. É importante ressaltar que, para o *framework Êxodo*, dispositivos, aplicações externas e qualquer outro fornecedor seu de serviço incorporado são denominados como “Coisas”. A Figura 6.4, a seguir, apresenta o esquema de dados das aplicações externas:

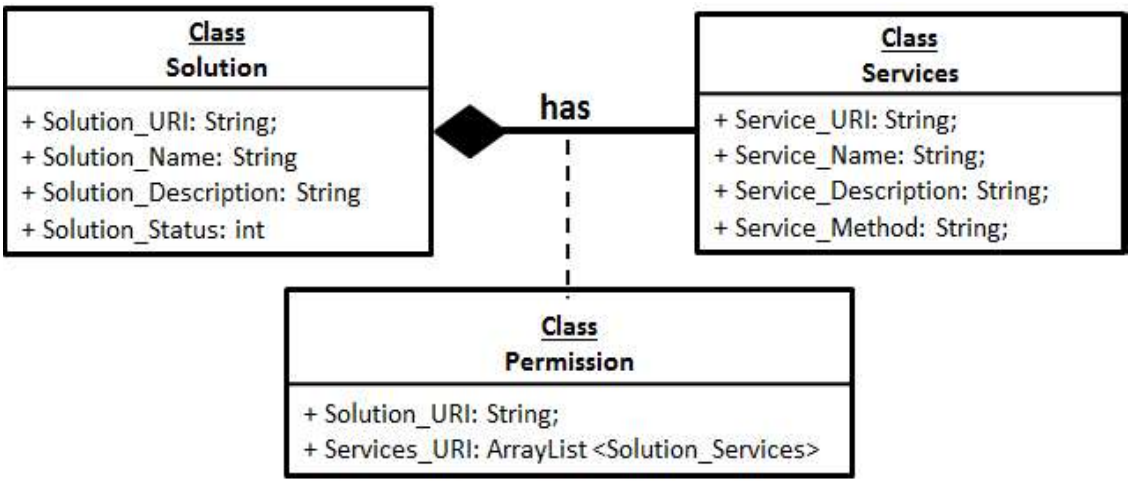


Figura 6.4. Esquema de dados utilizado para gerenciamento das aplicações externas.

A classe *Solution* contém as principais informações das aplicações externas, como, URI, nome, descrição e status, enquanto a classe *Services*, as informações dos serviços providos pelas aplicações como URI, nome, descrição e o escopo do método, que dá acesso ao serviço.

O componente "gerenciador de dados" é responsável por controlar o registro, alteração, busca e exclusão dos dados gerados pelas "Coisas" gerenciadas pelo *framework*. Além dos serviços básicos de CRUD (acrónimo de *Create*, *Read*, *Update* e *Delete*), esse componente é também responsável por gerenciar o *log* ou o histórico dos serviços e eventos executados pelos usuários e/ou dispositivos. Outro importante serviço oferecido por esse componente é a possibilidade de converter os dados para os seguintes formatos: XML, JSON, CSV (*Comma-Separated Value*) e PFD (*Portable Document Format*). Contudo, o *framework* não se limita apenas a esses formatos. Para possibilitar maior flexibilidade quanto à inclusão de novos formatos, foi utilizado o padrão de projeto Strategy (Freeman, 2007) para implementação desse serviço, conforme pode ser observado pela Figura 5.5:

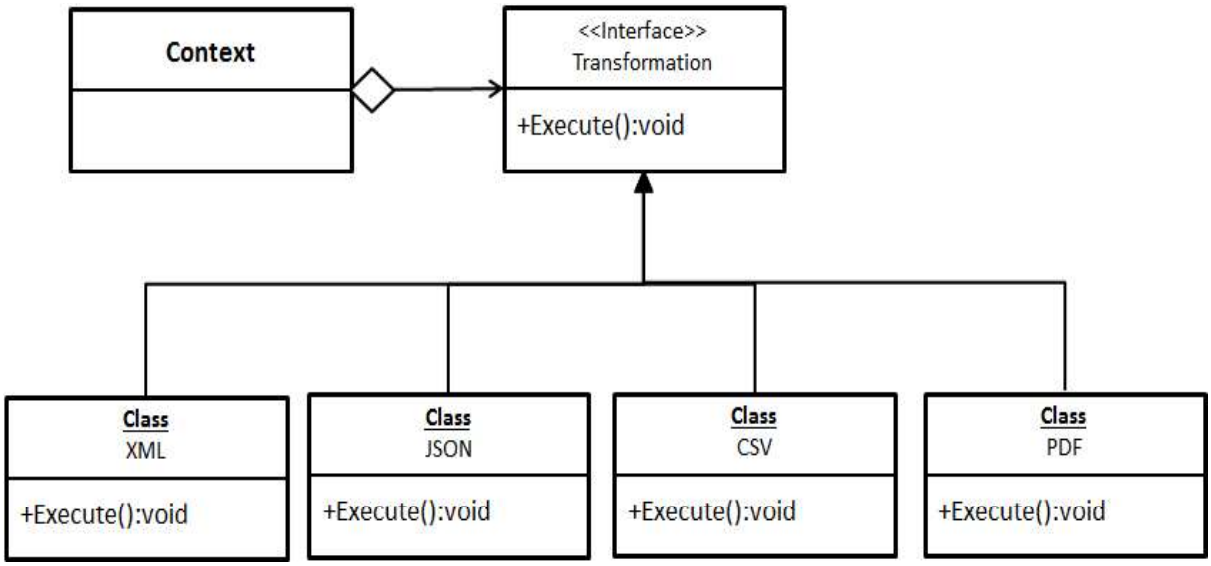


Figura 6.5. Transformação de dados utilizando o padrão de projeto *Strategy*.

O componente "gerenciador de eventos e contexto" é responsável por analisar os eventos disparados no ambiente e verificar, na base de modelos de comportamento, se existe algum que faça uso do evento como um inicializador de fluxo de execução, a fim de se realizá-la. Primeiramente, o serviço "analisar evento disparado" recebe os dados de um evento que tenha sido disparado, analisa e seleciona todos os modelos de comportamento

que possuem relação com o evento; em seguida, o serviço "analisar modelos de comportamento", que recebe todos os modelos selecionados e aplica filtros para selecionar somente os modelos de comportamento que realmente precisam ser executados; por fim, o serviço "executar comportamento", que carrega os comportamentos em uma estrutura de dados do tipo árvore e realiza a execução do comportamento conforme apresentado nos Algoritmos 1 e 2, respectivamente:

ALGORITMO 1: Iniciar execução de comportamentos

```
1: Parâmetros: Modelo de comportamento no formato BDML
2: BHVTree ← Get_BHVTree (BDML);
3: enquanto BHVTree <> nulo faça
4:     Nd ← Get_StartNode (BHVTree);
5:     executar comportamento (Nd);
6:     BHVTree ← Get_BHVTree (BDML);
7: fim enquanto
```

O Algoritmo 1 realiza a leitura do modelo de comportamento em formato BDML e extrai todos os comportamentos existentes nesse arquivo, carregando cada comportamento extraído em uma estrutura do tipo árvore, denominada como BHVTree (do inglês, *Behavior Tree*, na linha 2); após a extração do comportamento, a função *Get_StartNode* é executada e retorna o elemento raiz da árvore, responsável por iniciar o fluxo de execução do comportamento (linha 4); após isso, o comportamento é executado pelo algoritmo 2 (linha 5). Este processo é repetido até que todos os comportamentos tenham sido executados.

ALGORITMO 2: Executar comportamento

```
1: Precondições: BHVTree = (V, E) <> nulo, onde BHVTree é um grafo de comportamento
   composto de objetos (E) e ligações (V).
2: Parâmetros: Nd = [ fn, k= {V1, V2, Vn} ], onde Nd é um E ∈ BHVTree;
3: enquanto não for o fim da lista k ∈ Nd faça
4:   se Nd é um dispositivo de início de fluxo então
5:     se Nd é do tipo engatilhado então
6:       se Resultado de fn ∈ Nd atende a condição de disparo então
7:         executar comportamento (Vn ∈ Nd);
8:       fim se
9:     fim se
10:    se Nd é do tipo temporizado então
11:      se data e hora atual atende a condição para execução então
12:        Resultado ← Nd.fn();
13:        se Resultado de fn ∈ Nd atende a condição de disparo então
14:          executar comportamento (Vn ∈ Nd);
```

```
15:           fim se
16:         fim se
17:       fim se
18:       se Nd é do tipo manual então
19:         Resultado ← Nd.fn ();
20:         se Resultado de fn ∈ Nd atende as condições de disparo então
21:           executar comportamento (V_n ∈ Nd);
22:         fim se
23:       fim se
24:     fim se
25:     se Nd é um dispositivo intermediário ou de fim de fluxo então
26:         Resultado ← Nd.fn ();
27:         se Resultado de fn ∈ Nd atende as condições de disparo então
28:           executar comportamento (V_n ∈ Nd);
29:         fim se
30:     fim se
31:     se Nd é um objeto auxiliar então
32:       Resultado ← Nd.fn ();
33:       se Resultado de fn ∈ Nd atende as condições de disparo então
34:         executar comportamento (V_n ∈ Nd);
35:       fim se
36:     fim se
37: fim enquanto
```

O Algoritmo 2 é responsável por realizar a execução do comportamento, sendo necessário percorrer toda a árvore de comportamento (*BHVTree*) a partir do nó raiz, até que todos os nós tenham sido visitados e suas ações executadas (linha 3). Um nó (*Nd*) da árvore de comportamento (*BHVTree*) é composto de uma função (*fn*), responsável por executar o comportamento do nó, como, por exemplo, enviar e-mail, ligar ou desligar um eletroeletrônico, entre outros, e uma lista (*k*) de vértices, que estão diretamente relacionados a *Nd*.

Ao ser executado, o algoritmo verifica o tipo de nó que está sendo analisado (linhas 4, 25 e 31). Caso seja um nó de início de fluxo de execução, o tipo de disparo utilizado pelo nó é verificado (linhas 5, 10 e 18), visto que o disparo, do tipo engatilhado, deve ser executado imediatamente após a execução do evento que desarma o gatilho. Disparos do tipo temporizado e manual dependem de eventos externos para serem executados, como o tempo (data e hora) e ação do usuário. Dessa forma, tais disparos são tratados separadamente (linhas 10 e 18). Após a execução da ação, é necessário verificar se as condições de disparo foram atendidas, a fim de que o algoritmo possa seguir para o

próximo nó, realizando a chamada recursiva do método e passando como parâmetro o nó a ser visitado (linhas 7, 14, 21, 28 e 34).

A definição do comportamento de um nó é criada dentro da função (*fn*). Assim, para que o comportamento de um nó possa ser executado, basta realizar a execução de *fn*. A Figura 6.6, a seguir, apresenta a estrutura de classe utilizada pelo nó *Nd*:

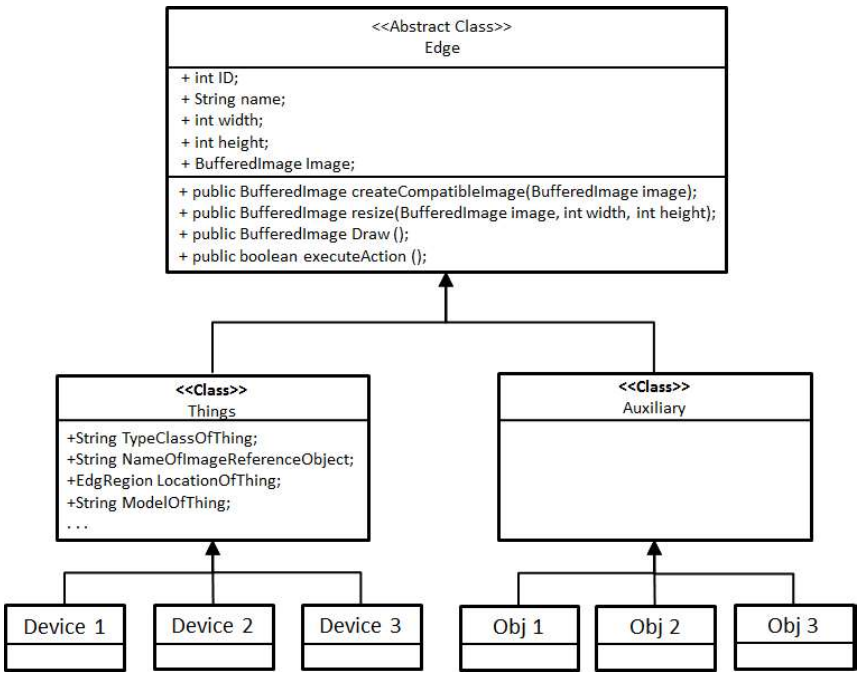


Figura 6.6. Esquema de classes de objetos do *framework Êxodo*.

A classe pai *Edge* possui todas as características e métodos comuns a todos os objetos, como identificador, nome, largura, altura e a imagem que representa objeto, além dos métodos responsáveis por redimensionar, arrastar e soltar, controle da imagem em *buffer*, entre outros. Somando-se a isso, todo nó *Nd* ∈ *BHVTree* é do tipo *Edge*, devido a um nó *Nd* poder assumir diferentes tipos de objetos, como *Things*, *Auxiliary* ou qualquer outra classe que estenda a classe pai *Edge*. A Classe *Things* representa os dispositivos físicos existentes no mundo real, como condicionador de ar, geladeira, televisão, *smart phone*, entre outros. Já a classe *Auxiliary* representa os objetos de regras de negócio existentes no modelo BDM4IoT, como *e-mail*, *sound*, *pause*, entre outros.

O componente "gerenciador de modelo de comportamento" é responsável por fornecer os serviços para registro e controle dos modelos de comportamento criados pelo usuário. Além disso, este componente fornece um serviço de busca que permite ao usuário localizar um modelo registrado na base de modelos do *framework*.

O componente “gerenciador de permissão de acesso” controla todas as permissões de acesso dos usuários, aplicações externas e dispositivos aos serviços gerenciados pelo *framework Êxodo*. Tais permissões são geralmente atribuídas a um usuário, uma aplicação externa ou, até mesmo, a um dispositivo que necessita fazer uso do serviço de outros dispositivos. Além disso, o componente fornece um serviço de listagem de permissões concedidas, a fim de fornecer um relatório ao usuário para controle das permissões.

O componente “gerenciador de dados de usuário” gerencia e controla os dados dos usuários cadastrados na base de dados do *framework*. Os principais serviços providos por esse componente são: registro, alteração, exclusão e busca do usuário na base de dados do *framework*. O serviço buscar usuário é o responsável por retornar os dados de um usuário, de acordo com as seguintes informações: código identificador, apelido ou nome completo.

6.2.2. A Camada de Aplicação *Êxodo Service Bus* (ESB)

O ESB (*Êxodo Service Bus*) é o barramento central do *framework Êxodo*, responsável por permitir que aplicações externas e dispositivos vinculados ao *framework* possam interoperar, a fim de compartilhar dados e serviços entre si. Assim, deve-se solicitar qualquer requisição de dados ou execução de serviços por meio de ESB, o qual, por sua vez, repassa essas solicitações às aplicações responsáveis. Para isso, o ESB dispõe de um conjunto de serviços, que são acessados remotamente pelas aplicações externas, através de interfaces criadas para esse propósito.

Além da responsabilidade de prover a comunicação, o ESB é também responsável por prover a segurança das requisições de dados e serviços tramitados pelo *framework*. O ESB possui em sua arquitetura um serviço para encriptar e decriptar as requisições trocadas entre as aplicações externas e os dispositivos com o *middleware*.

A segurança dos dados no barramento ESB é garantida utilizando-se o algoritmo 3DES (*Triple Data Encryption Standard*) [Kelsey, Schneier e Wagner 1996], que é uma versão melhorada do algoritmo DES. No 3DES, os dados são encriptados com a primeira chave, decriptados com a segunda chave e, finalmente, encriptados novamente com a terceira chave, fazendo-o ser mais lento que o DES original. Por outro lado, o algoritmo 3DES oferece maior segurança, ressaltando que é um algoritmo de cifra simétrica, ou seja, a chave utilizada para criptografar é a mesma utilizada para descriptografar.

Os principais serviços providos pelo ESB para as aplicações externas e dispositivos são: i) Registrar Aplicação Externa; ii) Alterar Status da Aplicação Externa; iii) Consumir Serviço; iv) Registrar Dispositivo; v) Escutar Eventos Disparados; vi) Receber Mensagens; vii) Buscar Catálogo de Serviços; e viii) Alterar Status do Dispositivo. Tais serviços são classificados no ESB como públicos, ou seja, são serviços que estão disponíveis para acesso por qualquer aplicação externa ou dispositivo por meio de suas respectivas interfaces de acesso. Além dos serviços públicos, o ESB possui quatro serviços privados, disponíveis apenas ao ESB e *Middleware*, como: i) Reportar Erro; ii) enviar dados para monitoramento; iii) criptografar; e iv) descriptografar. A Figura 6.7 apresenta o ESB e seus principais serviços:

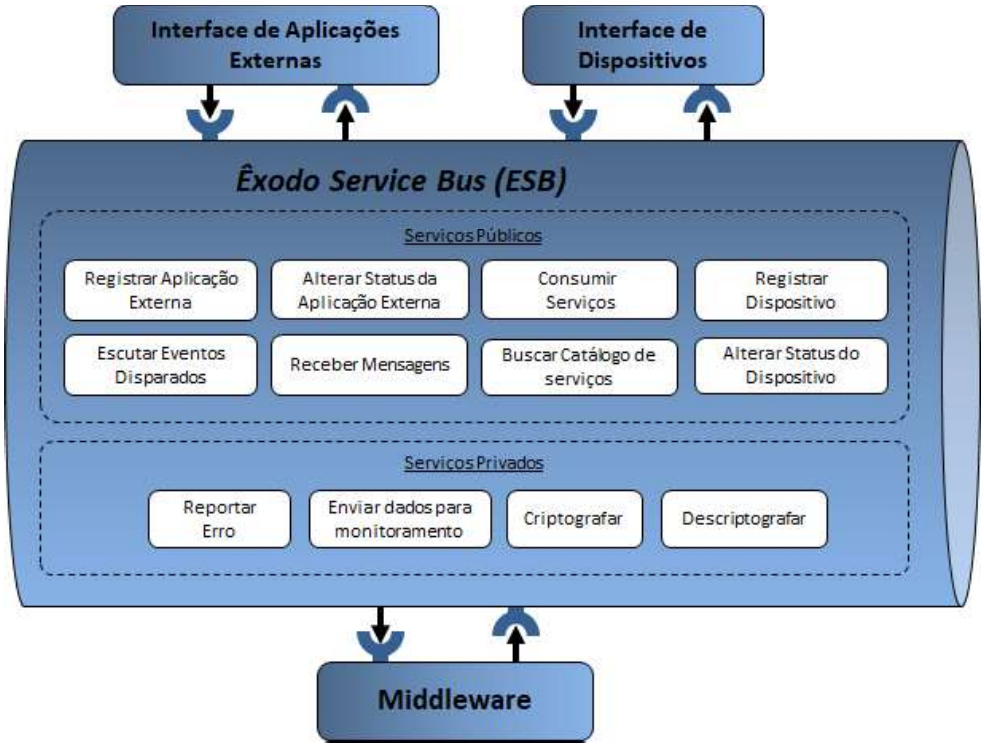


Figura 6.7. ESB e seus principais serviços.

Para que aplicações externas e dispositivos possam fazer uso do ESB, é necessário que façam uso de suas respectivas interfaces, as quais permitem que possam acessar os serviços públicos providos pelo ESB, com a finalidade de realizar suas principais atividades, que são: i) o serviço público "registrar aplicação externa" possibilita que os usuários registarem as informações e serviços providos por uma aplicação externa, permitindo que essa possa ser integrada ao *framework*. Para o registro de uma aplicação, é necessário que o usuário informe os seguintes dados: 1) URI da aplicação, que é utilizada

no *framework* como identificador único; 2) nome; 3) descrição; 4) status; e 5) uma lista com as informações dos serviços providos pela aplicação.

Para facilitar o entendimento do processo realizado no registro de aplicações externas, a Figura 6.8 apresenta o diagrama de sequência do processo de registro de aplicações, conforme se segue:

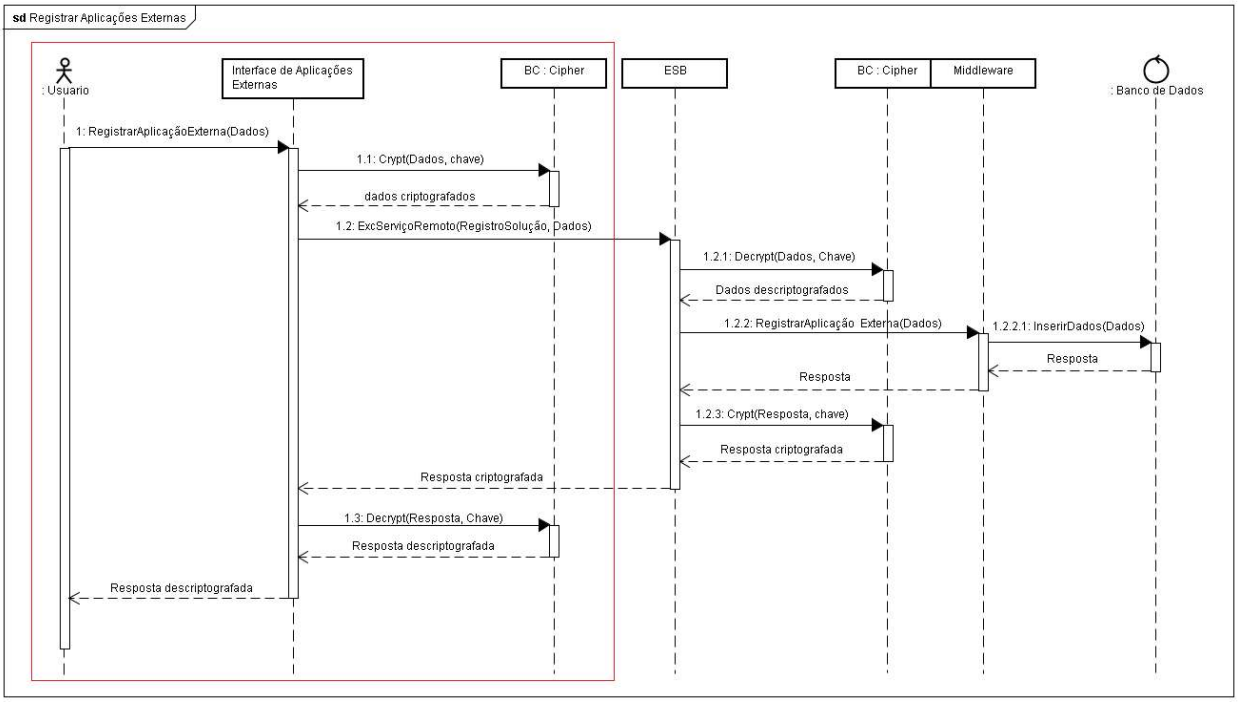


Figura 6.8. Processo de registro de aplicações externas.

O processo para registro de aplicações é iniciado após o envio dos dados da aplicação pelo usuário, utilizando o serviço "registrar aplicação externa", passando, como parâmetro, os dados da aplicação. Antes dos dados serem enviados ao ESB, é necessário que esses sejam encriptados. O ESB, ao receber os dados encriptados, realiza a sua decriptação e os envia para serem efetivamente registrados pelo *middleware*. O *middleware*, após o registro, responde ao ESB se os dados foram registrados ou não com sucesso, que, por sua vez, retorna a resposta encriptada ao solicitante. É importante observar que o retângulo em vermelho demarca o processo executado no lado do cliente. Após isso, o processo é executado do lado do servidor.

