UNIVERSIDADE FEDERAL DA BAHIA

TRABALHO DE GRADUAÇÃO

Detecção inteligente de efeitos colaterais indesejáveis na Internet das coisas - um estudo de caso no Home Network System

Heron Sanches Gonçalves Pires Ferreira

Programa de Graduação em Ciência da Computação

Salvador
31 de outubro de 2016

HERON SANCHES GONÇALVES PIRES FERREIRA

DETECÇÃO INTELIGENTE DE EFEITOS COLATERAIS INDESEJÁVEIS NA INTERNET DAS COISAS - UM ESTUDO DE CASO NO HOME NETWORK SYSTEM

Este Trabalho de Graduação foi apresentado ao Programa de Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Profa. Dra. Daniela Barreiro Claro
Co-orientador: Roberto Cerqueira Figueiredo

Salvador
31 de outubro de 2016

Ficha catalográfica.

Ferreira, Heron Sanches Gonçalves Pires

Detecção inteligente de efeitos colaterais indesejáveis na Internet das coisas - um estudo de caso no Home Network System/ Heron Sanches Gonçalves Pires Ferreira– Salvador, 31 de outubro de 2016.

39p.: il.

Orientadora: Profa. Dra. Daniela Barreiro Claro.
Co-orientador: Roberto Cerqueira Figueiredo.
Monografia (Graduação)– UNIVERSIDADE FEDERAL DA BAHIA, INSTITUTO DE MATEMÁTICA, 31 de outubro de 2016.

"1. Efeitos Colaterais Indesejáveis. 2. Internet das Coisas. 3. Home Network System. 4.Dispositivos como serviços Web RESTFul. 5.DECORATE. 6.Cross-validation".
I. Claro, Daniela Barreiro. II. .
III. UNIVERSIDADE FEDERAL DA BAHIA. INSTITUTO DE MATEMÁTICA. IV. Título.

CDD 005.13307

À minha mãe.

AGRADECIMENTOS

Obrigado minha mãe, por sua dedicação durante todos estes anos. Obrigado minha tia Suzi, por ter me dado oportunidade de estudo e sempre está disposta a me ajudar em qualquer situação. Obrigado as forças boas que vêem me guiando e me ajudando. Obrigado Daniela e Roberto pela orientação prestada para este trabalho.

''Fale para ele não desistir''.

—MENSAGEM ESPÍRITA.

RESUMO

Diversos dispositivos têm sido incorporados em ambientes domésticos através dos *Home Server* (HS) com o intuito de automatizar as tarefas cotidianas de uma casa inteligente. Estes dispositivos podem ser disponibilizados como serviços Web RESTFul. Porém, muitos destes dispositivos independentes não conseguem atender aos requisitos dos usuários. Assim, há a necessidade de composição destes com outros dispositivos para prover funcionalidades que atendam a esses requisitos. Quando existe uma composição de dispositivos, o comportamento de pelo menos um destes pode provocar interação de características com efeitos colaterais indesejáveis. Assim, o presente trabalho propõe detectar efeitos colaterais indesejáveis entre dispositivos utilizando-se do *ensemble* DECORATE. Para isto, observou-se o comportamento físico e as mensagens trocadas entre os dispositivos. Através do conjunto de dados obtidos realizou-se um processo classificatório e implantou um classificador no FIM (*Feature Interaction Manager*) do HS, mostrando que e possıvel detectar efeitos colaterais indesejaveis entre dispositivos.

Palavras-chave: dispositivos, serviços Web RESTFul, *Home Server*, *Feature Interaction Manager*, efeitos colaterais indesejáveis, interação de características, composição de dispositivos, *ensenble*, DECORATE, processo classificatório, classificador

ABSTRACT

Many devices have been put on domestics environments through the Home Servers with the deal of automate the daily tasks of a smart home. These devices may be available how Web RESTFul services. But, some devices when alone do not attend the user specifications. Then, there is the need to composition with the other devices to provide features that are accord to them specifications. When there is a device's composition, the behave of at least one of them can cause feature interaction with undesirable side effects. So, this work proposes detect undesirable side effects between devices using the ensemble DECORATE. For it, was realized observation of the physical behave and the messages changed by these devices. Through the data collected was made a classification process and put a classifier into FIM (Feature Interaction Manager), showing that is possible detect undesirable side effect between devices.