O serviço público "alterar status da aplicação externa" permite que o status da aplicação seja alterado, possibilitando o *framework* inferir se a aplicação e seus serviços estão disponíveis ou não para acesso. O processo de atualização do status da aplicação deve ser iniciado em dois momentos. Primeiramente, quando a aplicação é inicializada,

deve-se imediatamente alterar seu status disponível. Segundo, quando a aplicação é finalizada, deve-se imediatamente alterar seu status para indisponível. A Figura 6.9 apresenta o diagrama de sequência para atualização de status da aplicação:

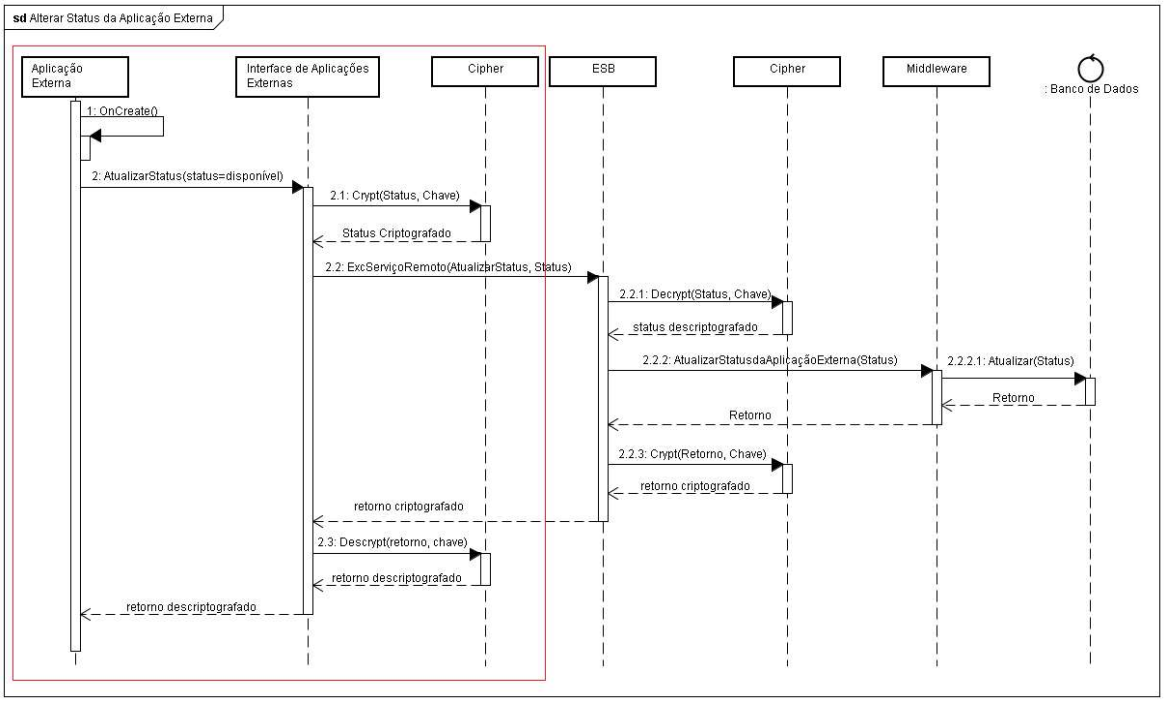


Figura 6.9. Processo de atualização do status da aplicação externa.

O processo de atualização do status da aplicação é iniciado no método *onCreate* da aplicação e enviado imediatamente ao *middleware* por meio do ESB, com a finalidade de atualização na base de dados. Após a alteração, o *middleware* informa a resposta ao ESB, o qual, por sua vez, repassa o resultado à aplicação, informando se o processo foi ou não realizado com sucesso. O diagrama de sequência (Figura 6.10) apresenta o processo de alteração do status da aplicação para disponível. Inclusive, o processo para atualização do status para indisponível é realizado da mesma forma, mas sendo iniciado por meio do método *onDestroy*.

O serviço público "consumir serviço" é responsável por permitir que os usuários, dispositivo e aplicações externas possam consumir os serviços gerenciados pelo *framework*, bem como os serviços públicos por esse providos. A Figura 6.10 apresenta o diagrama de sequência do processo de consumo de serviços:

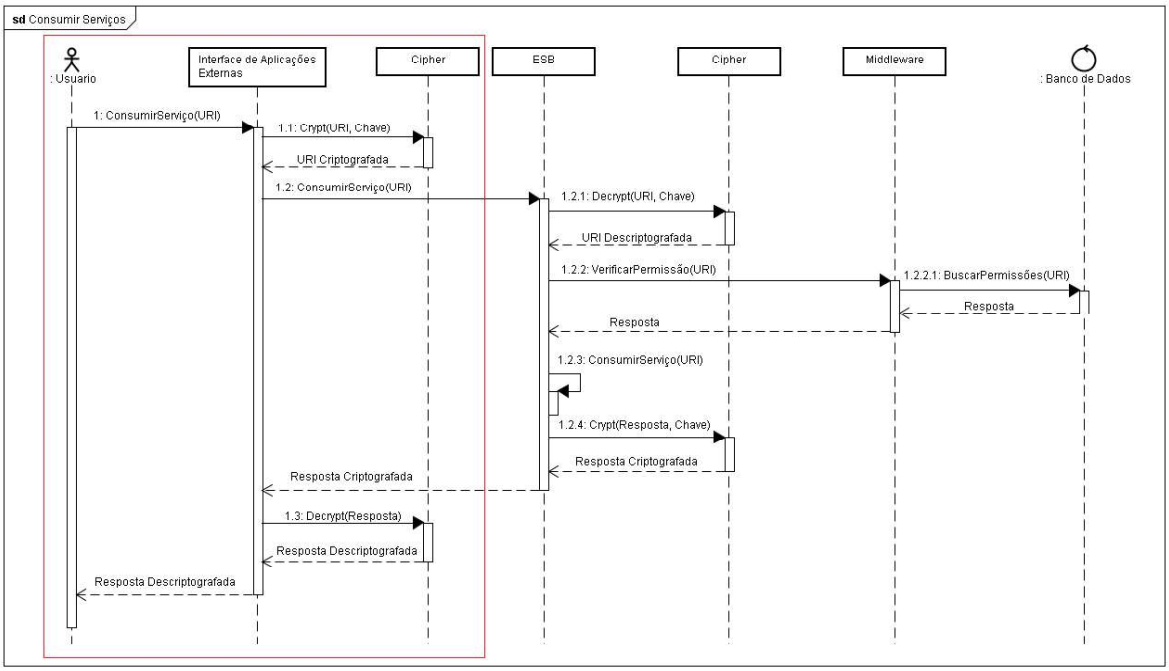


Figura 6.10. Processo de consumo de serviços.

O processo de consumo de serviço é iniciado quando o usuário envia uma requisição ao ESB para execução do serviço, passando como parâmetro a URI do serviço a ser executada. Esse, por sua vez, solicita ao *middleware* verificar se a aplicação solicitante possui permissão para execução do método solicitado. Caso haja a referida permissão, o ESB faz a execução do serviço e retorna o resultado da execução ao solicitante.

O serviço público "registrar dispositivo" é responsável por permitir que os usuários possam registrar novos dispositivos e serviços no *framework*. Para registro dos dispositivos, o usuário deve informar os seguintes dados: i) URI; ii) nome; iii) classe do dispositivo; iv) modelo; v) fabricante; vi) endereço MAC (se houver); vii) status; viii) localização; xi) data; e x) hora de registro. As informações de localização, data e hora de registro não são obrigatórias, uma vez que deverão ser preenchidas somente quando o objeto for efetivamente utilizado.

O serviço público "escutar evento disparado" possibilita que o usuário possa configurar, em sua aplicação, o momento quando um evento é disparado. Os dados do evento devem ser repassados para o *middleware* para armazenamento e análise, a fim de verificar se existe algum modelo de comportamento que possua em seu escopo o evento como um iniciador de fluxo. Com base nesse evento, definem-se quais modelos de comportamento deverão ser executados. A Figura 6.11 apresenta o diagrama de sequência do processo escutar evento disparado:

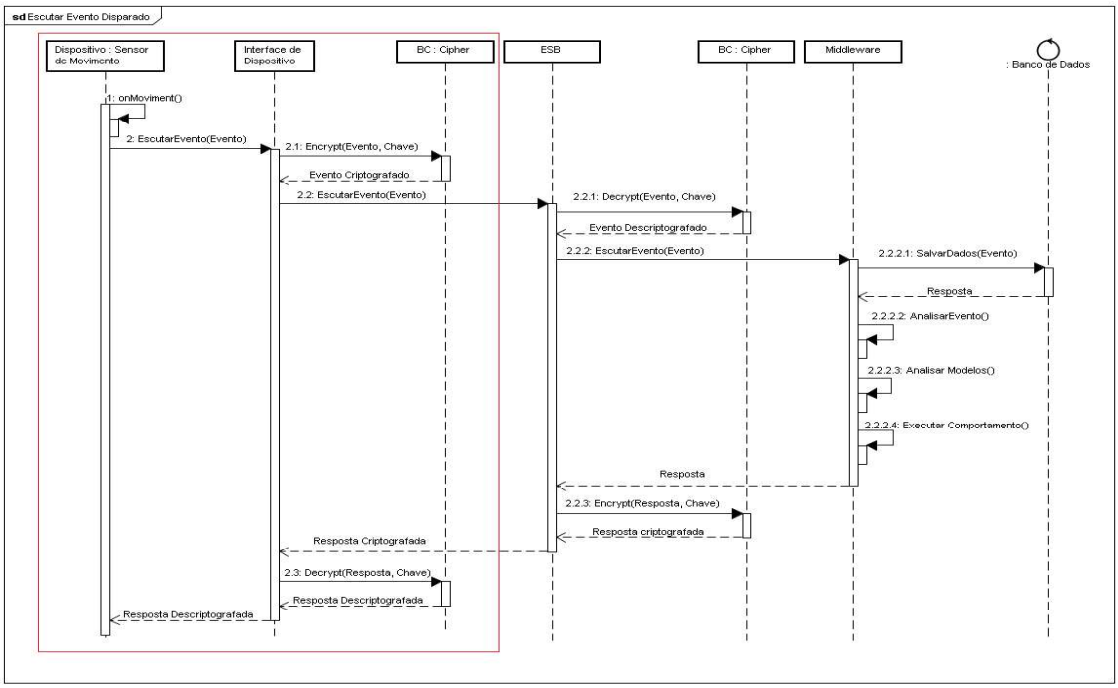


Figura 6.11. Processo Escutar Evento Disparado.

No exemplo apresentado pela Figura 5.13, o processo “escutar evento disparado” é iniciado quando o evento *onMoviment* do sensor é disparado; em seguida, os dados do evento são passados por parâmetros para o serviço Escutar Evento, são eles: i) a URI do dispositivo que disparou o evento; ii) URI do evento disparado; iii) nome do evento; iv) os dados gerados pelo evento; e v) data e hora que o evento ocorreu. Após isso, os dados são armazenados e analisados e os modelos de comportamento executados. Por fim, a resposta dos processamentos executados pelo middleware retorna ao ESB e repassada o dispositivo para controle.

O serviço público “receber mensagens” foi criado para possibilitar que aplicações externas recebam mensagens encaminhadas pelo ESB e/ou pelo *middleware*, a fim de exibir essas mensagens por meio de uma interface gráfica ao usuário ou realizar tratamentos internos, no caso de mensagens de erro interno do *framework*. É importante ressaltar que somente aplicações externas fazem uso de tal funcionalidade.

O serviço público “buscar catalogo de serviços” é responsável buscar, na base de serviços do *framework*, todos os serviços permitidos a um usuário, dispositivo e/ou aplicação externa. Para que uma aplicação externa tome conhecimento dos serviços públicos disponíveis, deve solicitar ao *middleware* um catálogo com todos os serviços disponíveis para acesso. Somente após o envio do catálogo, tais aplicações poderão

solicitar o consumo desses serviços. Além dos serviços públicos, é possível solicitar permissão de acesso a serviços privados, desde que esses sejam previamente aprovados pelo administrador do *framework*. A Figura 6.12 apresenta o diagrama de sequência do processo de busca do catalogo de serviços:

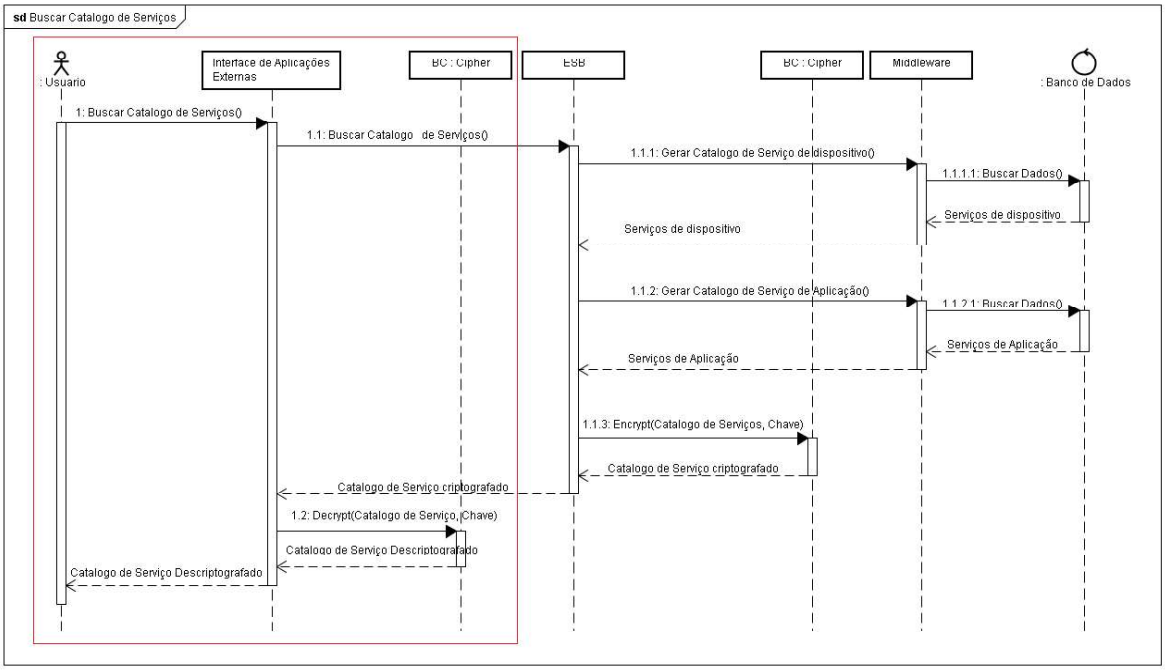


Figura 6.12. Processo buscar catálogo de serviços.

O processo “buscar catalogo de serviço” é iniciado após a solicitação de busca por um dispositivo ou aplicação externa ao *middleware*, que, por sua vez, busca todos os serviços públicos disponíveis para acesso na base de dados do *framework.* Após isso, o catálogo de serviços é retornado e entregue ao solicitante.

O serviço público “alterar status do dispositivo” mantém atualizado o status do dispositivo, informando ao *framework* se um dispositivo e seus serviços estão disponíveis ou não para acesso. O processo de atualização do status do dispositivo funciona de forma idêntica ao processo apresentado pela Figura 6.9. A seguir, serão previamente apresentados os serviços privados utilizados pelo ESB.

O serviço privado “reportar erro” registra todos os erros ocorridos no *framework* na base de histórico e envia uma mensagem para o usuário, que será recebida por meio do serviço “Receber Mensagens”. Todo erro ocorrido nos processamentos internos do *framework* utiliza o serviço reportar erro.

O serviço privado “enviar dados para monitoramento” é responsável por enviar para as aplicações externas todas as informações referentes às ações realizadas na execução dos modelos de comportamento. Dessa forma, é possível permitir que os usuários acompanhem, em tempo real, a execução dos modelos de comportamento, bem como os possíveis erros gerados.

Os serviços privados “criptografar e descriptografar” são utilizados pelo ESB para criptografar e descriptografar requisições e mensagens enviadas entre os usuários, ESB e middleware, conforme apresentado, anteriormente, nos diagramas de sequência.

6.2.3. A Interface de Aplicações Externas

Para que aplicações externas sejam vinculadas ao *framework Êxodo*, foi criada uma interface para possibilitar que essas aplicações venham prover e consumir serviços. Para cumprir esse objetivo, optou-se, no seu desenvolvimento, por fazer uso do padrão de projeto estrutural *Adapter* [Freeman 2007], o qual possibilita as aplicações, cujas interfaces possuam assinatura diferente as do ESB, sejam convertidas e integradas ao *framework* por meio de um adaptador. A Figura 6.13 apresenta o diagrama de classe para integração de aplicações externas ao *framework Êxodo*:

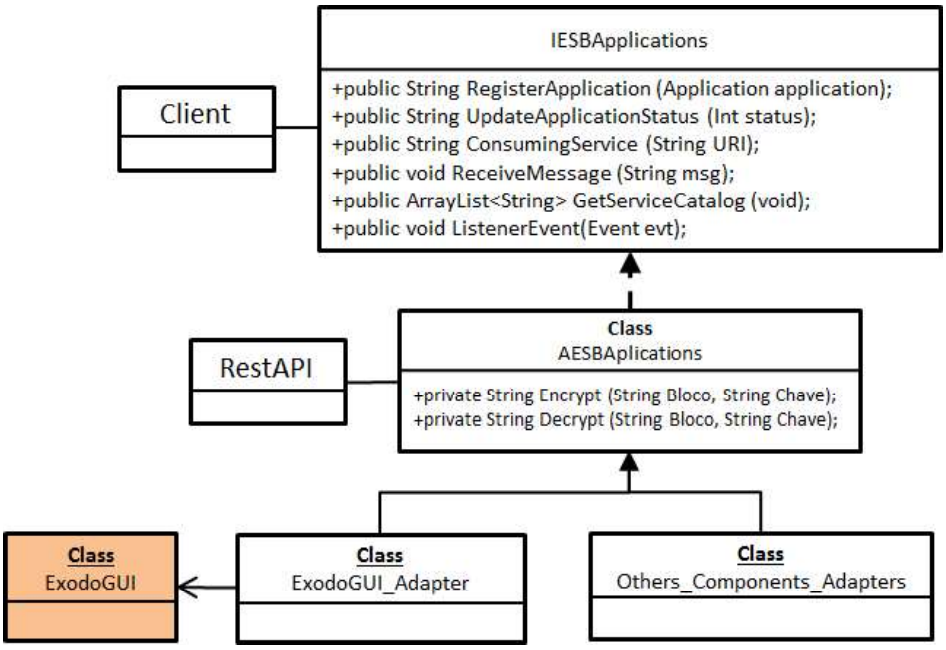


Figura 6.13. Diagrama de classe para integração de aplicações externas ao *framework*.

A interface *IESBApplications* contém a assinatura dos principais serviços públicos utilizados pelo ESB na comunicação com o *middleware*, tais como: registrar aplicações,

atualizar status da aplicação, consumir serviços, receber mensagens, buscar catálogo de serviços e ouvir eventos disparados.

Para que um usuário especialista venha incorporar uma nova aplicação ao *framework*, é necessário apenas ele criar uma classe adaptadora de forma que estenda a classe abstrata *AESBApplications*, cabendo apenas a responsabilidade de definir os serviços a serem providos e adaptado pela interface. Além disso, todo serviço provido pela aplicação externa que se deseje compartilhar deve, antes, ser registrado no *framework* por intermédio do serviço "registrar aplicações externas".

A classe abstrata *AESBApplications* dispõe de dois serviços locais: i) o *Encrypt*, responsável por criptografar os dados e solicitações de serviços, antes que sejam enviadas ao ESB, impedindo que essas informações sejam acessadas e vistas por aplicações e/ou pessoas não autorizadas; e ii) o *Descript*, responsável por descriptografar as informações recebidas, para se poder interpretar corretamente as informações desejadas.

Para integração de aplicações externas ao *framework*, o único requisito é o fato de a classe de interface, ou classe adaptadora, ser implementada utilizando a linguagem de programação *Java* (linguagem base utilizada no desenvolvimento do *Êxodo*). Contudo, isso não significa que aquelas soluções que utilizam de diferentes linguagens de programação não possam ser integradas ao *Êxodo*. Diferentes abordagens podem ser utilizadas para permitir essa integração, tais como: o uso do padrão JNI (*Java Notation Interface*), que possibilita que códigos executados em JVM chamem ou sejam chamados por bibliotecas escritas em outras linguagens, como C, C++ e/ou *Assemble*. Outra solução de fácil implementação é o uso de *Web Services*, que permitiria a integração de aplicações, independentemente da linguagem de programação utilizada.

6.2.4. A Interface de Dispositivos

Atualmente, a maioria das soluções existentes para IoT, conforme em Ebbits [2011], Zhang e Hansen [2008] e De Souza [2008], concentra esforços em manter todos os seus principais serviços sob responsabilidade de uma única camada: a camada de *middleware*. No *Êxodo*, optou-se por utilizar uma abordagem diferente para prover maior facilidade ao usuário especialista, quanto a futuras modificações e/ou adaptações em relação a que a solução possa sofrer no futuro. Nesse contexto, o barramento ESB dispõe de uma interface com os principais serviços necessários, para que um dispositivo do mundo real se integre

ao *framework Êxodo*, no sentido de prover novos serviços. A Figura 6.14 apresenta o esquema de classe para inclusão de dispositivos:

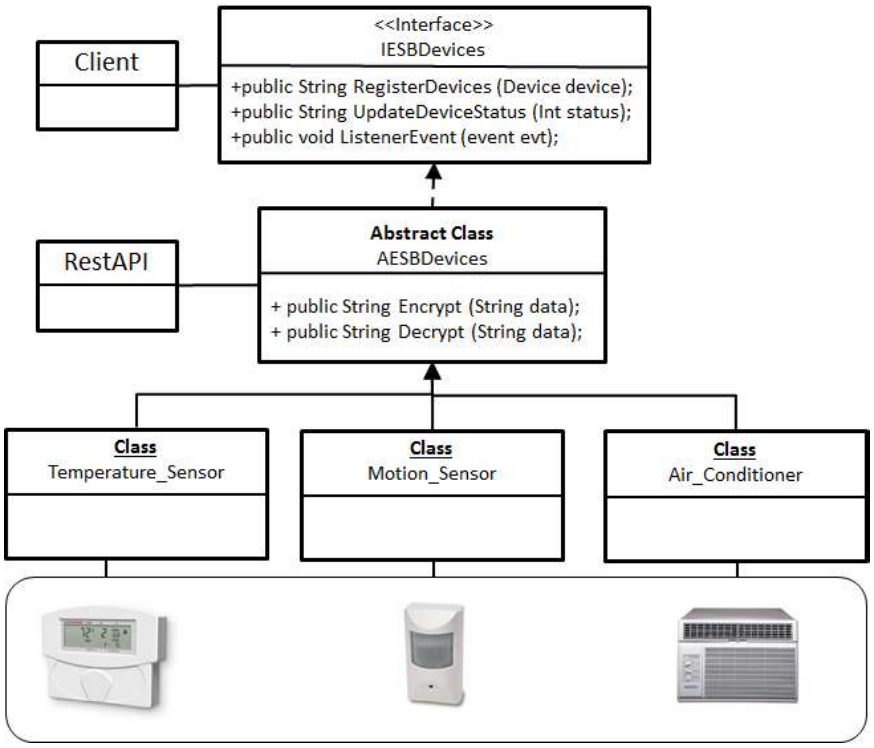


Figura 6.14. Esquema de classe para inclusão de dispositivos ao *framework*.

A interface *IESBDevices* é composta apenas por três serviços principais: registrar dispositivos, alterar status do dispositivo e escutar eventos. Diferente da interface de aplicações externas, que permite que aplicações externas possam tanto prover quanto consumir serviços, a interface de dispositivos permite apenas que se possam prover serviços. Contudo, caso seja necessário que os dispositivos venham consumir serviços, basta apenas o usuário especialista alterar a assinatura da interface, adicionando os serviços que habilitam tais funcionalidades.

A classe abstrata *AESBDevices* é responsável por implementar não só todos os serviços existentes na interface *IESBDevices*, mas também os serviços para criptografar e descriptografar mensagens, além dos fornecidos pelos dispositivos que deverão trafegar entre os componentes do *framework*. Por questões de segurança, todo dispositivo incorporado ao framework deve ser obrigatoriamente registrado por meio do serviço "registrar dispositivos" e, em seguida, ser aprovado por um usuário administrador, evitando, assim, que dispositivos desconhecidos tenham acesso a serviços não autorizados pelo usuário.

6.2.5. A Arquitetura da Aplicação Externa *Êxodo GUI*

Atualmente, a maioria das soluções de *middleware* existentes para IoT são desprovidas de interfaces gráficas capazes de incluir usuários não especialistas no processo criativo e gerencial de ambientes para IoT.

Em Warriach et al. [2011], os autores apresentam uma solução provida de interfaces gráficas desenvolvidas apenas para possibilitar o acesso dos usuários aos dispositivos e serviços, não no sentido de se criarem conexões lógicas entre eles para definição de comportamentos ou, até mesmo, para modificação dos comportamentos pré-definidos pela solução.

Já em Song, Cárdenas e Masuoka [2010], apresenta-se uma solução que possibilita a inclusão do usuário não especialista no processo criativo de ambientes inteligentes por meio de interfaces gráficas e modelos, o que possibilita a criação de comportamentos de forma sequencial. Contudo, a solução não permite a criação de regras de negócios complexas, como, por exemplo, estrutura de decisão, repetição e/ou operadores lógicos *AND* e *OR*.

Tendo como base as limitações supracitadas, o principal objetivo da aplicação externa *Êxodo GUI*, ou simplesmente *GUI*, é permitir a inclusão de usuários não especialistas no processo criativo de comportamento de ambientes para IoT. Para isso, duas soluções foram propostas: i) um modelo conceitual de fácil entendimento para possibilitar que os usuários criem diferentes comportamentos; e ii) o desenvolvimento de uma aplicação gráfica para permitir a usuários manipularem os dispositivos e serviços existentes no ambiente.

Para facilitar a manipulação dos objetos por parte dos usuários, a GUI emprega a técnica de arrastar e soltar, para que os usuários possam adicionar, excluir e manipular os objetos do modelo. Foram também definidas telas para manipulação e configuração das propriedades e eventos dos objetos. A Figura 6.15 apresenta a arquitetura da aplicação gráfica *Êxodo GUI* e seus principais componentes:

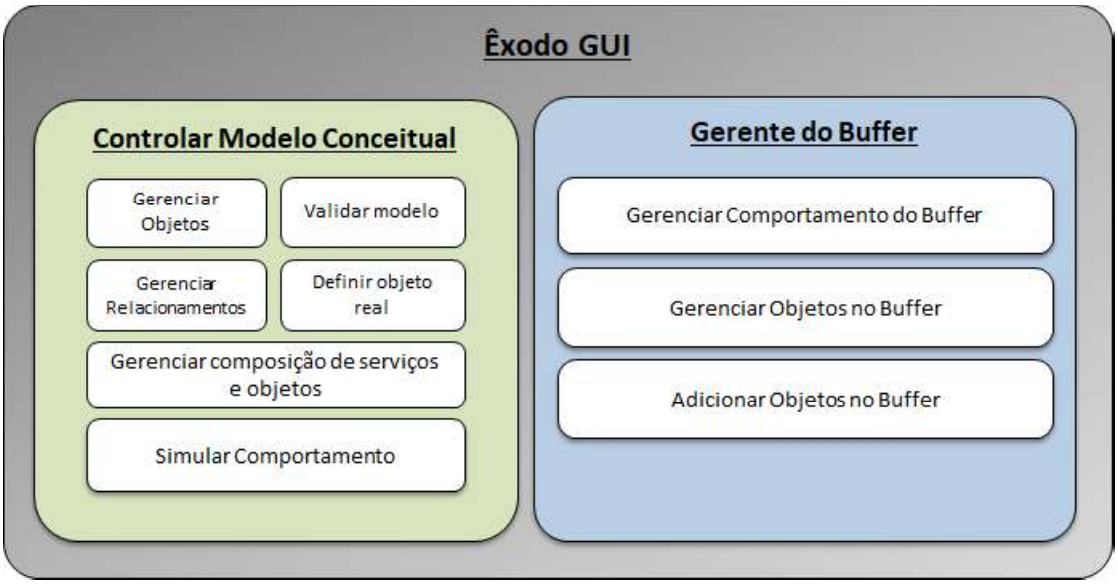


Figura 6.15. Arquitetura da solução Êxodo GUI e seus componentes principais.

A arquitetura da aplicação externa *Êxodo GUI* foi dividida em dois componentes principais: "controlar modelo conceitual" e "gerente de *buffer*".

O primeiro é o responsável por gerenciar todos os objetos e comportamentos que compõem o modelo conceitual BDM4IoT. Os principais serviços fornecidos por esse componente são: i) gerenciar objetos; ii) validar modelo; iii) gerenciar relacionamentos; iv) definir objeto real; v) gerenciar composição de serviços e objetos; e vi) simular comportamento. Tudo de acordo com o seguinte:

- Serviço "gerenciar objetos" – responsável por gerenciar e definir os parâmetros que descrevem os objetos do modelo conceitual BDM4IoT, bem como a definição do comportamento de cada objeto;
- Serviço "validar modelo" – responsável por realizar a validação dos modelos criados, de acordo com a sintaxe definida na linguagem BDML;
- Serviço "gerenciar relacionamentos" – responsável por controlar e gerenciar a tabela com todas as relações existentes entre os objetos de um modelo;
- Serviço "definir objeto real" – responsável por gerenciar e controlar a relação existente entre um objeto real e um virtual, ressaltando-se que essa ação possibilita que todas as outras ações realizadas pelo modelo de comportamento sejam refletidas diretamente no ambiente real;
- Serviço "gerenciar composição de serviços e objetos" – responsável por gerenciar a tabela de composição de objetos e serviços realizados pelo o usuário, sendo uma ação

importantíssima, porquanto ajuda a tornar o modelo menos confuso e complexo. Por exemplo, imagine que um usuário esteja modelando o comportamento de uma linha de produção que possui um maquinário muito complexo, composto por diversos sensores. Além de suas funcionalidades básicas, seria difícil para o usuário gerenciar todos os sensores desse maquinário. Portanto, para facilitar o gerenciamento de todos os dispositivos, uma boa saída seria agrupar tais objetos em um único objeto mais simples de se manipular e gerenciar, mantendo a simplicidade e legibilidade do modelo de comportamento;

- Serviço "simular comportamento" – possibilita a execução simulada de eventos do mundo real, a fim de verificar se o modelo de comportamento irá se comportar, conforme definição do usuário. É importante ressaltar que a simulação poderá ser executada por um período tempo que o usuário definir ou, ainda, podendo ser executada e interrompida por ele manualmente. No final da execução, um relatório deverá ser gerado, informando ao usuário como o ambiente se comportou enquanto a simulação estava sendo executada.

Já o componente "gerente do *buffer*" é o responsável por gerenciar e controlar todas as funcionalidades diretamente relacionadas à exibição, manipulação e configuração dos objetos criados pelo usuário, para compor um modelo de comportamento. Os serviços fornecidos por esse componente são: i) gerenciar comportamento do *buffer*; ii) gerenciar objetos no *buffer*; e iii) adicionar objetos no *buffer*.

O serviço "gerenciar comportamento do *buffer*" controla e mantém o buffer e a interface principal de acesso do usuário constantemente sincronizadas. Quaisquer alterações realizadas no *buffer* devem ser refletidas diretamente na interface principal de acesso. Além disso, esse serviço controla e antecipa todos os processamentos relacionados aos objetos de modelo, antes de serem exibidos, assim se evita que o usuário perceba as sobreposições que ocorrem na imagem dos objetos, a partir do momento em que sofrem alguma alteração, como: redimensionar, arrastar ou criar relações entre objetos. A Figura 6.16 apresenta o processo de atualização entre o *buffer* e a interface principal:

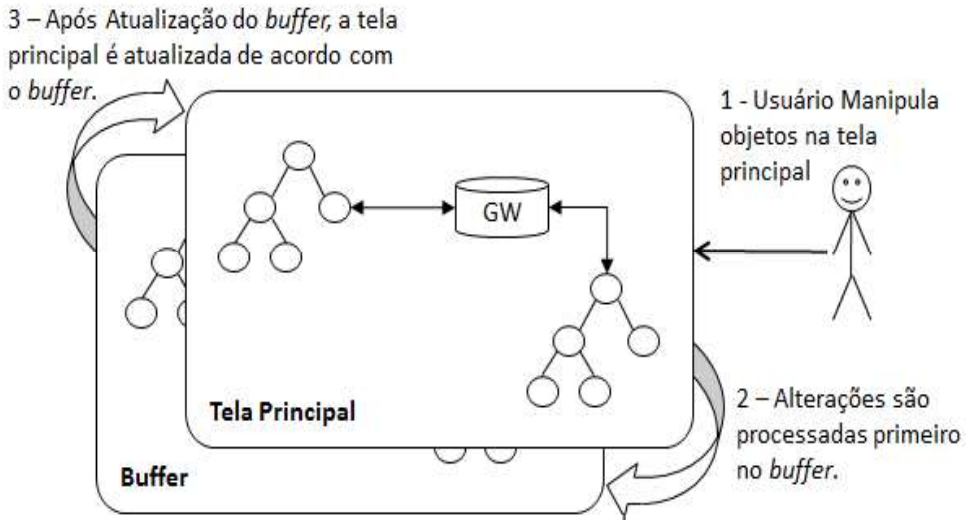


Figura 6.16. Processo de atualização do *buffer* e interface principal.

O serviço "gerenciar objetos no *buffer*" é responsável por controlar todos os objetos no próprio *buffer* presentes, controlando as informações importantes, como atualização as de coordenadas cartesianas, dimensões e exclusão de objetos. Se um objeto sofrer modificações, como ser movimentado, redimensionado, excluído ou sofrer alterações em suas características, é de sua responsabilidade o serviço manter tais informações constantemente atualizadas.

O serviço "adicionar objetos no *buffer*" é responsável por criar fisicamente a imagem dos objetos em uma pequena área de transferência, enviando os objetos ao *buffer* somente quando preparados. Esse serviço é responsável por controlar todas as diferentes formas que os objetos do modelo conceitual podem assumir.

A seção a seguir tem por objeto apresentar a interface gráfica do *Êxodo GUI* e suas telas principais.

6.2.6. A Interface Gráfica da Aplicação Externa *Êxodo GUI*

Para o desenvolvimento da interface gráfica *Êxodo GUI* foi utilizada a linguagem de programação Java, bem como a biblioteca Java 2D para elaboração do modelo conceitual proposto. Além disso, a interface gráfica tem um conjunto de classes e métodos bem definidos, de modo que possibilitam aos usuários especialistas fazerem adaptações na solução, como, por exemplo, adicionar novos objetos ao modelo e/ou, até mesmo, criar novos serviços e funcionalidades à solução. A Figura 6.17 apresenta a tela principal da ferramenta gráfica *Êxodo GUI*:

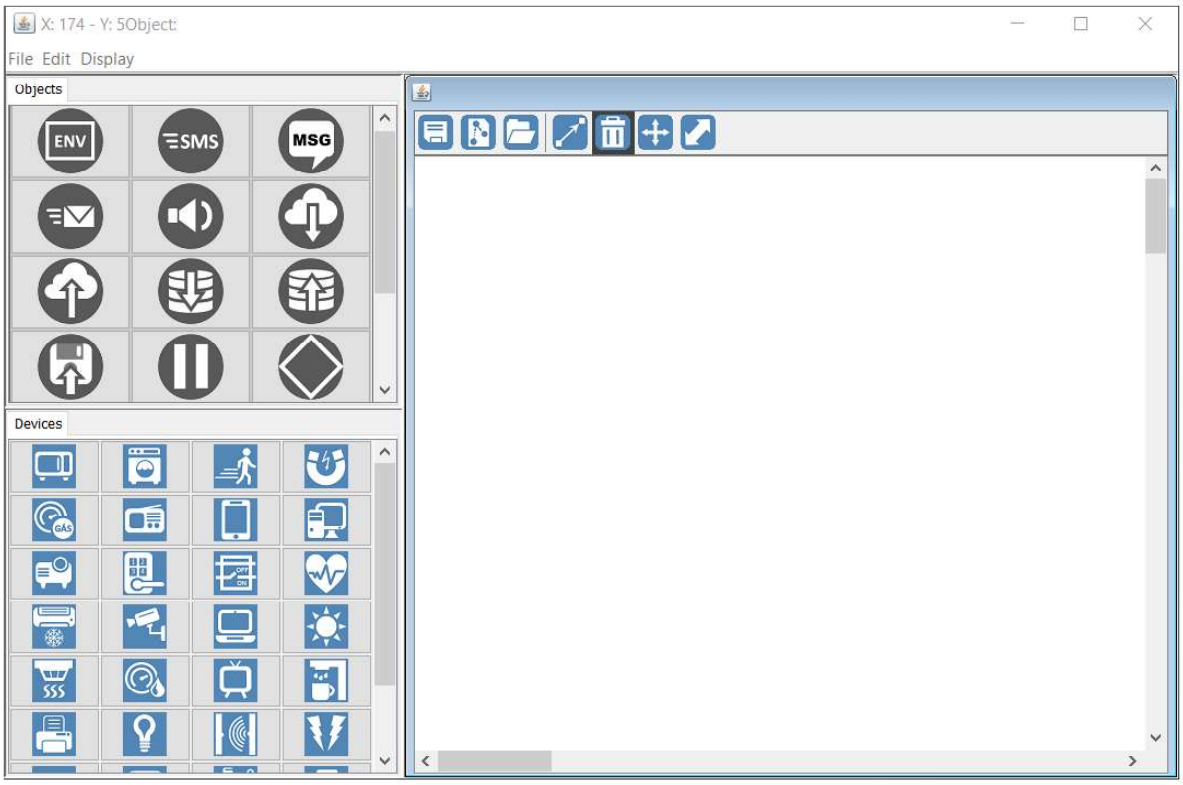


Figura 6.17. Tela Principal da Interface *Êxodo GUI*.

A tela principal da ferramenta gráfica é dividida em três partes principais. A área de objetos, localizada no lado superior esquerdo da tela, mantém os componentes que auxiliam o usuário na modelagem e definição das regras de negócio do comportamento.

No lado inferior esquerdo, encontra-se a área de componentes de dispositivos, que corresponde à lista de sensores e eletrodomésticos, comumente encontrados em ambientes domésticos ou comerciais. Contudo, é importante ressaltar que a ferramenta não se restringe apenas a esses dispositivos, pois o *Êxodo GUI* permite que o usuário possa cadastrar diferentes objetos para atender a necessidades específicas do ambiente.

No lado direito, encontra-se a área de manipulação gráfica dos objetos do modelo, usada pelo usuário para adicionar os dispositivos e objetos auxiliares que irão compor o comportamento a ser modelado. Essa área também permite que os usuários possam editar e configurar as propriedades e características de cada objeto e dispositivo. As funcionalidades localizadas na parte superior da tela de manipulação são destinadas a possibilitar que os objetos possam ser excluídos, movimentados, redimensionados, interligados, além de permitir que salvem seus modelos, testem e abram um modelo salvo em formato BDML.

Para incluir um objeto na tela de manipulação, primeiramente o usuário deverá selecionar o objeto desejado e clicar em qualquer parte da tela para definir a posição inicial do objeto. Após isso, o usuário poderá acessar as propriedades do objeto, configurando-o de acordo com as suas necessidades, como, por exemplo, definir o tipo de objeto (início de fluxo, intermediário ou fim de fluxo), e inserir informações, como: modelo, fabricante, nome, entre outras informações importantes, conforme pode-se observar na Figura 6.18.

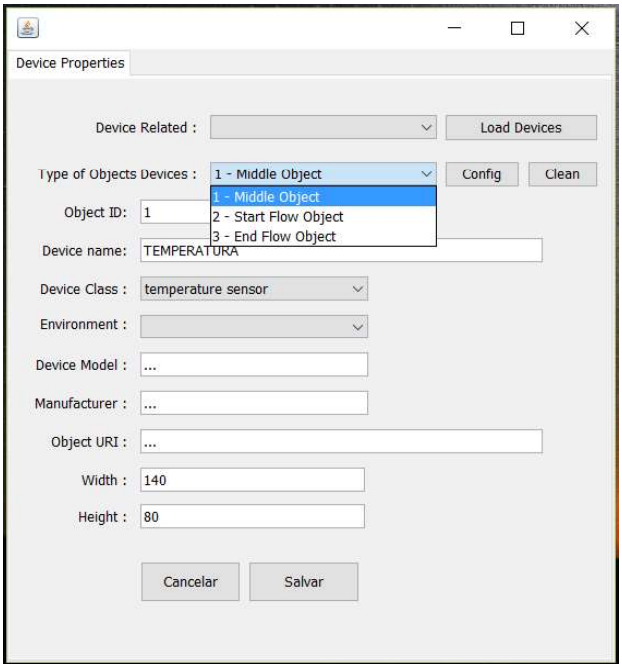


Figura 6.18. Tela de configuração das propriedades de um objeto.

A tela de propriedades do objeto possui três botões: i) "*load devices*", responsável por fazer a solicitação dos dispositivos cadastrados e gerenciados pelo *framework Êxodo*, ou seja, dá início ao processo de busca do catálogo de dispositivos e serviços. Somente após a busca do catálogo, o usuário poderá vincular o dispositivo real ao objeto virtual em configuração; ii) "*Config*", o qual possibilita o acesso à tela de configuração dos tipos de disparo que um objeto pode assumir; e iii) "*Clean*", que pode tornar o objeto nulo, ou seja, limpa todas as propriedades de disparo já configuradas, sendo essa funcionalidade utilizada somente quando o usuário deseja que um objeto retorne a seu estado inicial, para que seja novamente configurado. A Figura 6.19 apresenta a tela de configuração dos tipos de disparo de um objeto, apresentada quando o botão "*Config*" é acionado:

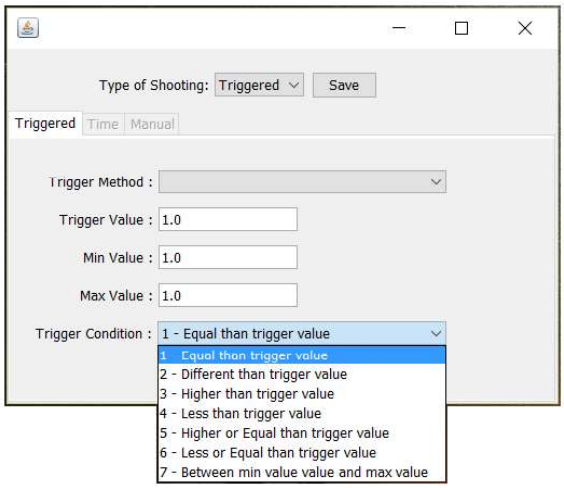


Figura 6.19. Tela de configuração de tipos de disparo dos objetos de dispositivo.

As informações a serem preenchidas pelo usuário na tela de configuração de disparo dependem do tipo de disparo selecionado. Para os do tipo engatilhado (*triggered*), os usuários deverão informar quatro parâmetros: i) o método a ser disparado quando o gatilho for desarmado; ii) valor do gatilho que será utilizado como base para validação das condições do gatilho; iii) o intervalo mínimo e máximo a ser retornado pelo método e utilizado quando a condição de disparo *between* for utilizada; e iv) a condição em que o gatilho irá disparar, tais como: igual, diferente, maior, menor, maior ou igual, menor ou igual e entre o valor máximo e mínimo.

Para disparos do tipo temporizado (*time*), os usuários deverão informar os seguintes parâmetros: i) o evento/método a ser executado; ii) a data em que o evento deverá ser executado; e iii) a hora em que o evento deverá ser executado.

Para disparos do tipo manual, os usuários deverão informar somente o evento/método que deve ser executado, uma vez que a sua execução está condicionada para o usuário executar de acordo com as suas necessidades. A seguir, a Figura 6.20 apresenta a tela de configuração dos objetos de regras de negócio.

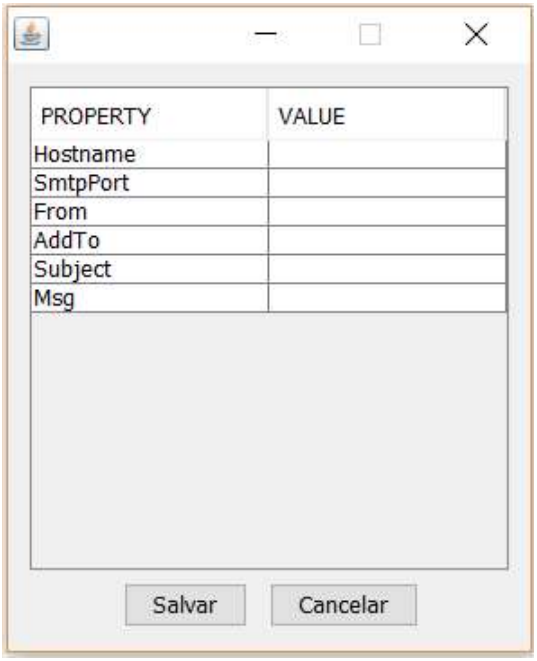


Figura 6.20. Tela de propriedades de objetos gerada dinamicamente pelo *Êxodo GUI.*

A tela de propriedade de objetos de regras de negócio é gerada dinamicamente de acordo com as propriedades definidas para cada objeto. Essa estratégia é importante, pois os usuários podem criar novos tipos de objetos e incorporá-los ao modelo, evitando assim que o usuário tenha que fazer uma tela para cada novo objeto definido. Neste caso, cabe ao usuário apenas mapear os parâmetros que definem as propriedades do objeto, que serão apresentadas na tela. A Figura 6.20 apresenta as propriedades de um objeto de regra de negócio do tipo *E-Mail*.

5.2.7. Tecnologias Utilizadas no Desenvolvimento do *Framework Êxodo*

Para elaboração e implementação do projeto do Êxodo, foram utilizadas diferentes tecnologias, frameworks e bibliotecas, com objetivo de prover uma solução mais flexível e extensível. Para o desenvolvimento da camada de *middleware* e do barramento ESB, foram utilizadas as seguintes tecnologias:

- **Tomcat 8** [Tomcat 2016] – Servidor de aplicação utilizado pelo *Êxodo* para levantamento dos serviços disponíveis pelas soluções, tornando-os disponíveis e acessíveis por meio da Internet;
- **Hibernate3** [Gonçalves 2007] – *Framework* utilizado pelo Êxodo para controle e persistência de dados;

- **DOM4J** [Dom4j 2016] **–** framework para leitura e criação de arquivos XML em aplicações Java. Como o Êxodo trabalha criando arquivos em formato BDML, a utilização do framework DOM4J é crucial para os processos de criação, processamento e execução dos arquivos em formato BDML;
- **Xerces** [Xerces 2010] **–** No Êxodo, o framework Xerces é o responsável pela validação dos arquivos BDML, gerados através de seu *XML Schema*;
- **Jersey** [Jersey 2016] **–** Framework utilizado pelo Êxodo para a implementação das especificações REST na arquitetura do Êxodo; e
- **Apache HTTPClient** [HTTPClient 2016] **–** Biblioteca utilizado pelo framework para invocação dos serviços Rest, por meio do protocolo HTTP.

6.3. Considerações Finais

Em vista do que foi apresentado neste capítulo, buscou-se ilustrar o *framework Êxodo* como uma solução projetada, com foco tanto no usuário especialista quanto não especialista. As principais vantagens oferecidas pelo *framework* aos usuários especialistas são a flexibilidade e a extensibilidade, alcançadas por meio do uso de uma arquitetura com base nos tipos de serviços que permite o compartilhamento e o consumo de serviços pelas novas aplicações e/ou dispositivos. Além disso, com o uso de padrões de projeto, foi possível criar interfaces e classes que podem ser facilmente estendidas, possibilitando que, mesmo com poucas linhas de código, o usuário venha integrar soluções e dispositivos ao *framework*, por meio do barramento ESB.

Para usuários não especialistas, o *framework Êxodo* propõe duas abordagens: a primeira corresponde à definição de um modelo conceitual de fácil compreensão, a fim de permitir que usuários com pouco ou quase nenhum conhecimento em computação criem modelos de comportamento; e a segunda abordagem é justamente a implementação de uma solução gráfica denominada *Êxodo GUI*, projetada para permitir aos usuários não especialistas manipularem efetivamente o modelo conceitual proposto, com a finalidade de criar os comportamentos que devem ser executados no ambiente real.

Capítulo 7 -
Experimentos e Avaliações

Nos capítulos 4 e 5, foram apresentados, respectivamente, o modelo conceitual BDM4IoT e a linguagem BDML, que são, evidentemente, as principais soluções da presente tese, no sentido de se criar um modelo conceitual extensível, podendo adaptá-lo para atender a qualquer domínio da IoT. Tais soluções possibilitam que usuários não especialistas possam, por meio de um conjunto de objetos bem definidos, planejar e modelar todo comportamento de um ambiente para IoT, de acordo com as suas necessidades, com a possibilidade de poder modificar, a qualquer momento, os comportamentos já definidos.

Já no capítulo 6, foi apresentado o *framework Êxodo*, sua arquitetura e a solução gráfica, denominada *Êxodo GUI*, que se destaca por ser a principal solução do *framework*, a qual foi desenvolvida para se permitir não somente a implementação real do modelo conceitual BDM4IoT, como também a manipulação do modelo pelos usuários.

Dessa forma, para avaliar a eficiência das principais propostas desta tese, depreende-se que, para inclusão do usuário não especialista no processo criativo e gerencial de ambientes para IoT (BDM4IoT e a Interface gráfica *Êxodo GUI*), um conjunto extenso de experimentos foram elaborados e executados com duas classes de usuários: especialistas e não especialistas. Os resultados, ainda, foram comparados, a fim de se analisar a expertise entre as duas classes, particularmente quanto ao uso do modelo e da ferramenta proposta.

Em vista dessa breve inferência, procurou-se organizar este capítulo em conformidade ao que se segue, a saber: a seção 7.1 apresenta o protocolo experimental, onde será introduzido o perfil dos usuários e como os experimentos foram organizados e executados; em seguida, nas seções 7.2, 7.3 e 7.4, serão apresentadas as conduções e os resultados obtidos da primeira, segunda e terceira rodadas de experimentos, respectivamente; e, finalmente, na seção 7.5, serão apresentadas as considerações finais.

7.1. Protocolo Experimental

Para verificar e validar a eficiência das abordagens utilizadas na inclusão do usuário não especialista nos processos de modelagem de ambiente para IoT, foi elaborado um conjunto de experimentos com um grupo de 40 usuários de diferentes perfis, sendo 20 homens e 20 mulheres, com idade variando entre 26 e 55 anos, com diferentes níveis acadêmicos, sendo 20% com nível médio, 47,5% com nível superior e 32,5% com especialização e/ou pós-graduação.

Antes de iniciar os experimentos, os usuários foram divididos de acordo com seu nível de conhecimento em computação, especialistas e não especialistas. Especialista é todo usuário que possui formação em alguma área da computação, sendo essa de nível médio, superior, especialização ou pós-graduação. Não especialistas são usuários que fazem uso básico da computação, como acesso à Internet, utilização de ferramentas de escritório, entre outros. Sendo assim, foi realizada a seguinte divisão: 45% dos usuários foram classificados como usuários especialistas; e 55% classificados como não especialistas.

Os grupos de usuários foram fisicamente separados durante os experimentos, a fim de não haver influência de outros usuários com maior nível de conhecimento (especialistas) para com os de menor nível intelectual no assunto (não especialistas). Em seguida, para alinhar o conhecimento dos usuários quanto aos principais conceitos relacionados a IoT, foi apresentada uma palestra intitulada: "*O futuro da Internet e sua influência no cotidiano das pessoas*", em que se ilustraram conceitos, como dispositivos inteligentes, Internet das Coisas, aplicabilidades no ambiente doméstico e ferramentas existentes. Ainda, foi explicado aos usuários como os experimentos serão conduzidos.

Para avaliar o modelo conceitual BDM4IoT e a ferramenta Êxodo GUI, foram elaboradas três séries de experimentos. A primeira série tendo como objetivo avaliar os objetos e os modelos de comportamento do BDM4IoT quanto à expressividade e a facilidade de leitura e interpretação, respectivamente.

A segunda, avaliar a facilidade de aprendizado do BDM4IoT para elaboração de modelos de comportamento e a avaliação da ferramenta *Êxodo GUI* quanto às seguintes características: i) intuitividade; ii) manipulabilidade; iii) configurabilidade; e iv) legibilidade.

A terceira, comparar, imparcialmente, a opinião dos usuários relacionadas às ferramentas *Êxodo GUI*, ASPIRE *Editor* e OpenHAB, considerando-se as características de intuitividade, manipulabilidade, configurabilidade e legibilidade. Por fim, compara-se o modelo BDM4IoT ao modelo BPMN, quanto à capacidade de representação de dispositivos e regras de negócio em modelos de comportamento para IoT.

Para execução dos experimentos, foram utilizados questionários, que, em princípio, é um método empírico por meio do qual se faz uso de um conjunto de perguntas sobre um determinado tópico, a fim de se medir a opinião/satisfação dos usuários. Para maiores detalhes sobre os questionários elaborados, consultar os Apêndices A, B, C e D.

7.2. Resultados

A seguir as seções 7.2.1 e 7.2.2 apresentam os resultados obtidos quanto a expressividade dos objetos e a facilidade de leitura e interpretação dos modelos. A seção 7.2.3 apresenta os resultados obtidos quanto a facilidade de aprendizado do modelo BDM4IoT e a avaliação da ferramenta *Êxodo GUI*. A seção 7.2.4 apresenta os resultados obtidos quanto a comparações realizadas com a ferramenta *Êxodo GUI* e o modelo BDM4IoT com soluções similares e a seção 7.2.5 apresenta as possíveis ameaças a validade identificadas para os experimentos realizados.

É importante destacar que, em resultados que foi necessário a aplicação de testes estatísticos para prova de hipótese, foram realizados os passos a seguir:

1. Verificar a normalidade da distribuição com o teste Shapiro Wilk. A fórmula deste teste é dada por, $W = \frac{b^2}{\sum_{i=1}^{n}(x(i)-\overline{x})^2}$.
2. Se a distribuição for normal, o teste de hipótese a ser utilizado é o *t-student*, caso contrário, o teste a ser aplicado é o não-paramétrico *Mann-Whitney*.

7.2.1. Expressividade

A primeira rodada de experimentos consistiu em avaliar as seguintes características dos objetos da BDM4IoT e dos modelos de comportamento: expressividade e facilidade de leitura e interpretação dos modelos de comportamento.

A expressividade dos objetos do BDM4IoT consiste em verificar se os ícones/imagens definidas para os objetos expressam a sua real finalidade, o que facilita os usuários com pouco conhecimento em computação e/ou ferramentas dessa natureza, no tocante a deduzir, empiricamente, a real finalidade dos objetos apresentados.

Para execução da primeira rodada, foi elaborado um questionário contendo 7 atividades (Apêndice A), sendo as quatro primeiras destinadas à avaliação dos objetos pertencentes ao modelo BDM4IoT. Por fim, os dados foram extraídos e agrupados, conforme apresentação das taxas de acertos obtidas, quanto à avaliação dos objetos, destacado na Figura 7.1.

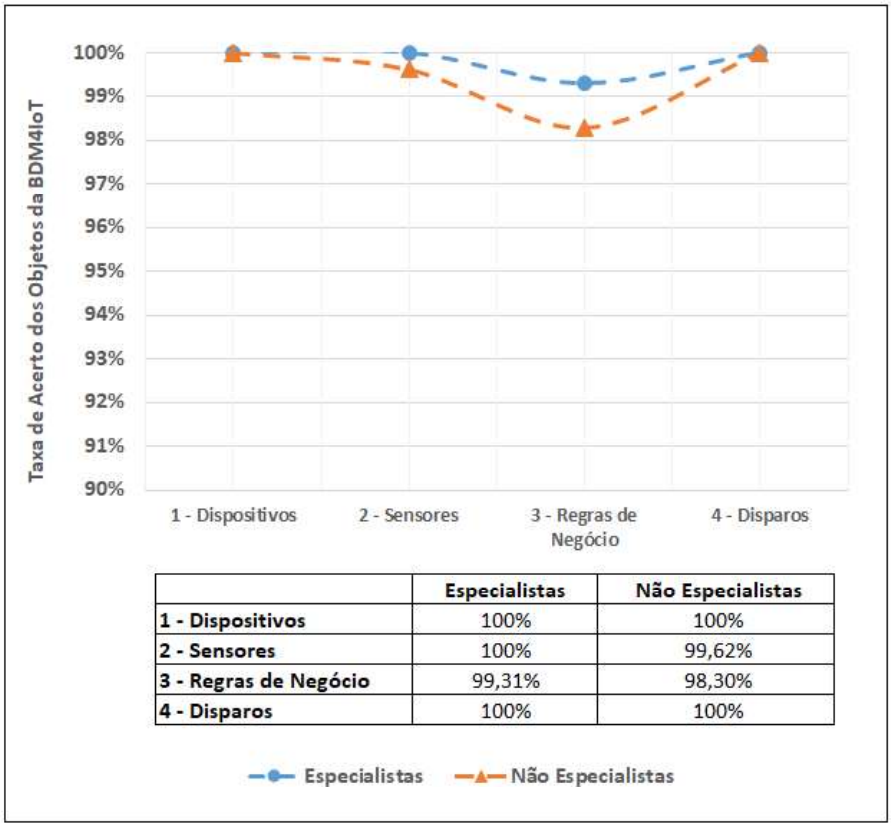
É importante ressaltar que a taxa de acerto é encontrada utilizando-se a fórmula a seguir:

$$TxAc = \left\{\frac{(TPOCS * TUsrP)}{(\sum_{i\leq TUsrP}^{i=1} OSC)}\right\} * 100$$

Analisando a expressão acima, de modo mais didático, ocorre que: *TPOCS* é o total previsto de objetos corretamente selecionados, que é multiplicado pelo total de usuários participantes (*TUsrP*) e dividido pela somada de todos os objetos selecionados corretamente (*OSC*) de todos os usuários participantes; esse resultado multiplicado por 100 para se obter o percentual. Sendo assim, calculando-se a taxa de acerto de dispositivos para usuários especialistas, conclui-se:

$$TxAc = \left\{\frac{(18*18)}{(324)}\right\} * 100 = \left\{\frac{(324)}{(324)}\right\} * 100 = 1 * 100 = 100\%$$

De acordo com os resultados obtidos, é possível perceber que tanto usuários especialistas quanto não especialistas obtiveram taxas de acerto de 100% para dispositivos disparos. No entanto, os usuários não especialistas obtiveram uma taxa de acerto de 99,62% para sensores, que é um resultado bem interessante, considerando-se a pouca experiência e contato dos usuários não especialistas com esse tipo de dispositivo. Em suma, isso demonstra que tais objetos foram facilmente interpretados tanto por usuários especialistas quanto não especialistas.



	Especialistas	Não Especialistas
1 - Dispositivos	100%	100%
2 - Sensores	100%	99,62%
3 - Regras de Negócio	99,31%	98,30%
4 - Disparos	100%	100%

Figura 7.1. Percentual de Objetos Identificados Corretamente.

Por outro lado, as taxas de acerto dos objetos de regras de negócio foram de 99,31% para o grupo de usuários especialistas e de 98,30% para os usuários não especialistas. Sendo assim, considerando a falta de experiência dos usuários não especialistas e o pouco contato dos usuários especialistas com objetos para modelagem de comportamento de ambientes para IoT, pode-se concluir que, em sua grande maioria, os objetos de regras de negócio foram interpretados corretamente pelos usuários de ambas as classes.

Além disso, quando observado o gráfico comparativo entre as respostas dos usuários especialistas e não especialistas, é possível perceber que ambos os grupos obtiveram resultados aproximados, ou seja, usuários com pouco conhecimento em computação e com modelos dessa natureza tiveram resultados bem próximos aos de usuários com maior nível de conhecimento. Logo, para ambas as classes, os objetos e modelos do BDM4IoT foram facilmente identificados e interpretados.

7.2.2. Facilidade de Leitura e Interpretação

A facilidade de leitura e interpretação dos modelos de comportamento consiste em verificar se os usuários conseguem descobrir – apenas por observação – o tipo de comportamento que é executado. Essa característica é relevante, porque permite diferentes usuários manipularem o modelo, mesmo que não tenha participado de sua elaboração. Dessa forma, os modelos não ficam restritos a apenas um grupo de usuários.

A quinta atividade do formulário elaborado para a primeira série de experimentos (Apêndice A) foi destinada a avaliar os modelos de comportamento. Para tal, foram elaborados 4 cenários onde os usuários deveriam informar qual resposta descreve melhor os cenários apresentados. Os resultados obtidos demonstram que 100% dos usuários participantes, ou seja, 18 especialistas e 22 não especialistas, selecionaram corretamente todas as respostas que melhor descreviam os cenários apresentados. Sendo assim, é possível concluir que tanto usuários especialistas quanto não especialistas não tiveram dificuldades em ler e interpretar corretamente os modelos de comportamento apresentados.

Por fim, as questões 6 e 7 (Apêndice A) foram elaboradas para se obter a satisfação dos usuários quanto à expressividade dos objetos e à facilidade de leitura dos modelos de comportamento. Nesse caso, eles deveriam informar seu nível de satisfação de acordo com uma escala *Likert*, onde: 1) discordo totalmente; 2) discordo parcialmente; 3) não sei opinar; 4) concordo parcialmente; e 5) concordo totalmente. A Figura 7.2 apresenta os resultados obtidos quanto à expressividade dos objetos e a Figura 7.3 apresenta os resultados obtidos quanto a facilidade de leitura e interpretação.

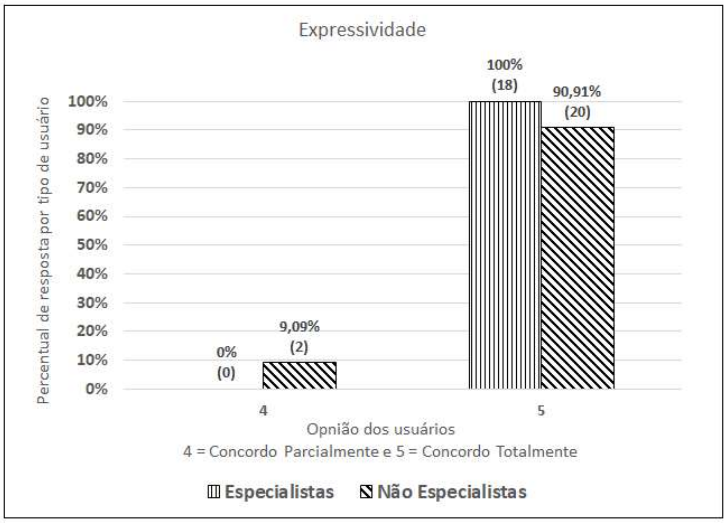


Figura 7.2. Satisfação dos usuários quanto à Expressividade dos objetos da BDM4IoT.

Com relação à satisfação dos usuários quanto à expressividade dos objetos do modelo BDM4IoT, 100% dos usuários especialistas concordam totalmente (5) que os objetos do modelo expressam sua real finalidade. Usuários não especialistas, 90,91% concordam totalmente (5) e 9,09% concordam parcialmente (4) com a expressividade dos objetos do BDM4IoT. Dessa forma, é possível concluir que tanto os usuários especialistas quanto não especialistas concordam que os objetos do modelo BDM4IoT são expressivos, isto é, permitindo que os usuários possam, somente pelas imagens definidas para os objetos, saber exatamente sua principal funcionalidade.

Com relação à satisfação dos usuários quanto à facilidade de leitura e interpretação, foram obtidos os seguintes resultados: 100% dos usuários especialistas afirmam concordar totalmente (5) que os modelos de comportamento e as regras de negócio são fáceis de compreender e interpretar; usuários não especialistas, 95,45% concordam totalmente (5); e 4.55% concordam parcialmente. Com isso, pode-se inferir que os modelos foram lidos e interpretados facilmente, tanto por usuários especialistas quanto não especialistas.

Figura 7.3. Satisfação dos usuários quanto à facilidade de leitura e interpretação dos modelos de comportamento.

Para verificar se, estatisticamente, há diferença entre a opinião dos usuários especialistas e não especialistas quanto à expressividade dos objetos da BDM4IoT e, também, a facilidade de leitura e interpretação dos modelos de comportamento, foram elaboradas as seguintes hipóteses:

- **Hipótese nula A (H0 A):** não há diferença entre a opinião dos usuários especialistas e não especialistas quanto a expressividade dos objetos da BDM4IoT; e

- **Hipótese nula B (H0 B):** não há diferença entre a opinião dos usuários especialistas e não especialistas quanto à facilidade de leitura e interpretação dos modelos de comportamento gerados, utilizando-se a BDM4IoT.

Conforme mencionado anteriormente, para verificar qual teste estatístico utilizar no teste de hipótese, foi verificado a normalidade da distribuição com o teste *Shapiro Wilk*. Para expressividade e facilidade de leitura e interpretação, o valor de W é 0 para ambas as amostras, e pela regra de decisão do teste, verificou-se que a amostra não provém de uma população normal, sendo assim, para decidir se as hipóteses seriam rejeitadas ou não, foi aplicado o teste estatístico não-paramétrico *Mann-Whitney* de amostras independentes. O intervalo de confiança utilizado é: $\alpha = 0{,}05$, ou seja, um nível de confiança de 95%.

Para hipótese nula H0 A, após aplicar o teste estatístico para amostras independentes, foi encontrado um p-valor de 0,229. Esse resultado é não significante para $p \leq 0{,}05$. Logo, a hipótese (H0 A) não é rejeitada, confirmando-se que, com um nível de confiança de 95%, não há diferença entre a opinião dos usuários especialistas e não especialistas quanto à expressividade dos objetos do BDM4IoT.

Para hipótese nula H0 B, após aplicado o teste estatístico, foi obtido o seguinte resultado: p-valor = 0.089, ou seja, é não significante para $p \leq 0{,}05$. Logo, a hipótese (H0 B) não é rejeitada, sobre a qual pode-se inferir que, com um nível de confiança de 95%, não há diferença entre a opinião dos usuários especialistas e não especialistas quanto à facilidade de leitura e interpretação dos modelos de comportamento gerados utilizando a BDM4IoT.

7.2.3. Facilidade de Aprendizado do BDM4IoT e Avaliação da *Êxodo GUI*

Para não tornar o experimento cansativo, após o término da primeira série de experimentos, foi dado um intervalo de 10 minutos aos usuários para água e café e, após o fim desse período, os usuários tomaram aos seus lugares de acordo com seus grupos e a segunda série de experimentos foi iniciada.

A segunda série de experimentos consistiu em analisar se o modelo proposto (BDM4IoT) é fácil de assimilar, no sentido de compreender se os usuários especialistas e não especialistas criem modelos de comportamento de diferentes níveis de dificuldade, mesmo tendo pouco contato com o modelo. Para alcançar esse objetivo, foram elaborados

quatro cenários com níveis gradativos de dificuldade sendo assim: o primeiro o mais fácil; o segundo com dificuldade média; o terceiro difícil; e o quarto cenário fazendo uso de objetos de regras de negócio mais complexas para usuários não especialistas. É evidente que os cenários definidos, apesar dos níveis de dificuldade, são simples de se entenderem, principalmente por já fazerem parte do cotidiano de quaisquer usuários, sendo ou não especialistas.

Para execução dessa atividade, foi dado um tempo livre para sua execução, porém, para registrar o tempo de modelagem dos usuários, foi acrescentado na ferramenta Êxodo GUI um cronômetro onde os usuários deveriam iniciá-lo ao início da modelagem e finalizá-lo após termino. Ao final da modelagem de cada cenário por usuário, fatores relevantes, como o tempo, objetos, configurações e ligações lógicas, foram detalhadamente registrados, a fim de calcular o percentual no final de completude dos modelos de cada cenário, por tipo de usuário (especialistas e não especialistas). Cabe destacar que o percentual de completude é calculado de acordo com a formula a seguir:

$$Pcmpl = \frac{TxAcObj + TxAcConf + TxAcSeqLog}{3}$$

Onde: *TxAcObj* é a taxa de acerto dos objetos utilizados; *TxAcConf*; taxa de acerto das configurações realizadas corretamente; e *TxAcSeqLog*, a taxa de acerto das sequencias lógicas realizadas corretamente. Por fim, é importante esclarecer que a taxa de acerto é calculada conforme fórmula apresentada na seção 7.2.1.

Antes de dar início à execução das atividades previstas na segunda série de experimentos, foi realizada uma palestra de 15 minutos aos usuários apresentando o modelo BDM4IoT, seu principal objetivo e todos os objetos que o compõem, seus parâmetros e a utilização correta de cada objeto. Após isso, foi apresentado uma variedade de exemplos de modelos de comportamento, momento em que, oportunamente, os usuários puderam sanar todas as suas dúvidas. Por fim, a ferramenta *Êxodo GUI* foi rapidamente apresentada aos usuários, sem entrar em grandes detalhes quanto à sua utilização, mas somente o acesso aos objetos e as principais funcionalidades da ferramenta. Ao termino da modelagem dos cenários, os dados foram registrados e agrupados conforme apresentado na Tabela 7.1:

Tabela 7.1. Percentual de Completude dos modelos de comportamento criados.

Usuários Especialistas				
Cenários	**Objetos**	**Configurações**	**Sequência**	**Completude**
1	100%	100%	100%	100%
2	100%	100%	100%	100%
3	100%	100%	100%	100%
4	100%	100%	100%	100%
Usuários Não Especialistas				
1	100%	100%	100%	100%
2	100%	100%	100%	100%
3	100%	96,43%	100%	98,81%
4	100%	97,73%	100%	99,24%

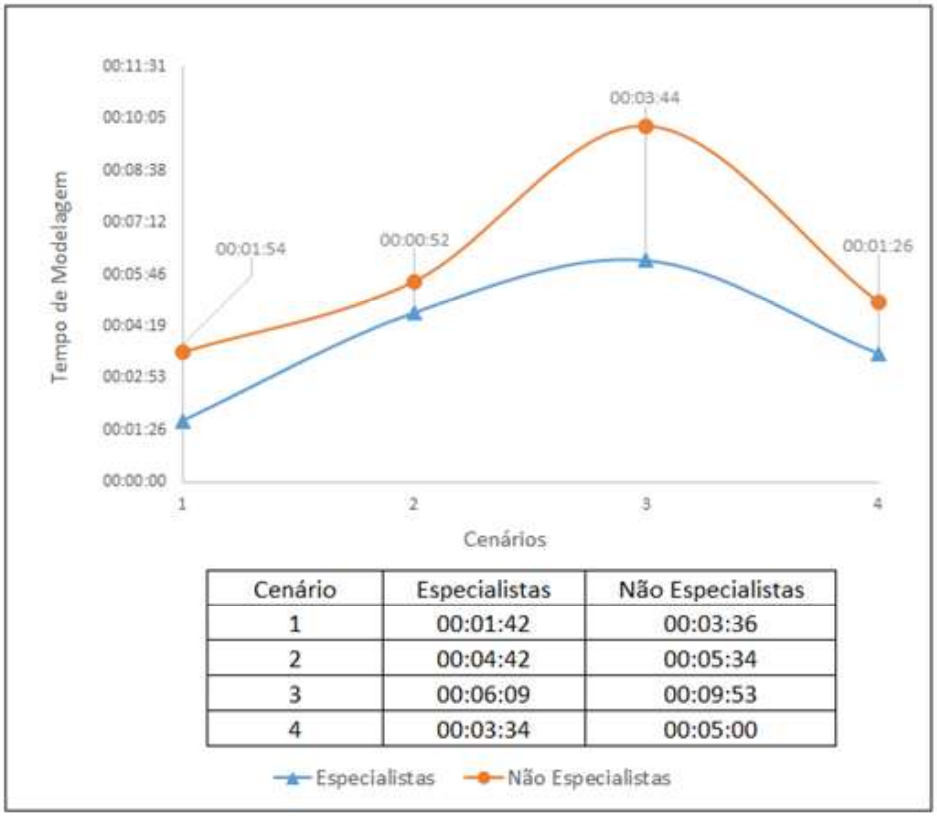
De acordo com os resultados obtidos, percebe-se que os usuários especialistas não tiveram dificuldades na elaboração dos cenários apresentados, utilizando, configurando e interligando corretamente 100% dos objetos solicitados para elaboração desses cenários.

Usuários não especialistas elaboraram corretamente 100% dos cenários 1 e 2. Já no cenário 3, onde se observa o uso de objetos de regras de negócio, aqueles usuários utilizaram corretamente 100% previstos e, dentre esses objetos, 96,43% foram configurados também corretamente, obtendo um percentual de completude de 98,81%. No quarto e último cenário, os usuários não especialistas obtiveram os seguintes resultados: 100% dos objetos foram utilizados corretamente, dos quais 97,73% foram corretamente configurados e 100% foram interligados conforme previsto, obtendo um percentual de completude de 99,24%. Dessa forma, é possível perceber que tanto usuários não especialistas quanto não especialistas não tiveram grandes dificuldades em modelar os cenários propostos. É importante ressaltar que a principal configuração realizada

incorretamente foram os gatilhos utilizados nos objetos de início de fluxo, onde os usuários deveriam informar o método e as condicionais para disparo do gatilho.

A Figura 7.4 apresenta os resultados obtidos quanto ao tempo utilizado pelos usuários especialistas e não especialistas para modelagem dos cenários.

A fim de se verificar o desempenho dos usuários com relação ao tempo de modelagem, foi necessário definir um tempo médio esperado para finalização da modelagem de cada cenário. Para isso, foi necessário modelar 4 vezes cada cenário proposto, registrar o tempo e extrair a média, cuja finalidade era de definir para cada um o tempo de modelagem esperado.



Cenário	Especialistas	Não Especialistas
1	00:01:42	00:03:36
2	00:04:42	00:05:34
3	00:06:09	00:09:53
4	00:03:34	00:05:00

Figura 7.4. Tempo de modelagem dos cenários.

De acordo com os resultados obtidos, percebe-se que os usuários especialistas ficaram bem próximos do tempo de esperado. Sendo assim, como base de comparação dos resultados obtidos pelos usuários não especialistas, serão utilizados os obtidos pelos usuários especialistas, a fim de definir se houve ganho de expertise dos não especialistas de um cenário para o outro.

No primeiro cenário, os usuários não especialistas demoraram 1 minuto e 54 segundos a mais em relação aos usuários especialistas para finalizar a modelagem.

Contudo, quando observada a diferença de tempo levada pelos usuários não especialistas na modelagem do segundo cenário, que foi de 52 segundos, percebeu-se que houve melhoras quanto às habilidades desses usuários com relação à modelagem de cenários que não fazem uso de objetos complexos (regras de negócio), reduzindo o tempo de modelagem em 1 minuto e 2 segundos em relação ao tempo levado para se modelar o primeiro cenário.

Com relação aos resultados obtidos na modelagem do terceiro cenário, o tempo de modelagem dos usuários não especialistas, em relação ao tempo levado pelos usuários não especialistas, foi de 3 minutos e 44 segundos, que, aparentemente, é elevado, se considerando os demais tempos obtidos. No entanto, quando observado o tempo obtido no cenário 4 pelos usuários não especialistas, é possível concluir que houve melhoras quanto às suas habilidades em relação à modelagem de cenários utilizando regras de negócio complexas. É importante destacar que esse ganho de habilidade pode, factualmente, ser observado por meio dos dados apresentados pela Tabela 7.1, em que se nota o aumento do percentual de completude, passando de 98,81%, no cenário 3, para 99,24%, no cenário 4.

A segunda atividade desse experimento consistiu em avaliar a ferramenta *Êxodo GUI* de acordo com as seguintes características: i) intuitividade, capacidade de a ferramenta ser utilizada por quaisquer usuários de forma simples, sem que haja a necessidade de treinamento de sua correta utilização: ii) manipulabilidade dos objetos, capacidade de os usuários utilizarem a ferramenta para manipulação dos objetos do BDM4IoT, como adicionar, remover, alterar, mover, redimensionar, interligar e acessar suas propriedades; iii) configurabilidade dos objetos, facilidade para configuração de todos os parâmetros obrigatórios dos objetos; e iv) legibilidade, capacidade da ferramenta transmitir informações aos usuários de forma legível.

Para avaliação das características supracitadas, foram elaborados questionários onde os usuários deveriam informar sua opinião de acordo com uma escala *Likert*, onde 1) discordo totalmente, 2) discordo parcialmente, 3) não sei opinar, 4) concordo parcialmente e 5) concordo totalmente. Após isso, os dados foram extraídos e agrupados conforme apresentado pela Figura 7.5.

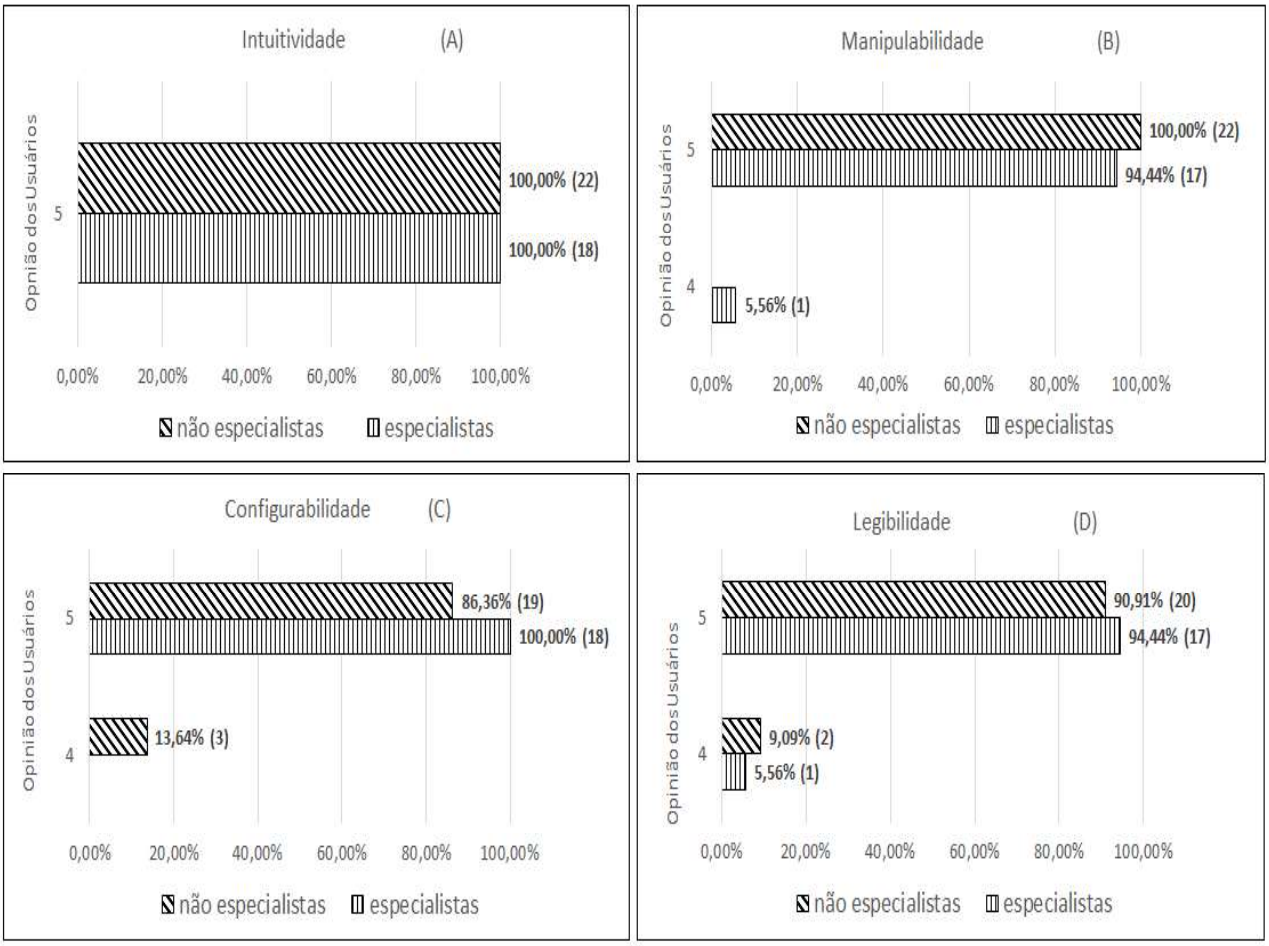


Figura 7.5. Resultados adquiridos quanto à opinião dos usuários com relação às métricas estabelecidas.

De acordo com os resultados obtidos quanto à intuitividade (Figura 7.4 A), 100% dos usuários de ambos os grupos concordaram totalmente (5) que a ferramenta Êxodo GUI é intuitiva e, ainda, que pode ser facilmente utilizada por qualquer tipo de usuário, sem a necessidade de treinamento.

De acordo com os resultados obtidos sobre manipulabilidade (Figura 7.4 B), 100% dos usuários não especialistas concordam totalmente (5) sobre a característica de manipulabilidade. Já usuários especialistas, 94,44% concordam totalmente (5) e 5,56% concordam parcialmente (4). Quando questionado sobre os motivos que o levaram a concordar parcialmente sobre a manipulabilidade, os usuários informaram que, em algumas vezes, a ferramenta não obedeceu aos comandos de arrastar e redimensionar, devendo-se considerar, segundo eles, como fator de avaliação, embora dissessem que se sentiam satisfeitos. Sendo assim, é notório que a ferramenta *Êxodo GUI* foi bem avaliada no critério de manipulabilidade dos objetos, recebendo opiniões entre 4 e 5, valores

considerados significantes, o que leva à conclusão de que a abordagem utilizada para manipulação dos objetos e acesso às suas propriedades são fáceis de serem utilizadas por qualquer tipo de usuário, tanto especialistas quanto não especialistas.

De acordo com os resultados obtidos sobre configurabilidade (Figura 7.4 C), 100% dos usuários especialistas concordam totalmente (5) que os objetos são fáceis de serem configurados. Já os usuários não especialistas, 86,36% concordam totalmente e 13,64% concordam parcialmente (4). Quando os que optaram por concordo parcialmente foram questionados, justificaram a falta de experiência quanto aos parâmetros exigidos, como, por exemplo, os valores máximo e mínimo e o valor de retorno do método esperado. Mas, para as demais configurações, como disparo manual e temporal, foram relativamente fáceis de entender.

Apesar das dificuldades detectadas, os resultados demonstram que a configuração dos objetos por meio da ferramenta é fácil de ser realizada para usuários com maior experiência em ferramentas dessa natureza. Já para usuários não especialistas, de acordo com os resultados, observa-se uma pequena dificuldade quanto ao processo de configuração. No entanto, todos os usuários, apesar das dificuldades, conseguiram configurar corretamente a grande maioria dos objetos requeridos para os cenários elaborados.

De acordo com os resultados obtidos quanto à legibilidade das informações (Figura 7.4 D), 94,44% dos usuários especialistas concordam totalmente (5) e 5,56% parcialmente (4) que as informações contidas na ferramenta são legíveis e de fácil entendimento. Usuários não especialistas: 90,91% concordam totalmente (5) e 9,09% concordam parcialmente (4). Quando os que concordaram parcialmente foram questionados, todos informaram que a ferramenta deveria possuir algum tipo de mecanismo por meio do qual pudessem aclarar as suas dúvidas em relação aos parâmetros exigidos na configuração dos objetos, como, por exemplo, a apresentação de informações, quando o usuário posicionar o *cursor* do *mouse* por sobre um parâmetro, a respeito do qual se tenham dúvidas. Analisando os resultados obtidos dos usuários especialistas e não especialistas, percebe-se que existem melhorias a se realizarem, para tornar as informações apresentadas pela ferramenta mais legíveis, o que influenciaria diretamente nos resultados obtidos, especialmente quanto à configuração dos objetos e por consequência, segundo a opinião dessa parcela de usuários. No entanto, é possível perceber que ambos os tipos de usuários,

dos dois grupos, demonstraram um bom nível de satisfação, principalmente quanto à legibilidade das informações apresentadas.

Para verificar se, estatisticamente, há diferença entre a opinião dos usuários especialistas e não especialistas quanto às características avaliadas, foram levantadas as seguintes hipóteses:

- **Hipótese nula C (H0 C):** não há diferença entre a opinião dos usuários especialistas e não especialistas quanto à Manipulabilidade dos objetos da BDM4IoT utilizando a ferramenta *Êxodo GUI*;
- **Hipótese nula D (H0 D):** não há diferença entre a opinião dos usuários especialistas e não especialistas quanto à configurabilidade dos modelos de comportamento gerados utilizando a ferramenta *Êxodo GUI*; e
- **Hipótese nula E (H0 E):** não há diferença entre a opinião dos usuários especialistas e não especialistas quanto à legibilidade das informações apresentadas pela ferramenta *Êxodo GUI*.

Para verificar qual teste estatístico utilizar no teste de hipótese, foi verificado a normalidade da distribuição com o teste *Shapiro Wilk*. Para todas as amostras, manipulabilidade, configurabilidade e legibilidade, o valor de W é 0, e pela regra de decisão do teste, verificou-se que as amostras não provem de uma população normal, sendo assim, para decidir se as hipóteses seriam rejeitadas ou não, foi aplicado o teste estatístico não-paramétrico *Mann-Whitney* para amostras independentes. O intervalo de confiança utilizado é: $\alpha = 0,05$, ou seja, um nível de confiança de 95%.

Para hipótese nula H0 C, após aplicar o teste estatístico, foi encontrado um p-valor de 0,778. Esse resultado é não significante para $p \leq 0,05$. Logo, a hipótese (H0 C) não é rejeitada, confirmando-se que, com um nível de confiança de 95%, não há diferença entre a opinião dos usuários especialistas e não especialistas quanto à manipulabilidade dos objetos do BDM4IoT, por meio da ferramenta *Êxodo GUI*.

Para hipótese nula H0 D, após aplicar o teste estatístico, foi encontrado um p-valor de 0,476. Esse resultado é não significante para $p \leq 0,05$. Logo, a hipótese (H0 D) não é rejeitada, confirmando-se que, com um nível de confiança de 95%, não há diferença entre a opinião dos usuários especialistas e não especialistas quanto à configurabilidade dos objetos do BDM4IoT, utilizando a ferramenta *Êxodo GUI*.

Para hipótese nula H0 E, após aplicar o teste estatístico, foi encontrado um p-valor de 0,861. Esse resultado é não significante para $p \leq 0,05$. Logo, a hipótese (H0 E) não é rejeitada, confirmando que, com um nível de confiança de 95%, não há diferença entre a opinião dos usuários especialistas e não especialistas quanto à legibilidade das informações apresentadas pela ferramenta *Êxodo GUI*.

É importante ressaltar que a hipótese para intuitividade não foi testada devido à todos os usuários de ambos os grupos concordarem totalmente com a característica de legibilidade. A seguir, a seção 7.2.4 apresenta a condução e os resultados obtidos na terceira e última série de experimentos.

7.2.4. Comparação da Ferramenta *Êxodo GUI* e do Modelo Conceitual BDM4IoT com Soluções Similares

A terceira série de experimentos consistiu em comparar a ferramenta *Êxodo GUI* com duas ferramentas similares existentes na literatura. A primeira ferramenta selecionada foi a *ASPIRE Editor*, ferramenta gráfica utilizada pelo *middleware* ASPIRE (seção 3.2.1) para definição das regras de negócio utilizando modelos. A segunda foi o *middleware* OpenHAB (seção 3.2.8), que faz uso de uma ferramenta gráfica para permitir que os usuários possam criar as regras de negócio a serem executadas no ambiente. O principal motivo que levou a adoção dessas ferramentas é que são as únicas, dentre as ferramentas apresentadas anteriormente no capítulo 3, que estão disponíveis na literatura para testes e que fazem uso de uma interface gráfica para modelagem de comportamento de ambientes para IoT.

Para avaliação dessas ferramentas, foi elaborado um cenário em que os usuários dispunham de um tempo máximo de 10 minutos para cada ferramenta modelar o cenário apresentado. Ao final, os usuários deveriam avaliar as ferramentas utilizando os mesmos critérios de avaliação utilizados na avaliação da ferramenta *Êxodo GUI* (seção 7.3). Objetivamente, foi utilizada uma escala *Likert* para definir a satisfação dos usuários, sendo: 1) discordo totalmente; 2) discordo parcialmente; 3) não sei opinar; 4) concordo parcialmente; e 5) concordo totalmente. Ao final dessa atividade, os dados foram analisados e agrupados por ferramenta, a fim de se tornar visível a comparação entre os agrupamentos. A Figura 7.6 apresenta a comparação das ferramentas quanto à característica de intuitividade.

Com relação à essa característica para ferramenta Êxodo GUI, foram obtidos os seguintes resultados: todos os 40 (100%) dos usuários concordam totalmente a intuitividade provida pela ferramenta *Êxodo GUI*. Para ferramenta *OpenHAB*, 22 (55%), usuários concordam parcialmente (4) e 18 (45%) discordam parcialmente (2). Para ferramenta ASPIRE, 38 (95%) discordam totalmente (1) e 2 (5%) discordam parcialmente (3). Sendo assim, com relação à intuitividade, a ferramenta *Êxodo GUI* foi a melhor avaliada, sendo, então, a ferramenta que mais se destaca quanto à intuitividade de uso por usuários com diferentes níveis de conhecimento, principalmente para os usuários não especialistas, segundo a pesquisa.

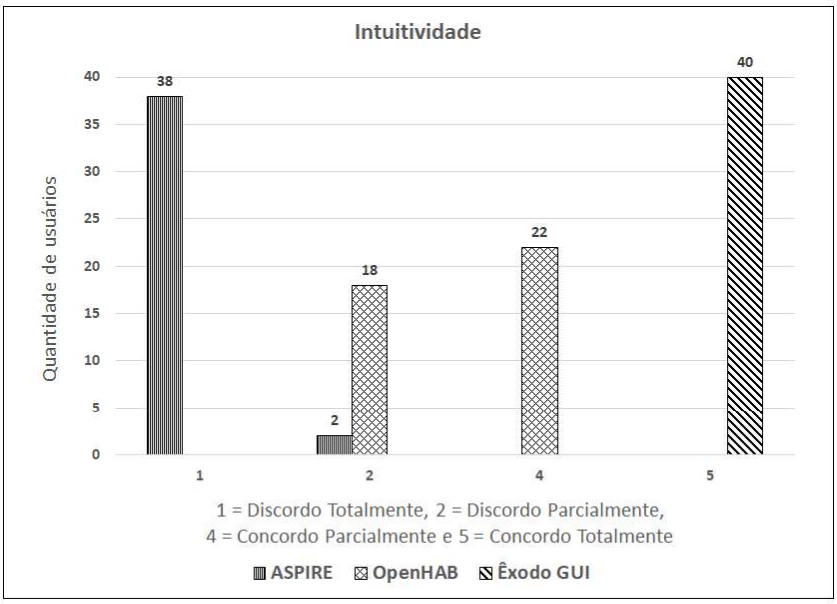


Figura 7.6. Comparação das ferramentas quanto à Intuitividade.

É importante ressaltar que a ferramenta *OpenHAB* é uma excelente alternativa quanto ao uso para modelagem de comportamento em ambientes para IoT, com base em modelos. Contudo, para melhor aproveitar o potencial dessa ferramenta, é necessário que haja algum treinamento com os usuários, para que, assim, possam manipular corretamente a ferramenta, a fim de se criarem e/ou adicionarem novos dispositivos. A ferramenta ASPIRE é uma ferramenta extremamente técnica voltada para ambientes industriais, o que dificulta os usuários não especialistas manipulá-la facilmente. A seguir, a Figura 7.7 apresenta os resultados obtidos quanto à manipulabilidade:

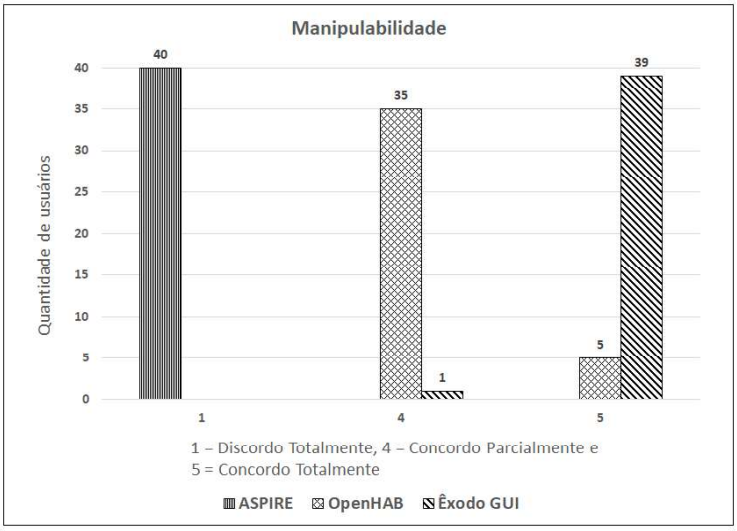


Figura 7.7. Comparação das ferramentas quanto à manipulabilidade.

Com relação à característica de manipulabilidade dos objetos, para a ferramenta *Êxodo GUI*, 39 (97,5%) usuários concordaram totalmente (5) e 1 (2,5%) concordou parcialmente (4). Para ferramenta *OpenHAB*, 5 (12,5%) usuários concordam totalmente (5) e 35 (87,5%) concordam parcialmente (4). Para ferramenta ASPIRE, todos os 40 (100%) usuários dizem discordar totalmente (1). De acordo com os resultados obtidos, é possível concluir que a ferramenta Êxodo GUI foi a melhor avaliada pelos usuários com relação à manipulação dos objetos, assim como a ferramenta OpenHAB. Já em relação à ferramenta ASPIRE, todos os usuários tiveram grandes dificuldades, uma vez que tal ferramenta não dispõe facilmente de objetos e dispositivos de acesso. Mesmo as que se encontram disponíveis, também são difíceis de serem manipulados intuitivamente. A seguir, a Figura 7.8 apresenta os resultados obtidos quanto à característica de configurabilidade:

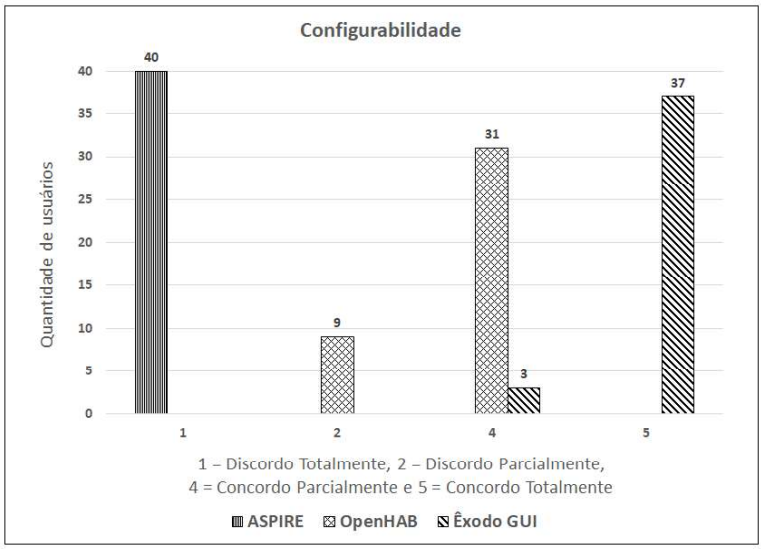


Figura 7.8. Comparação das ferramentas quanto à Configurabilidade

De acordo com os resultados obtidos quanto à configurabilidade dos objetos, para ferramenta *Êxodo GUI*, 37 (92,5%) usuários concordam totalmente (5) e 3 (7,5%) concordam parcialmente (4). Para ferramenta *OpenHAB*, 31 (77,5%) usuários concordam totalmente (5) e 9 (22,5%) concordam parcialmente. Para ferramenta ASPIRE, todos os 40 (100%) usuários discordam totalmente. Sendo assim, a *Êxodo GUI* foi a ferramenta melhor avaliada com relação à configurabilidade dos objetos. O uso da ferramenta *OpenHAB*, por exigir uma quantidade maior de parâmetros a serem informados para configuração dos objetos, gerou muitas dúvidas nos usuários quanto à configuração dos objetos. Ainda assim, a ferramenta OpenHAB foi bem avaliada. Já a ferramenta ASPIRE, devido à sua elevada característica técnica, os objetos que conseguiram ser manipulados não conseguiram ser configurados devido à complexidade dos parâmetros exigidos. A seguir, a Figura 7.9 apresenta os resultados obtidos quanto à característica de legibilidade das informações apresentadas:

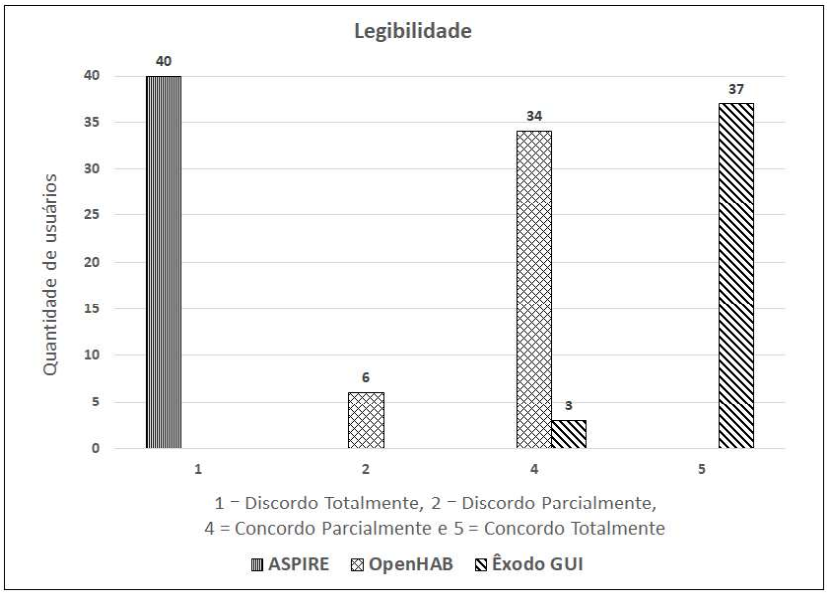


Figura 7.9. Comparação das ferramentas quanto a Legibilidade.

Com relação aos resultados obtidos quanto à legibilidade, para ferramenta *Êxodo GUI*, 37 (92,5%) usuários concordam totalmente (5) e 3 (7,5%) concordam parcialmente (4). Para a ferramenta *OpenHAB*, 34 (85%) usuários concordam parcialmente (4) e 6 (15%) discordam parcialmente (4). Para a ferramenta ASPIRE, todos os 40 (100%) usuários discordam totalmente. Sendo assim, das três ferramentas avaliadas pelos usuários, *Êxodo GUI* é ferramenta que apresenta maior legibilidade das informações, que os permitiram construir modelos de comportamento, mesmo no primeiro contato.

Por fim, a última atividade desse experimento teve por objetivo comparar o modelo BDM4IoT com o modelo BPMN, em relação à capacidade de representação dos objetos de dispositivos e regras de negócio, para executar modelagem de comportamento de ambientes para IoT, bem como a facilidade de leitura e interpretação dos modelos apresentados. Para essa atividade, foram solicitados somente os usuários especialistas para execução dessa atividade, uma vez que todos já tinham experiência quanto ao uso do modelo BPMN. Com isso, para cada modelo, foi dado um tempo máximo de 5 minutos para os especialistas poderem modelar o comportamento solicitado. Ao final do tempo estipulado para o uso de cada modelo, os dados foram extraídos e agrupados, conforme apresentados pela Figura 7.9:

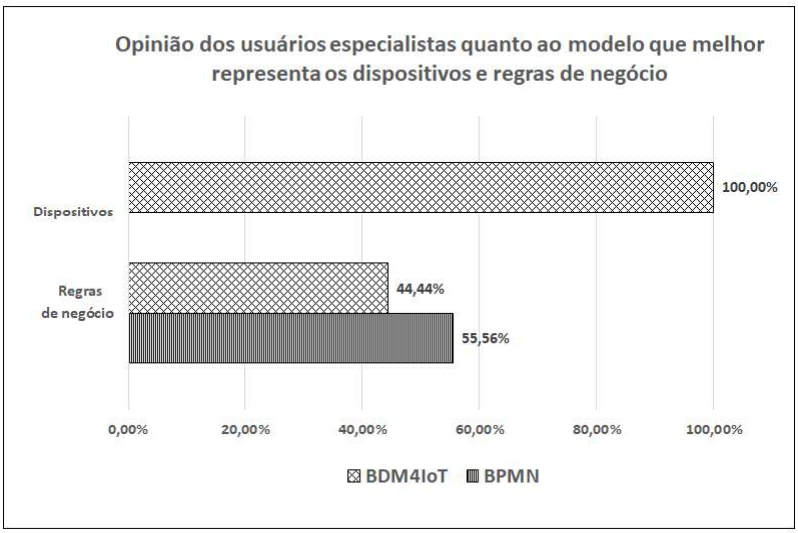


Figura 7.10. Modelos que melhor representam os objetos de dispositivos e regras de negócio em um modelo de comportamento para IoT.

De acordo com os resultados obtidos, para representatividade de dispositivos, o BDM4IoT foi selecionado por 100% dos usuários especialistas como sendo o melhor modelo para representar dispositivos em um modelo de comportamento de ambiente para IoT. Quanto à representatividade das regras de negócio, 44,44% dos usuários selecionaram o modelo BPMN e 55,56% o modelo BPMN, sendo o BPMN classificado como o melhor para representar regras de negócio. No entanto, quando confrontado sobre os motivos que os levaram a definir o BPMN, eles informaram que o BPMN possui uma quantidade maior de objetos para representação de regras de negócio, inclusive os objetos de regras de negócio existentes no BDM4IoT, mas que não existem no BPMN, podem ser facilmente contornados utilizando os objetos disponíveis pelo BPMN, como, por exemplo, o objeto para repetição condicional do BDM4IoT.

Após o termino dessa atividade, os usuários não especialistas foram convidados a avaliar os modelos criados e definir quais modelos mais fáceis de ler e interpretar na opinião desses usuários. De acordo com os resultados obtidos, 100% dos usuários não especialistas informaram que os modelos criados utilizando o BDM4IoT são mais fáceis de entender e interpretar, em comparação aos modelos criados no BPMN. Sendo assim, pode-se concluir que, apesar de o modelo BPMN ser o melhor para representação de regras de negócio, a quantidade de objetos utilizados para representar uma determinada regra de negócio, como, por exemplo, uma repetição, influência de forma negativa na legibilidade e clareza do modelo.

7.2.5. Ameaças à Validade

Nessa seção são apresentadas e discutidas as possíveis ameaças à validade dos resultados obtidos nos experimentos realizados, são elas:

1. **Validade Interna -** Relacionada ao risco de fatores não identificados terem influenciado em eventual relacionamento de casualidade entre tratamento e resultado, sem o conhecimento prévio do experimentador.
    a. *Instrumentação:* Foram utilizados os instrumentos adequados para ambos os grupos (especialistas e não especialistas), tais como, questionários que passaram por revisões e foram submetidos a um piloto. Apresentação de palestras para introdução de termos e importantes conceitos relacionados a IoT, a fim de nivelar conhecimento de ambos os grupos com relação a IoT. Para realização do experimento de facilidade de aprendizado, foi necessário fazer uso da ferramenta gráfica *Êxodo GUI*, dessa forma, a tela principal da ferramenta foi previamente apresentada, bem como a localização dos objetos e funcionalidades. No experimento de comparação da ferramenta *Êxodo GUI* com ferramentas similares, um ponto que poderia apresentar uma ameaça significativa nos resultados foi a apresentação previa da ferramenta *Êxodo GUI*, no experimento de facilidade de aprendizado, no entanto, para nivelar o conhecimento dos usuários, as ferramentas *ASPIRE* e *OpenHAB* foram também previamente apresentada a ambos os grupos, utilizando os mesmos critérios utilizados na apresentação da *Êxodo GUI*, tela principal e localização de objetos e funcionalidades. Por fim, no

experimento de comparação do modelo BDM4IoT com o BPMN, conforme mencionado na seção 7.2.4, para não influenciar a resposta dos usuários não especialistas, os mesmos foram retirados do ambiente até que os usuários especialistas finalizassem a modelagem dos cenários solicitados, após isso, os usuários não especialistas foram chamados para avaliação dos modelos.

b. *Seleção:* Todos os usuários de ambos os grupos (especialistas e não especialistas), foram selecionados de maneira aleatória, não havendo qualquer influência do pesquisados com os usuários que participaram dos experimentos. A divisão por grupo especialista e não especialistas se deu por meio do perfil de cada usuário, sendo assim, usuários com qualquer formação técnica em computação, foram selecionados para o grupo de usuários especialistas e usuários sem formação técnica foram selecionados para o grupo de usuários não especialistas. É evidente que nesse tipo de seleção, existe o risco de um usuário, selecionado como não especialista, possuir tanto conhecimento técnico em computação quanto um usuário especialista. Dessa forma, para reduzir essa probabilidade, foi aplicado um questionário com os usuários sobre possíveis níveis de conhecimento em computação, antes que os mesmos fossem selecionados para um determinado grupo.

c. *Maturação:* Para evitar a desmotivação dos grupos durante o andamento dos experimentos, no final de cada experimento era dado um intervalo de 15 minutos para os usuários para ingestão de água, café e um pequeno lanche. Além disso, para evitar a desistência dos usuários participantes, foi sorteado no final dos experimentos 2 *Pen Drivers* de 16Gb.

d. *Contaminação:* Para evitar a contaminação entre ambos os grupos, os usuários foram instruídos a não realizar comunicação sobre os experimentos realizados até a finalização dos experimentos. No momento do intervalo de 15 minutos, para evitar a comunicação entre os grupos, os mesmos foram organizados e dispostos em locais distintos.

2. **Validade Externa -** Questões relacionadas à ameaça dos resultados do estudo não serem generalizáveis a projetos reais na indústria. As principais ameaças identificadas foram:

a. *Tempo:* O tempo para realização dos experimentos foi o mesmo para ambos grupos, por fim, o tempo de treinamento dado aos grupos no início de cada experimento foi também o mesmo.
b. *Configuração do Experimento:* Para realização dos experimentos onde foi necessária a utilização de um computador, tais experimentos foram realizados em um laboratório onde o parque computacional possuía as mesmas configurações, processador, memoria, armazenamento, sistema operacional e sistemas.

3. **Validade de Conclusão -** Relacionada a questões que ameaçam a habilidade de traçar conclusões corretas sobre o relacionamento entre tratamentos e resultados de um estudo experimental:
    a. *Analise e interpretação estatística dos resultados:* foram realizados os passos necessários para aplicabilidade correta de testes estatísticos e análise, verificação de normalidade, tamanho de amostra e teste estatístico apropriado.

7.3 Considerações Finais

No bojo deste capítulo buscou-se apresentar, da forma mais expressiva possível, o planejamento e condução dos experimentos realizados para avaliação do modelo BDM4IoT e da ferramenta Êxodo GUI, tendo sido realizado com um grupo de 40 usuários, com idades variadas, entre 26 e 55 anos de idade. Esses usuários voluntários foram divididos em dois principais grupos: especialistas e não especialistas. O Grupo de usuários especialistas foi composto por 18 profissionais, com formação em alguma área da computação, seja de nível médio, superior, especialização e/ou pós-graduação, enquanto o grupo dos não especialistas, por 22 profissionais de diferentes áreas.

Os experimentos realizados foram divididos em três rodadas, das quais a primeira avaliou duas importantes características do modelo: i) Expressividade, para verificar se os ícones/imagens, definidas para os objetos do modelo, expressam sua real finalidade; e ii) legibilidade do modelo, para verificar se os modelos criados são fáceis de se ler e se compreender.

Já na segunda rodada, a finalidade do experimento foi verificar se o modelo BDM4IoT é de fácil entendimento e utilização. Para a sua execução, utilizou-se da

interface gráfica Êxodo GUI e, ao final, foi necessário realizar a avaliação da ferramenta em quatro diferentes características: i) intuitividade, capacidade de a ferramenta ser utilizada por quaisquer usuários de forma simples, sem que haja a necessidade de treinamento para sua correta utilização; ii) manipulabilidade dos objetos, capacidade de os usuários utilizarem a ferramenta na manipulação dos objetos do BDM4IoT, como adicionar, remover, alterar, mover, redimensionar, interligar e acessar suas propriedades; iii) configurabilidade dos objetos, facilidade para configuração de todos os parâmetros obrigatórios dos objetos; e iv) legibilidade, capacidade de a ferramenta transmitir informações aos usuários de forma legível.

Concluindo a experiência, a terceira e última rodada teve por objetivo comparar o *Êxodo GUI* com ferramentas disponíveis na academia, tais como: *ASPIRE Editor* e *OpenHAB*, além de também comparar o modelo BDM4IoT ao modelo BPMN, para fins de modelagem de comportamento de ambientes para IoT.

Ademais, verificou-se que os resultados obtidos na primeira rodada de experimentos evidenciam que os objetos que compõem o BDM4IoT possuem um bom grau de expressividade, posto que foi possível de comprovar que tanto a usuários especialistas quanto não especialistas puderam identificar corretamente a maioria dos objetos do BDM4IoT, ou seja, obtiveram sucesso não só em relação a objetos de dispositivo, mas também aos de regras de negócio. Além disso, quanto à facilidade de leitura e interpretação, os resultados obtidos demonstraram que os modelos de comportamento criados utilizado o BDM4IoT podem ser facilmente lidos e interpretados por usuários com diferentes níveis de conhecimento, sejam eles técnicos ou não.

Os resultados obtidos na segunda rodada de experimentos evidenciam que o modelo BDM4IoT pode ser facilmente compreendido por usuários com diferentes níveis de conhecimento (especialista e não especialista). Inclusive, quanto à avaliação da fermenta Êxodo GUI, os resultados obtidos demonstram que a ferramenta pode ser utilizada intuitivamente por quaisquer usuários com conhecimentos mínimos em computação, além de possibilitar que possam facilmente manipular, configurar e interligar os objetos do modelo na criação de comportamentos para o ambiente.

Atingindo o ápice dos experimentos, os resultados obtidos na terceira rodada evidenciam que a ferramenta *Êxodo GUI* foi melhor avaliada que o *middleware ASPIRE* e o *OpenHAB*. Já pelos resultados obtidos quanto à comparação do modelo BDM4IoT com o

BPMN, evidenciou que a BDM4IoT representa da melhor forma os dispositivos presentes no ambiente. Cabe ressaltar, também, que, com relação a regras de negócio, o BPMN foi escolhido a melhor ferramenta para representar regras de negócio. Contudo, com relação à facilidade de leitura e legibilidade dos modelos, a BDM4IoT foi escolhida como tendo os modelos mais fáceis de poderem se ler e interpretar por usuários com pouco ou quase nenhum conhecimento em computação.

Capítulo 8 – Conclusões e Trabalhos Futuros

Com a evolução do paradigma da IoT, é evidente a presença acentuada de dispositivos inteligentes, como sensores, atuadores, *smart phones*, *smart TV*, entre outros, nos mais diversos ambientes do cotidiano das pessoas (doméstico, industrial e comercial). Acredita-se que a IoT logo será um paradigma tão presente no cotidiano das pessoas quanto à Internet. Dessa forma, novas abordagens para facilitar a interação entre os usuários e a IoT devem ser propostas, seja para acesso e/ou gerenciamento remoto de ambientes inteligentes por meio da Internet, ou, até mesmo, ferramentas mais completas, que, além do acesso e gerenciamento remoto, permitam aos usuários criarem e/ou modelarem seus próprios ambientes inteligentes para IoT. Isto é, assim como faz atualmente a Internet, sendo repleta de soluções criadas para os usuários não especialistas criarem seus próprios *web sites*, como, por exemplo, *yahoo pipes*, *Blogs*, *Wix*, entre outros.

Nesse sentido, de forma abrangente, por meio desta tese buscou-se apresentar o *framework Êxodo*, uma solução para IoT com foco no usuário final. Diferente das atuais soluções para IoT, que foram desenvolvidas para atender somente a usuários especialistas, o *framework Êxodo*, uma vez utilizando uma abordagem baseada em modelos conceituais e ferramentas gráficas, permite que tanto usuários especialistas quanto não especialistas configurarem e criem o comportamento de seus próprios ambientes para IoT.

Nesse contexto, o presente capítulo tem o objetivo não só de instruir com a demonstração de simples ações, mas também de apresentar as conclusões, contribuições e trabalhos futuros que se pretendem desenvolver, a fim de tornar o *framework Êxodo* disponível para qualquer usuário (especialista e não especialista). Desse modo, está dividido conforme se segue: a seção 7.1 apresenta as conclusões, dificuldades enfrentadas e contribuições desse trabalho; depois disso, a seção 7.2 apresenta as principais propostas de trabalho futuros que precisam ser desenvolvidas, para que o *framework Êxodo* possa se tornar praticável por usuários com diferentes níveis de conhecimento.

8.1 Conclusões e Contribuições

Para o sucesso do desenvolvimento deste trabalho, o principal fator foi a percepção de que as atuais soluções existentes para a IoT são criadas com objetivo de atender apenas a usuários especialistas e a domínios específicos. Em contrapartida, devido à grande massa de dispositivos inteligentes, com capacidade de se comunicarem e de serem controlados por meio da Internet, é possível perceber que a IoT – embora já até esteja presente no atual cotidiano de pessoas comuns – infelizmente está inacessível. Atualmente, de acordo com Evans [2011], o número de dispositivos conectados à Internet já ultrapassou, em bilhões, o número de pessoas conectadas. Dessa forma, soluções com foco nos usuários devem ser desenvolvidas, a fim de permitir que a IoT se torne acessível a todos os usuários, não apenas aos especialistas de ambientes específicos, como industrial e acadêmico.

Cabe um destaque ao *framework Êxodo*, porquanto é a primeira solução na literatura a caminhar nesse sentido e que, factualmente, propõe soluções que viabilizam o acesso da IoT a todas as pessoas. Para se chegar à essa conclusão, foi realizada, de forma crítica e abrangente, uma profunda pesquisa na literatura, a fim de se entenderem melhor os conceitos, desafios e a problemática ainda a ser estudada. Em se tratando da problemática, procuraram-se buscar soluções, arquiteturas e/ou qualquer abordagem (dentro ou fora do contexto da IoT) que pudessem ser utilizadas para viabilizar o acesso da IoT a usuários com diferentes perfis e níveis de conhecimento. Assim sendo, o estado da arte desenvolvido por este trabalho apresenta 11 soluções de middleware para IoT existentes na literatura, que buscou esclarecer criteriosamente os aspectos arquiteturais e tecnológicos, além de também analisar as abordagens utilizadas pelas soluções que, porventura, sejam utilizadas para permitir a inclusão de usuários não especialistas no processo criativo e gerencial de ambientes para IoT.

Ainda com base no estado da arte desenvolvida, foi possível perceber que a maioria das soluções apresentadas são focadas em atender a domínios e usuários específicos, o que dificulta o acesso à essas soluções para outros domínios e usuários. No entanto, foi possível aprender e extrair dessas soluções a abordagem que possivelmente seriam utilizadas para facilitar e permitir o desenvolvimento de uma nova abordagem para IoT com foco no usuário, como, por exemplo, o uso de modelos conceituais e interface gráfica, abordagem utilizada pelas soluções *Task Computing*, *ASPIRE* e *GSN*. Além desses fatores,

a abordagem interessante para flexibilizar a inclusão de novos serviços e/ou dispositivos providos pela arquitetura, como, por exemplo, a abordagem com base no desenvolvimento de *drivers* para inclusão de novos dispositivos, utilizada pelas soluções *EcoDIF* e *RestThing*.

Entretanto, para se atender a problemática de IoT para todos, bem como superar os principais desafios atualmente existentes na IoT, esta tese apresentou o *framework Êxodo*, uma solução para IoT com foco no usuário, para permitir que usuários com diferentes níveis de conhecimento possam criar/modelar seus próprios ambientes para IoT. Então, foi criado um modelo conceitual, denominado BDM4IoT (*Behavior Definition Model for Internet of Things Ecosystems*), um modelo expressivo e de fácil entendimento por usuários não especialistas. O BDM4IoT foi criado para representar, de forma expressiva, dispositivos inteligentes do mundo real, permitindo que os usuários possam manipulá-los, a fim de se criarem comportamentos. Somando-se a isso, o modelo BDM4IoT também possui um conjunto de objeto de regras de negócio, os quais possibilitam que os usuários possam inferir regras aos modelos criados. Para permitir o acesso e manipulação do modelo BDM4IoT por usuários não especialistas, foi desenvolvida uma interface gráfica denominada *Êxodo GUI*, que possibilita aos dispositivos do mundo real serem vinculados e manipulados, a fim de comporem modelos de comportamento.

Em função desses conceitos e em busca da melhor forma de argumento, emergiu a necessidade de experimentos, principalmente para se testar a eficiência da abordagem desenvolvida com usuários voluntários. Por isso, foi realizado um conjunto de experimentos com um grupo de 40 pessoas, sendo usuários com diferentes perfis profissionais, dos quais 20 são homens e 20 mulheres. Eles foram distribuídos em dois principais grupos: usuários especialistas, formado por 18; e não especialistas, 22 usuários não especialistas. Os resultados obtidos demonstraram que usuários com diferentes níveis de conhecimento não tiveram dificuldades nos seguintes aspectos: identificação correta dos objetivos de cada objeto do modelo; leitura dos modelos de comportamento; e utilização da ferramenta *Êxodo GUI* para criar/modelar comportamentos com diferentes níveis de dificuldade.

As principais contribuições do presente trabalho são apresentadas, conforme se segue:

- Um modelo conceitual para definição de comportamento de ambientes para IoT, denominado BDM4IoT (Behavior Definition Model for IoT EcoSystems), o qual é um intuitivo e de fácil compreensão, composto de objetos que permitem a um usuário sem grandes conhecimentos em computação facilmente modelar os diferentes comportamentos que um ambiente para IoT poderá assumir, sem que, para isso, seja necessária a presença de um usuário especialista para auxiliá-lo;
- Diferentemente do modelo utilizado pela ferramenta gráfica *Task Computing*, o modelo BDM4IoT possibilita tanto a modelagem de cenários simples quanto a automatização de ambientes domésticos, como, por exemplo, cenários complexos (a automatização de ambientes industriais, utilizando um conjunto de objetos auxiliares e de regras de negócio);
- Uma interface gráfica denominada *Êxodo GUI*, criada para auxiliar os usuários na manipulação, configuração e definição de comportamentos, utilizando o modelo BDM4IoT. Tal interface permite que os usuários alterem o comportamento do ambiente em tempo de execução;
- Uma arquitetura flexível para possibilitar que usuários especialistas possam alterar e estender a solução, a fim de melhor adaptá-la às suas necessidades; e
- Um *framework Open Source* com foco no usuário para habilitar a IoT a quaisquer usuários, possibilitando que eles possam criar/modelar seus próprios ambientes para IoT.

8.2 Trabalhos Futuros

Em vista de todos os conceitos que se puderam pesquisar para, depois, aqui analisar e desenvolver, depreende-se que é possível observar várias limitações que, embora não venham afetar o resultado final esperado, seriam de grande importância sua correção. Algumas dessas limitações foram impostas pela complexidade no desenvolvimento, outras impostas ao escopo do projeto, tornando seu desenvolvimento inviável devido à restrição de tempo. Assim sendo, essas limitações, por serem complexas, podem ser suficientemente apontadas para possíveis trabalhos futuros. Em suma, pode-se inferir que se configuram objetos de pesquisa.

Apesar de o objetivo inicial deste trabalho ter sido alcançado, é possível verificar que ainda há o que se desenvolver. Com isso, pode-se ressaltar diversos pontos que podem

ser estendidos e melhorados, como, por exemplo, o desenvolvimento completo do *framework Êxodo*, a fim de produzir resultados mais precisos e permitir que seja disponibilizado gratuitamente a qualquer usuário que, porventura, queira utilizar a solução para gerenciamento de modelagem de comportamentos para IoT.

Outra medida seria a implementação completa da interface gráfica Êxodo GUI, a fim de se realizar o seguinte: a melhora aspectos visuais; a forma de manipulação dos objetos do modelo BDM4IoT; a conclusão do módulo de simulação e geração de base de dados para testes acadêmicos; a criação de um módulo *wizard* para facilitar a inclusão de novos dispositivos e serviços; e, finalmente, a criação de um agente inteligente para auxílio na criação dos modelos, dando dicas em tempo de modelagem, tais como: melhor forma de usar um objeto ou representar um comportamento.

É importante ressaltar que o modelo BDM4IoT é uma solução que pode ser estendida em diversas direções. Dessa forma, o desenvolvimento de uma solução que permita aos usuários especialistas definirem novos objetos e/ou dispositivos ao modelo possibilitaria estender o modelo, disponibilizando novas versões para atender a diferentes domínios e/ou necessidades dos usuários.

Por fim, o desenvolvimento completo da arquitetura, com foco em desempenho dos serviços prestados, uma vez que o foco principal deste trabalho foi totalmente direcionado em verificar se a abordagem desenvolvida possibilitaria a inclusão de usuários com diferentes perfis de conhecimento no processo de gerenciamento e criação de ambientes para IoT.

Referências

ABERER, Karl; HAUSWIRTH, Manfred. Middleware support for the" Internet of Things". 2006.

ABERER, Karl; HAUSWIRTH, Manfred; SALEHI, Ali. Infrastructure for data processing in large-scale interconnected sensor networks. In: Mobile Data Management, 2007 International Conference on. IEEE, 2007. p. 198-205.

ADELMANN, Robert; LANGHEINRICH, Marc; FLÖRKEMEIER, Christian. Toolkit for Bar Code Recognition and Resolving on Camera Phones-Jump Starting the Internet of Things. GI Jahrestagung (2), v. 94, p. 366-373, 2006.

ALCARAZ, Cristina, et al. Wireless sensor networks and the internet of things: Do we need a complete integration? In: 1st International Workshop on the Security of the Internet of Things (SecIoT'10). 2010.

ALDRICH, Frances K. Smart homes: past, present and future. In: Inside the smart home. Springer London, 2003. p. 17-39.

ALLAN, Alasdair. IOS Sensor Apps with Arduino: Wiring the iPhone and iPad into the Internet of Things. "O'Reilly Media, Inc.", 2011.

ALLARD, Jérémie, et al. Jini meets UPnP: an architecture for Jini/UPnP interoperability. In: Applications and the Internet, 2003. Proceedings. 2003 Symposium on. IEEE, 2003. p. 268-275.

AL-MUHTADI, Jalal, et al. Secure smart homes using Jini and UIUC SESAME. In: Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference. IEEE, 2000. p. 77-85.

ANGELES, Rebecca. RFID technologies: supply-chain applications and implementation issues. Information Systems Management, 2005, 22.1: 51-65.

ANGGORJATI, Bayu, et al. RFID added value sensing capabilities: European advances in integrated RFID-WSN middleware. In: Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on. IEEE, 2010. p. 1-3.

ANTONIOU, Grigoris; VAN HARMELEN, Frank. Web ontology language: Owl. In: Handbook on ontologies. Springer Berlin Heidelberg, 2004. p. 67-92.

ATZORI, Luigi, et al. The social internet of things (SIoT)–when social networks meet the

internet of things: Concept, architecture and network characterization. Computer Networks, 2012, 56.16: 3594-3608.

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. The internet of things: A survey. Computer networks, 2010, 54.15: 2787-2805.

BASSI, Alessandro; HORN, Geir. Internet of Things in 2020: A Roadmap for the Future. European Commission: Information Society and Media, 2008.

BIN, Shen; YUAN, Liu; XIAOYI, Wang. Research on data mining models for the internet of things. In: Image Analysis and Signal Processing (IASP), 2010 International Conference on. IEEE, 2010. p. 127-132.

BLUETOOTH, S. I. G. "Specification of the Bluetooth System, version 1.1.", Disponível em: http://www. Bluetooth.com. Acessado em: Jul 2014.

BOOCH, G; RUMBAUGH, J e JACOBSON, I: UML, Guia do Usuário: tradução; Fábio Freitas da Silva, Rio de Janeiro, Campus ,2000.

BRIZZI, P., et al. The ebbits platform: leveraging on the Internet of Things to support meat traceability. In: Proceedings of the EFITA-WCCA-CIGR conference 'sustainable agriculture through ICT innovation', Torino, Italy. 2013. p. 23-27.

CANNATA, A.; GEROSA, M.; TAISCH, M. SOCRADES: A framework for developing intelligent systems in manufacturing. In: Industrial Engineering and Engineering Management, 2008. IEEM 2008. IEEE International Conference on. IEEE, 2008. p. 1904-1908.

CANNATA, A.; KARNOUSKOS, S.; TAISCH, M. Evaluating the potential of a service oriented infrastructure for the factory of the future. In: Industrial Informatics (INDIN), 2010 8th IEEE International Conference on. IEEE, 2010. p. 592-597.

CATARCI, Tiziana et al. Service composition and advanced user interfaces in the home of tomorrow: The sm4all approach. In: Ambient Media and Systems. Springer Berlin Heidelberg, 2011. p. 12-19.

CHANG, Kuor-Hsin. Bluetooth: a viable solution for IoT?[Industry Perspectives]. IEEE Wireless Communications, v. 21, n. 6, p. 6-7, 2014.

CHEN, Guanling, et al. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

CHENG, Doreen Yining. UPnP enabling device for heterogeneous networks of slave devices. U.S. Patent Application No 09/742,278, 2000.

CHRISTIN, Delphine, et al. Wireless sensor networks and the internet of things: selected challenges. Proceedings of the 8th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, 2009, 31-34.

CIOBANU, Radu-Ioan, et al. Big Data Platforms for the Internet of Things. In: Big Data and Internet of Things: A Roadmap for Smart Environments. Springer Internacional Publishing, 2014. p. 3-34.

DATE, C. J. Introdução a Sistemas de Banco de Dados, Tradução da 4ª Edição Americana, p.674, Editora Campus, 1996.

DE SOUZA, JOÃO NUNES. Lógica para ciência da computação. Elsevier Brasil, 2008.

DE SOUZA, Luciana Moreira Sá, et al. Socrades: A web service based shop floor integration infrastructure. In: The internet of things. Springer Berlin Heidelberg, 2008. p. 50-67.

DE, Suparna, et al. Service modelling for the Internet of Things. In: Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on. IEEE, 2011. p. 949-955.

DELICATO, Flávia C.; PIRES, Paulo F.; BATISTA, Thais. Middleware solutions for the Internet of Things. Springer London, 2013.

DELICATO, Flavia C. et al. Towards an IoT ecosystem. In: Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems. ACM, 2013. p. 25-28.

DOM4J. Site oficial. http://dom4j.sourceforge.net/index.html. Último acesso em Nov/2016.

DOMINGO, Mari Carmen. An overview of the Internet of Things for people with disabilities. Journal of Network and Computer Applications, 2012, 35.2: 584-596.

DOUKAS, Charalampos. Building Internet of Things with the ARDUINO. Create Space Independent Publishing Platform, 2012.

EBBITS, "D8.1 Ebbits network Architecture", novembro de 2011, disponível em: http://www.ebbits-project.eu/downloads.php. Acessado em: 20 de julho de 2014.

EIKERLING, Heinz-Josef, et al. Ambient Healthcare Systems Using the Hydra Embedded Middleware for Implementing an Ambient Disease Management System. 2009.

EISENHAUER, Markus; ROSENGREN, Peter; ANTOLIN, Pablo. HYDRA: A development platform for integrating wireless devices and sensors into ambient intelligence systems. In: The Internet of Things. Springer New York, 2010. p. 367-373.

ERGEN, Sinem Coleri. ZigBee/IEEE 802.15. 4 Summary. UC Berkeley, September, 2004, 10.

EVANS, Dave. The internet of things: how the next evolution of the internet is changing everything. CISCO white paper, 2011, 1.

FAN, Chunxiao, et al. A middleware of internet of things (IoT) based on ZigBee and RFID. In: Communication Technology and Application (ICCTA 2011), IET International Conference on. IET, 2011. p. 732-736.

FERREIRA FILHO, Otávio Freitas. Serviços semânticos: uma abordagem Restful. 2011. PhD Thesis. Universidade de São Paulo.

FREEMAN, E. Use a Cabeça–Padrões de Projetos. 2ª Edição. 2007.

FREIRE, Herval. Web Services: A Nova Arquitetura da Internet. Developers Magazine, Rio de Janeiro, edição 73, páginas 24-25, 2002.

GAMA, Kiev; TOUSEAU, Lionel; DONSEZ, Didier. Combining heterogeneous service technologies for building an Internet of Things middleware. Computer Communications, 2012, 35.4: 405-417.

GONÇALVES, E. Desenvolvendo Aplicações WEB com JSP, Servlets, JSF, Hibernate, EJB e AJAX. Editora Moderna, Rio de Janeiro – RJ, 2007.

GORBACH, G. Pursuing manufacturing excellence through Real-Time performance management and continuous improvement. ARC Whitepaper, April, 2006.

GRUBER, Tom. What is an Ontology. WWW Site http://www-ksl. stanford. edu/kst/whatis-an-ontology. html (accessed on 07-09-2004), 1993.

GU, Tao; PUNG, Hung Keng; ZHANG, Da Qing. A service-oriented middleware for building context-aware services. Journal of Network and computer applications, 2005, 28.1: 1-18.

GUINARD, Dominique; TRIFA, Vlad. Towards the web of things: Web mashups for embedded devices. In: Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain. 2009. p. 15.

GUPTA, Rahul; TALWAR, Sumeet; AGRAWAL, Dharma P. Jini home networking: a step toward pervasive computing. Computer, 2002, 35.8: 34-40.

HA, Young-Guk. Dynamic integration of ZigBee home networks into home gateways using OSGI service registry. Consumer Electronics, IEEE Transactions on, 2009, 55.2:

470-476.

HALLSTROM, Jason O., et al. A container-based Approach to Object-Oriented product lines. Journal of Object Technology, 2004, 3.4: 161-175.

HONG, Sungmin, et al. SNAIL: an IP-based wireless sensor network approach to the internet of things. Wireless Communications, IEEE, 2010, 17.6: 34-42.

HOSEK, Jiri, et al. Universal smart energy communication platform. In: Intelligent Green Building and Smart Grid (IGBSG), 2014 International Conference on. IEEE, 2014. p. 1-4.

HRIBERNIK, Karl A., et al. Co-creating the Internet of Things—first experiences in the participatory design of Intelligent Products with Arduino. In: Concurrent Enterprising (ICE), 2011 17th International Conference on. IEEE, 2011. p. 1-9.

HTTPClient. Site oficial. https://hc.apache.org/. Último acesso em Nov/2016.

IETF, "Simple Service Location Protocol (SSLP) for 6LoWPAN", Outubro de 2009, Disponível em: http://tools.ietf.org/html/draft-daniel-6lowpan-sslp-02. Acessado em: 10 de Julho de 2014.

IETF. “The goal of the IETF is to make the Internet work better”. Disponível em: https://www.ietf.org/. Acessado em: Jan/2015.

JAFFEY, Toby. MQTT and CoAP, IoT protocols. 2014. Disponível em: http://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

JAMMES, François. Real time device level service-oriented architectures. In: Industrial Electronics (ISIE), 2011 IEEE International Symposium on. IEEE, 2011. p. 1722-1726.

JARA, Antonio J., et al. IPv6 addressing proxy: Mapping native addressing from legacy technologies and devices to the Internet of Things (IPv6). Sensors, 2013, 13.5: 6687-6712.

JARA, Antonio J.; ZAMORA, Miguel A.; SKARMETA, Antonio. Glowbal IP: An adaptive and transparent IPv6 integration in the Internet of Things. Mobile Information Systems, v. 8, n. 3, p. 177-197, 2012.

JERSEY. Site oficial. https://jersey.java.net/. Último acesso em Nov/2016.

JO, Tae-Wook, et al. A bluetooth-UPnP bridge for the wearable computing environment. Consumer Electronics, IEEE Transactions on, 2008, 54.3: 1200-1205.

JUELS, Ari. RFID security and privacy: A research survey. Selected Areas in Communications, IEEE Journal on, 2006, 24.2: 381-394.

KATASONOV, Artem, et al. Smart Semantic Middleware for the Internet of Things. ICINCO-ICSO, 2008, 8: 169-178.

KEFALAKIS, Nikos, et al. Integrated Development Environment for RFID Development. 2011.

KEFALAKIS, Nikos, et al. Middleware building blocks for architecting RFID systems. In: Mobile Lightweight Wireless Systems. Springer Berlin Heidelberg, 2009. p. 325-336.

KEFALAKIS, Nikos, et al. Supply chain management and NFC picking demonstrations using the AspireRfid middleware platform. In: Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion. ACM, 2008. p. 66-69.

KHRIYENKO, Oleksiy; NAGY, Michal. Semantic Web-driven Agentbased Ecosystem for Linked Data and Services. In: Proceedings of the Third International Conferences on Advanced Service Computing. 2011. p. 25-30.

KIM, Dong-Sung, et al. Design and implementation of home network systems using UPnP middleware for networked appliances. IEEE Transactions on Consumer Electronics, 2002, 48.4: 963-972.

KIRKHAM, T. et al. SOA middleware and automation: Services, applications and architectures. In: Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on. IEEE, 2008. p. 1419-1424.

KORTUEM, Gerd, et al. Smart objects as building blocks for the internet of things. Internet Computing, IEEE, 2010, 14.1: 44-51.

KOSTELNIK, Peter; SARNOVSK, Martin; FURDIK, Karol. The semantic middleware for networked embedded systems applied in the Internet of Things and Services domain. Scalable Computing: Practice and Experience, 2011, 12.3.

KOUCHE, A. E. Towards a wireless sensor network platform for the Internet of Things: Sprouts WSN platform. In: Communications (ICC), 2012 IEEE International Conference on. IEEE, 2012. p. 632-636.

LEE, Choonhwa; HELAL, Sumi. Protocols for service discovery in dynamic and mobile networks. International Journal of Computer Research, 2002, 11.1: 1-12.

LEE, Shinho, et al. Correlation analysis of MQTT loss and delay according to QoS level. In: Information Networking (ICOIN), 2013 International Conference on. IEEE, 2013. p. 714-717.

LI, Li, et al. The applications of Wi-Fi-based wireless sensor network in internet of things and smart grid. In: Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on. IEEE, 2011. p. 789-793.

LIU, Yuxi; ZHOU, Guohui. Key technologies and applications of internet of things. In: Intelligent Computation Technology and Automation (ICICTA), 2012 Fifth International Conference on. IEEE, 2012. p. 197-200.

LONG-GANG, D. I. N. G. Application and Development on Internet of Things about Automobile Based on Bluetooth. Modern Electronics Technique, 2011, 17: 060.

LU, Chia-Wen; LI, Shu-Cheng; WU, Quincy. Interconnecting ZigBee and 6LoWPAN wireless sensor networks for smart grid applications. In: Sensing Technology (ICST), 2011 Fifth International Conference on. IEEE, 2011. p. 267-272.

MAIA, Pedro, et al. A Middleware Platform for Integrating Devices and Developing Applications in e-Health. In: Computer Networks and Distributed Systems (SBRC), 2015 XXXIII Brazilian Symposium on. IEEE, 2015. p. 10-18.

MAINETTI, Luca; PATRONO, Luigi; VILEI, Antonio. Evolution of wireless sensor networks towards the internet of things: A survey. In: Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on. IEEE, 2011. p. 1-6.

MARTINS, Ismael Rodrigues; ZEM, José Luís. ESTUDO DOS PROTOCOLOS DE COMUNICAÇÃO MQTT E COAP PARA APLICAÇÕES MACHINE-TO-MACHINE E INTERNET DAS COISAS1, 2. Americana, v. 3, n. 1, p. 64-87, 2015.

MASUOKA, Ryusuke; PARSIA, Bijan; LABROU, Yannis. TASK COMPUTING–the semantic web meets pervasive computing. In: The Semantic Web-ISWC 2003. Springer Berlin Heidelberg, 2003. p. 866-881.

MATTERN, Friedemann; FLOERKEMEIER, Christian. From the Internet of Computers to the Internet of Things. In: From active data management to event-based systems and more. Springer Berlin Heidelberg, 2010. p. 242-259.

MCGUINNESS, Deborah L., et al. OWL web ontology language overview. W3C recommendation, 2004, 10.10: 2004.

MEHTA KARANKUMAR, D.; MEHTA SHREYA, B.; RAVIYA KAPIL, S. Analysis of TOI (Things of Internet) Industrial Monitoring System on Raspberry pi Platform. Analysis, 2014, 2.11.

MILLER, Eric. An introduction to the resource description framework. Bulletin of the American Society for Information Science and Technology, 1998, 25.1: 15-19.

MIORANDI, Daniele, et al. Internet of things: Vision, applications and research challenges. Ad Hoc Networks, 2012, 10.7: 1497-1516.

MITRA, Nilo; LAFON, Yves. SOAP Version 1.2 Part 0: Primer (Second Edition). In:

http://www.w3.org/TR/soap12-part0/, Abril de 2007.

MONTAVONT, Julien; ROTH, Damien; NOËL, Thomas. Mobile IPv6 in Internet of Things: Analysis, experimentations and optimizations. Ad Hoc Networks, 2014, 14: 15-25.

NAGY, Michal et al. Challenges of middleware for the internet of things. INTECH Open Access Publisher, 2009.

NI, Lionel M., et al. LANDMARC: indoor location sensing using active RFID. Wireless networks, 2004, 10.6: 701-710.

NIELSEN, Jakob. Usability engineering. Elsevier, 1994.

OSGi Alliance, OSGI Specifications. Disponível em: <https://www.osgi.org/developer/specifications/>. Acesso em 08 de julho de 2017.

PERERA, Charith, et al. capturing sensor data from mobile phones using global sensor network middleware. In: Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on. IEEE, 2012. p. 24-29.

PERERA, Charith, et al. Context aware computing for the internet of things: A survey. Communications Surveys & Tutorials, IEEE, 2014, 16.1: 414-454.

PERERA, Charith, et al. Context-aware sensor search, selection and ranking model for internet of things middleware. In: Mobile Data Management (MDM), 2013 IEEE 14th International Conference on. IEEE, 2013. p. 314-322.

PERERA, Charith, et al. Semantic-driven configuration of internet of things middleware. In: Semantics, Knowledge and Grids (SKG), 2013 Ninth International Conference on. IEEE, 2013. p. 66-73.

PIRES, Paulo F. et al. A platform for integrating physical devices in the Internet of Things. In: Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on. IEEE, 2014. p. 234-241.

PIRES, PAULO F. et al. Plataformas para a Internet das Coisas. In: Simpósio Brasileiro de Redes de Computadores (SBRC), 2015.

PIYARE, R.; TAZIL, M. Bluetooth based home automation system using cell phone. In: Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on. IEEE, 2011. p. 192-195.

PRATES, Raquel Oliveira; BARBOSA, Simone Diniz Junqueira. Avaliação de Interfaces de Usuário–Conceitos e Métodos. In: Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação, Capítulo. 2003.

PREDIĆ, Bratislav et al. Exposuresense: Integrating daily activities with air quality using mobile participatory sensing. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on. IEEE, 2013. p. 303-305.

QIN, Weijun et al. RestThing: A Restful Web service infrastructure for mash-up physical and Web resources. In: Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on. IEEE, 2011. p. 197-204.

QU, Liguo, et al. Node design of internet of things based on ZigBee multichannel. Procedia Engineering, 2012, 29: 1516-1520.

RAMLEE, R. A.; TANG, D. H. Z.; ISMAIL, M. M. Smart home system for Disabled People via Wireless Bluetooth. In: System Engineering and Technology (ICSET), 2012 International Conference on. IEEE, 2012. p. 1-4.

RICHARDSON, Leonard; RUBY, Sam. Restful web services. "O'Reilly Media, Inc.", 2008.

ROBIN, Alexandre; BOTTS, Michael E. Creation of Specific SensorML Process Models. White Paper Earth System Science Center (NSSTC). University of Alabama in Huntsville (UAH), 2006.

RODRIGUEZ, Alex. Restful web services: The basics. IBM Developers Works, 2008.

SAIRAM, KVSSSS; GUNASEKARAN, N.; REDD, S. R. Bluetooth in wireless communication. Communications Magazine, IEEE, 2002, 40.6: 90-96.

SALIHBEGOVIC, A., et al. Design of a domain specific language and IDE for Internet of things applications. In: Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on. IEEE, 2015. p. 996-1001.

SAMEH, Ahmed; EL-KHARBOUTLY, Rehab. Modeling Jini-UPnP Bridge using rapide ADL. In: Pervasive Services, 2004. ICPS 2004. Proceedings. The IEEE/ACS International Conference on. IEEE, 2004. p. 237.

SARNOVSKÝ, Martin, et al. First demonstrator of hydra middleware architecture for building automation. In: Proceedings of the Scientific Conference Znalosti. 2008.

SHAH, Pritam Gajkumar; AMBAREEN, Javeria. A Survey of Security Challenges in Internet of Things (IoT) Integration with WSN. In: Australian Journal of Wireless Technologies, Mobility & Security, 2014.

SHELBY, Zach. CoAP: the Web of things protocol. 30 ago. 2014. Disponível em: http://pt.slideshare.net/zdshelby/coap-tutorial>. Acesso em 20 Nov. 2015.

SHRESTHA, Neela; KUBLER, Sylvain; FRÄMLING, Kary. Standardized framework for integrating domain-specific applications into the IoT. In: Future Internet of Things and Cloud (FiCloud), 2014 International Conference on. IEEE, 2014. p. 124-131.

SIEKKINEN, Matti et al. How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4. In: Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE. IEEE, 2012. p. 232-237.

SOLDATOS, John; SERRANO, Martin; HAUSWIRTH, Manfred. Convergence of utility computing with the internet-of-things. In: Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on. IEEE, 2012. p. 874-879.

SONG, Zhexuan, et al. TASK COMPUTING for ubiquitous multimedia services. In: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia. ACM, 2004. p. 257-262.

SONG, Zhexuan; CÁRDENAS, Alvaro A.; MASUOKA, Ryusuke. Semantic middleware for the Internet of Things. In: Internet of Things (IOT), 2010. IEEE, 2010. p. 1-8.

SONG, Zhexuan; LABROU, Yannis; MASUOKA, Ryusuke. Dynamic service discovery and management in task computing. In: Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on. IEEE, 2004. p. 310-318.

SPIESS, Patrik, et al. SOA-based integration of the internet of things in enterprise services. In: Web Services, 2009. ICWS 2009. IEEE International Conference on. IEEE, 2009. p. 968-975.

SUNDMAEKER, Harald, et al. "Vision and challenges for realising the Internet of Things." EUR-OP, 2010.

TAN, Lu; WANG, Neng. Future internet: The internet of things. In: Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on. IEEE, 2010. p. V5-376-V5-380.

TERZIYAN, Vagan; KAYKOVA, Olena; ZHOVTOBRYUKH, Dmytro. Ubi-Road: Semantic Middleware for Cooperative Traffic Systems and Services. International Journal on Advances in Intelligent Systems, 2011, 3.3 and 4: 286-302.

TERZIYAN, Vagan; ZHOVTOBRYUKH, Dmytro; KATASONOV, Artem. Proactive future Internet: smart semantic middleware for overlay architecture. In: Networking and Services, 2009. ICNS'09. Fifth International Conference on. IEEE, 2009. p. 149-154.

Tomcat. Site oficial. http://tomcat.apache.org/. Último Acesso em Nov/2016.

VAJDA, Viliam, et al. The EBBITS Project: An Interoperability platform for a Real-world populated Internet of Things domain. In: Proceedings of the International Conference Znalosti (Knowledge), Technical University of Ostrava, Czech Republic. 2011. p. 317-320.

VERMESAN, Ovidiu, et al. Internet of things strategic research roadmap. Internet of Things-Global Technological and Societal Trends, 2011, 9-52.

VERTAN, Cristina; MERKMALE, RDF-Fortgeschrittene. Resource description framework (rdf). 2004.

WANG, Haolin et al. Transmitting IPv6 packets over Bluetooth low energy based on BlueZ. In: Advanced Communication Technology (ICACT), 2013 15th International Conference on. IEEE, 2013. p. 72-77.

WANT, Roy. An introduction to RFID technology. Pervasive Computing, IEEE, 2006, 5.1: 25-33.

WARRIACH, Ehsan Ullah et al. A tool for integrating pervasive services and simulating their composition. In: Service-Oriented Computing. Springer Berlin Heidelberg, 2010. p. 726-727.

WARRIACH, Ehsan Ullah et al. Heterogeneous device discovery framework for the Smart Homes. In: GCC Conference and Exhibition (GCC), 2011 IEEE. IEEE, 2011. p. 637-640.

WARRIACH, Ehsan Ullah. Design and Implementation of a Middleware Platform for a Smart Home, University of Groningen, The Netherlands, 2013, Disponível em: www.cs.rug.nl/~aiellom/tesi/warriach.pdf, Acessado em: Nov 2014.

WARRIACH, Ehsan Ullah. State of the art: embedded middleware platform for a smart home. Int. J. Smart Home, 2013, 7: 275-294.

WEISER, Mark. The computer for the 21st century. Scientific American, 1991, 265.3: 94-104.

WELBOURNE, Evan, et al. Building the internet of things using RFID: the RFID ecosystem experience. Internet Computing, IEEE, 2009, 13.3: 48-55.

WU, Zhenyu, et al. A web-based two-layered integration framework for smart devices. EURASIP Journal on Wireless Communications and Networking, 2012, 2012.1: 1-12.

XERCES. Site oficial. http://xerces.apache.org/. Último acesso em Nov/2016.

XU, L.; HE, Wu; LI, Shancang. Internet of Things in industries: A survey. 2014.

YANG, L.; YANG, S. H.; PLOTNICK, L. How the internet of things technology enhances

emergency response operations. Technological Forecasting and Social Change, 2013, 80.9: 1854-1867.

YUCE, Mehmet Rasit. Recent wireless body sensors: Design and implementation. In: Microwave Workshop Series on RF and Wireless Technologies for Biomedical and Healthcare Applications (IMWS-BIO), 2013 IEEE MTT-S International. IEEE, 2013. p. 1-3.

ZHANG, Daqiang; YANG, Laurence Tianruo; HUANG, Hongyu. Searching in internet of things: Vision and challenges. In: Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on. IEEE, 2011. p. 201-206.

ZHANG, Weishan; HANSEN, Klaus Marius. An OWL/SWRL Based Diagnosis Approach in a Pervasive Middleware. In: SEKE. 2008. p. 893-898.

ZHAO, Xue Feng. The Application of Bluetooth in the Control System of the Smart Home with Internet of Things. Advanced Materials Research, 2013, 712: 2753-2756.

ZHI-GANG, Liu; WEI, Huang. The design of smart home system based on Wi-Fi. In: Computational Problem-Solving (ICCP), 2012 International Conference on. IEEE, 2012. p. 454-456.

ZHU, Qian, et al. IoT gateway: Bridging wireless sensor networks into internet of things. In: Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on. IEEE, 2010. p. 347-352.

APÊNDICE A – Formulário Utilizado para Obtenção do Perfil dos Participantes

1. Sou do sexo: ( ) Masculino ( ) Feminino

2. Tenho idade entre: ( ) 18 - 25 ( ) 26 - 33 ( ) 34 - 41 ( ) 42 - 49 ( ) Maior que 50

3. Meu nível de escolaridade é:
( ) Fundamental ( ) Médio ( ) Superior
( ) Especialização ( ) Pós-Graduação Stricto senso ( ) Doutorado

4. Minha área de formação é: _____

5. Possui alguma formação em computação? ( ) sim ( ) não, qual _____

6. No meu dia a dia faço uso de dispositivos eletrônicos inteligentes: ( ) Sim ( ) Não, se sim, quais:

( ) Telefone Celular (Smartphone)	( ) TV Inteligente
( ) Condicionador de Ar Inteligente	( ) Tablet
( ) Relógio Inteligente (Smart watch)	( ) Geladeira Inteligente
( ) Notebook	( ) Outros

7. Eu faria uso de dispositivos inteligentes para otimizar atividades do meu dia a dia, tais como,

a) Automatização do controle de temperatura e/ou iluminação do ambiente.
( ) Sim ( ) Não, porque? _____

b) Automatização do controle do consumo de energia, água e gás, permitindo que eu mesmo controle meu consumo.
( ) Sim ( ) Não, porque? _____

c) Configurar o ambiente de acordo com o perfil dos usuários.
( ) Sim ( ) Não, porque? _____

d) Automatização da segurança e monitoramento do ambiente (exemplo: câmera com sensor de movimento e envio de e-mail ou SMS em caso de detecção de movimento):
( ) Sim ( ) Não, porque? _____

APÊNDICE B – Formulário Para Avaliação dos Objetos e Modelos de Comportamento do BDM4IoT

1. Dado um conjunto de dispositivos domésticos inteligentes, enumere as opções que para você melhor representa os dispositivos a seguir, caso contrário informe 0 (zero) para não identificado.

1 	2 	3 	4 
5 	6 	7 	8 
9 	10 	11 	12 
13 	14 	15 	16 
17 	18 		

( ) Condicionador de Ar	( ) Lâmpada	( ) Fogão
( ) Câmera	( ) Televisor	( ) Fechadura Automática
( ) Microondas	( ) Datashow	( ) Computador
( ) Notebook	( ) Relé	( ) Smart Phone/Tablet
( ) Rádio	( ) Cafeteira	( ) Máquina de Lavar
( ) Geladeira	( ) Tomada	( ) Impressora

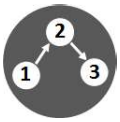
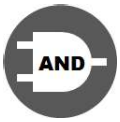
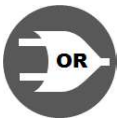
2. Dado um conjunto de dispositivos sensores, enumere as opções que para você melhor representa os dispositivos a seguir, caso contrário informe 0 (zero) para não identificado.

1 	2 	3 	4 	5 
6 	7 	8 	9 	10 
11 	12 			

( ) Umidade	( ) Tensão Elétrica	( ) Temperatura	( ) Gás
( ) Movimento	( ) Magnético	( ) Cardíaco	( ) Fumaça
( ) Impacto	( ) Proximidade	( ) Luminosidade	( ) Pressão

3. Dado um conjunto de objetos de regras de negócio, enumere as opções que para você melhor representa os dispositivos a seguir, caso contrário informe 0 (zero) para não identificado.

Objetos de regras de negócio: *São objetos criados para auxiliar o usuário quanto a definição das leis e regulamentos que definem como o negócio irá funcionar.*

1 		3 	4 	5 
6 	7 	8 	9 	10 
11 	12 	13 	14 	15 
16 				

( ) Salvar em Arquivo
( ) Emitir Som
( ) Baixar do Banco
( ) Salvar nas nuvens
( ) Baixar das Nuvens
( ) Execução Sequencial
( ) Enviar E-mail
( ) Exibir Mensagem
( ) Enviar SMS
( ) Pausar Processo
( ) Desvio Condicional
( ) Operador OR
( ) Salvar em Banco
( ) Cálculos
( ) Operador AND
( ) Repetição Sequencial

4. Dado os objetos a seguir, enumero a segunda coluna de acordo com a primeira com a opção que para você melhor representa os objetos a seguir.

1 

( ) O evento do dispositivo é acionado somente com a intervenção do usuário.

2 

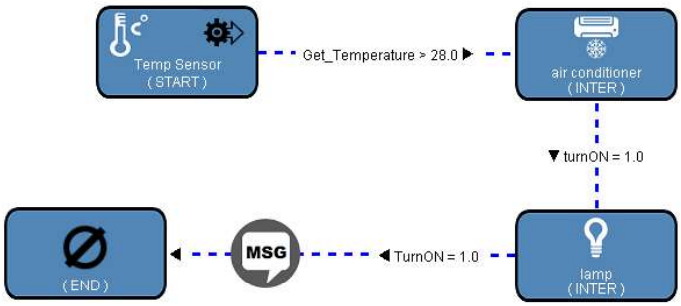
( ) O evento do dispositivo é acionado em data e hora determinada pelo usuário.

3 

( ) O evento do dispositivo é acionado automaticamente, sem a intervenção do usuário.

5. Dado os modelos de comportamento a seguir, selecione a opção que para você melhor define o comportamento representado pelos modelos.

A) Modelo de comportamento 1



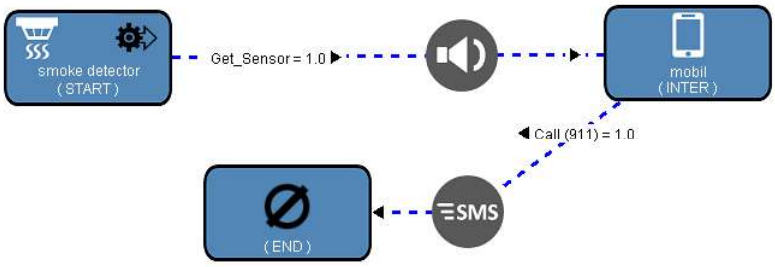
( ) É um modelo de comportamento que quanto o ar condicionado marca 28º uma lâmpada é ligada e o processo finalizado?

( ) É um modelo de comportamento que quanto o sensor de temperatura marca temperatura menor que 28º, o ar condicionado é ligado, uma luz e apagada e o processo finalizado.

( ) É um modelo de comportamento que quando o sensor de temperatura marcar valores maiores que 28º, o ar condicionado e lâmpada são ligados, uma mensagem é enviada e o processo finalizado.

( ) Não consigo definir corretamente o comportamento apresentado pelo modelo.

B) Modelo de Comportamento 2



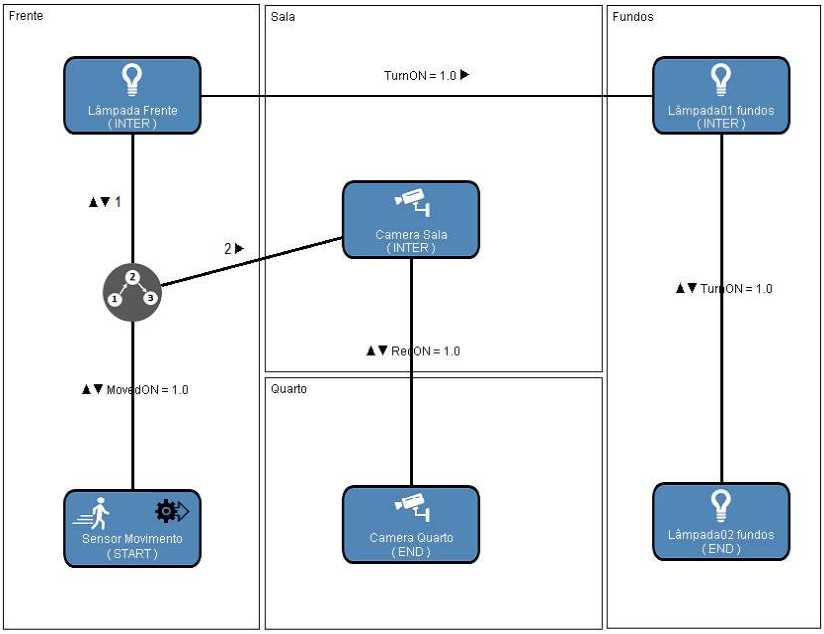
( ) É um comportamento que é iniciado todas os dias as 16:00 para verificar se o sensor de fumaça detecta fumaça no ambiente e em seguida liga para a polícia e o processo é finalizado.

( ) É um comportamento que é iniciado quando o sensor detecta fumaça no ambiente, sem segui um alarme sonoro é executado, uma ligação para polícia e realizada e um SMS é enviado para o usuário informando o ocorrido.

( ) É um comportamento que é iniciado quando o sensor detecta um movimento no ambiente, em seguida o alarme é acionado e uma ligação e mensagem para polícia é realizada.

( ) Não consigo definir corretamente o comportamento apresentado pelo modelo.

C) Modelo de comportamento 3



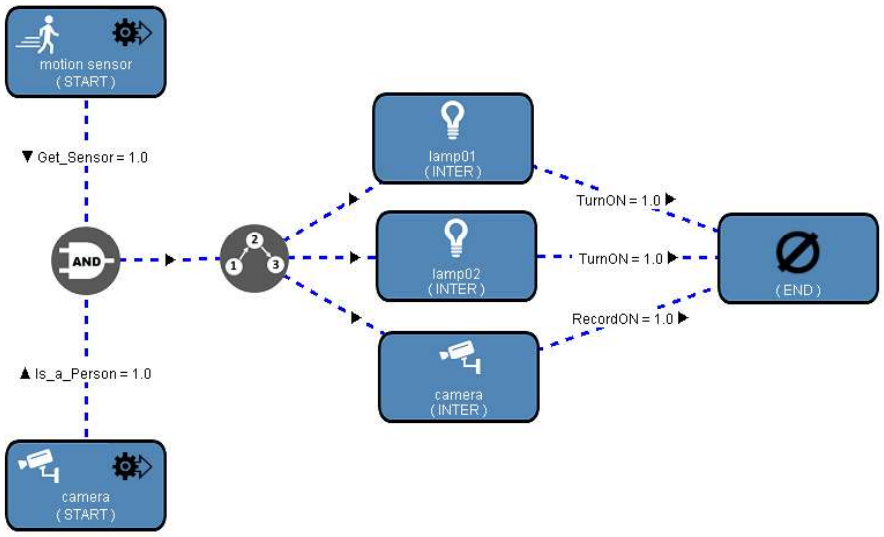
( ) É um modelo de comportamento que é iniciado ao identificar um usuário no ambiente, adequando o ambiente as necessidades do usuário identificado, tais como ligar luzes e controlar a temperatura do ambiente.

( ) É um modelo de comportamento que detecta por meio de câmeras os usuários do ambiente, adequando-o ao seu perfil, como ligar TVs e Câmeras, ajustar automaticamente o ar-condicionados e controlar a luminosidade do ambiente.

( ) É um modelo de comportamento que detecta a presença de um usuário por meio de sensores de movimento, acionando um plano de segurança como, ligar as luzes externas e colocar as câmeras em modo de gravação.

( ) Não consigo definir corretamente o comportamento apresentado pelo modelo.

D) Modelo de comportamento 4



( ) É um comportamento que é iniciado quando o sensor detecta um movimento e a câmera confirma que se trata de uma pessoa, em seguida, sequencialmente duas lâmpadas são acionadas e uma câmera é posta em modo de gravação e por fim o processo é finalizado.

( ) É um comportamento que é iniciado quando o sensor detecta um movimento ou a câmera confirma que se trata de uma pessoa, em seguida, sequencialmente duas lâmpadas são acionadas e uma câmera é posta em modo de gravação e por fim o processo é finalizado.

( ) É um comportamento que é iniciado quando um movimento é detectado, em seguida duas lâmpadas e uma câmera são acionadas simultaneamente e o processo finalizado sem que nenhuma ação seja realizada.

( ) Não consigo definir corretamente o comportamento apresentado pelo modelo.

6. Os modelos de comportamento foram fáceis de compreender e sinto-me confortável em identificar o tipo de comportamento que o modelo representa.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

7. As imagens definidas para os objetos de dispositivos e objetos de regras de negócio expressam sua real finalidade.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

APÊNDICE C – Formulário Para Avaliação do Modelo BDM4IoT quanto a facilidade de aprendizado e para avaliação da ferramenta Êxodo GUI.

1. Utilizando a ferramenta gráfica Êxodo GUI, modele os seguintes cenários de comportamento como se pede.

a. Crie um modelo que represente o seguinte comportamento, utilize os objetos que achar necessário para elaboração: Um sensor localizado no quarto principal, ao marcar uma temperatura maior ou igual a 35º, deve ligar o ar-condicionado em uma temperatura de 20° para redução da temperatura atual.

b. Crie um modelo que represente o seguinte comportamento, utilize os objetos que achar necessário para elaboração: Rotineiramente um usuário chega em casa às 17:30min, em seguida, liga as duas luzes externas da frente e as duas luzes externas de trás da casa, após isso, vai até seu quarto e liga o ar-condicionado na temperatura de 20° e por fim, vai até a sala e liga a TV em seu canal de notícias favorito.

c. Crie um modelo que represente o seguinte comportamento, utilize os objetos que achar necessário para elaboração: Para sua segurança um usuário gostaria de modelar o seguinte comportamento, um sensor ao detectar o movimento na parte externa da casa, deve ligar as 4 luzes externas, acionar um alarme sonoro, colocar as 5 cameras localizadas nos seguintes cômodos (2 na parte externa, 1 na sala, 1 no quarto e 1 na cozinha) em modo de gravação e mandar uma mensagem de emergência para o 911, além disso, o sistema de travas elétricas deve ser acionados para as duas portas principais de entrada, localizadas na sala e na cozinha. Utilize o objeto **ENV** para representar os ambientes supracitados.

d. Elabore um modelo de comportamento que faça uso dos seguintes objetos de regras de negócio: Desvio Condicional, Repetição Sequencial, Operador AND e Operador OR.

2. Com base em sua experiência quanto ao uso do modelo BDM4IoT, marque a opção que melhor representa sua opinião quanto aos critérios a seguir:

a) A ferramenta é intuitiva e pude rapidamente assimilar como fazer uso dos objetos para criação dos meus modelos.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

b) descobri sem grandes problemas como manipular os objetos como, mover, redimensionar, deletar e relacionar diferentes objetos.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

c) não tive dificuldades em descobrir como acessar as propriedades dos objetos a fim de configurar o comportamento dos mesmos.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

d) as informações apresentadas nas telas de propriedades dos objetos são claras e de fácil entendimento.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

APÊNDICE D – Formulário Utilizado Para Comparação da Ferramenta Êxodo GUI e do Modelo BDM4IoT com Soluções Existentes.

1. Utilizando a ferramenta ASPIRE Editor e OpenHAB, tente elaborar os seguintes comportamentos.

Um sensor, localizado no quarto principal, ao marcar uma temperatura maior ou igual a 35º, deve ligar o ar-condicionado em uma temperatura de 20° para redução da temperatura atual e desligar as luzes do quarto caso as mesmas estejam ligadas. Caso seja detectada, por meio de sensores de pressão espalhados na cama, a presença de um usuário, as cortinas deverão baixar, o condicionador de ar colocado na temperatura de 23° e o som ambiente deverá ser ligado no volume 10.

2. Com base em sua experiência com a utilização da ferramenta *ASPIRE Editor*, marque a opção que melhor representa sua opinião quanto aos critérios a seguir:

d) A ferramenta é intuitiva e pude rapidamente assimilar como fazer uso dos objetos para criação dos meus modelos.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

e) Descobri sem grandes problemas como manipular os objetos como, mover, redimensionar, deletar e relacionar diferentes objetos.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

f) Não tive dificuldades em descobrir como acessar as propriedades dos objetos a fim de configurar o comportamento dos mesmos.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

g) As informações apresentadas nas telas de propriedades dos objetos são claras e de fácil entendimento.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

3. Com base em sua experiência com a utilização da ferramenta *OpenHAB*, marque a opção que melhor representa sua opinião quanto aos critérios a seguir:

d) A ferramenta é intuitiva e pude rapidamente assimilar como fazer uso dos objetos para criação dos meus modelos.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

e) Descobri sem grandes problemas como manipular os objetos como, mover, redimensionar, deletar e relacionar diferentes objetos.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

f) Não tive dificuldades em descobrir como acessar as propriedades dos objetos a fim de configurar o comportamento dos mesmos.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

g) As informações apresentadas nas telas de propriedades dos objetos são claras e de fácil entendimento.

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Não sei opinar

( ) Concordo Parcialmente ( ) Concordo Totalmente

4. Crie um modelo para o comportamento descrito na atividade 1, utilizando o modelo BPMN e em seguida o modelo BDM4ioT. Após isso responsa o que se pede.

a) Dos modelos utilizados, qual para você tem melhor capacidade para representação dos objetos de dispositivos em um modelo de comportamento de ambientes para IoT (*Somente usuários especialistas*).

( ) DBM4IoT ( ) BPMN

b) Dos modelos utilizados, qual para você tem melhor capacidade para representação das regras de negócio em um modelo de comportamento de ambientes para IoT (*Somente Usuários Especialistas*).

( ) DBM4IoT ( ) BPMN

c) Dos modelos de comportamento criados, qual possui melhor capacidade de leitura e interpretação. (*Somente usuários não especialistas*).
( ) BDM4IoT ( ) BPMN