Keywords: devices, Home Servers, Web RESTFul services, device's composition, feature interaction, undesirable side effect, Feature Interaction Manager, ensenble, DECORATE, classification process, classifier

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Uma breve introdução sobre do que se trata esta monografia e a maneira como o texto está organizado.

INTRODUÇÃO

A Internet nesta última década tem contribuído de forma significativa na economia e sociedade, deixando como legado uma notável infraestrutura de rede de comunicação. O seu maior disseminador nesse período vem sendo *World Wide Web*(WWW), o qual permite o compartilhamento de informação e mídia de forma global (Chandrakanth et al., 2014).

A Internet está se tornando cada vez mais persistente no cotidiano, devido, por exemplo, ao crescente número de usuários de dispositivos móveis, os quais possuem tecnologias de conexão com a Internet, as quais cada dia tornam-se mais acessíveis (Chandrakanth et al., 2014).

Em 2010 havia aproximadamente 1,5 bilhão de PCs conectados a Internet e mais que 1 bilhão de telefones móveis (Sundmaeker et al., 2010). Segundo Gartner[1], 6,4 bilhões de coisas estarão conectadas até o final de 2016 e, em 2020 esse número atingirá cerca de 20,8 bilhões. A previsão de (Sundmaeker et al., 2010), a qual dizia que a denominada Internet dos PCs seria movida para o que se chama de Internet das Coisas fica então mais evidente neste atual cenário.

A ideia básica da *Internet of things (IoT)*, traduzido para o português como Internet das Coisas é a presença pervasiva de uma variedade de "coisas ou objetos", tais como RFID tags, sensores, telefones móveis, dentre outros. Os quais, através de esquemas de endereçamento único são capazes de interagir com os outros e cooperar com seus vizinhos para alcançar um objetivo em comum (Atzori et al., 2010).

Apesar da grande potencialidade da Internet das Coisas, a qual gerou em 2015 um faturamento em torno de 130,33 bilhões de dólares americanos e tem prospecção de chegar até 883.55 bilhões em 2020[2], ainda existem muitos desafios a serem vencidos, tais como segurança da informação, armazenamento e processamento de grande quantidade de dados, dentre outros. Adentrando um pouco em um dos desafios da IoT, o da disponibilidade

[1] ⟨http://www.gartner.com/newsroom/id/3165317⟩

[2] http://www.marketsandmarkets.com/Market-Reports/iot-application-technology-market-258239167.html

de uma interface de comunicação (acesso aos serviços e informações dos dispositivos) e programação comum aos objetos, pode-se dizer que a falta desta padronização faz com que se torne oneroso o desenvolvimento de aplicações para o objeto. Mais difícil ainda é prover uma única funcionalidade ou serviço com a composição dos diversos objetos. Para diminuir a dificuldade deste cenário, pode-se disponibilizar os dispositivos como serviços Web, desta forma pode-se utilizar os protocolos Web como linguagem comum de integração dos dispositivos a Internet (Franca et al., 2011).

Diante deste cenário de crescente adição de dispositivos a IoT, existe outro problema que também merece atenção. Muitos dispositivos quando de forma isolada, não conseguem atender aos requisitos dos usuários finais (humano ou outro dispositivo). Assim, há a necessidade de composição destes com outros dispositivos para prover funcionalidades que atendam a esses requisitos. Quando existe uma composição de dispositivos, o comportamento de pelo menos um destes pode provocar interação de características com efeitos colaterais indesejáveis, que é quando esta composição leva a um estado inconsistente do sistema, um sistema instavel ou dados imprecisos.

Assim, o presente trabalho propõe detectar efeitos colaterais indesejáveis entre dispositivos utilizando-se do *ensemble* DECORATE. Para isto, é proposto um estudo de caso através de um cenário em um *Home Network System*, onde observou-se o comportamento físico e as mensagens trocadas entre os dispositivos. A partir do conjunto de dados obtidos realizou-se um processo classificatório e implantou um classificador com alto grau de generalização no *Feature Interaction Manager* do *Home Server*, mostrando que e possıvel detectar efeitos colaterais indesejaveis entre dispositivos.

O restante deste trabalho está organizado da seguinte maneira:

- Capítulo 2: Traz todo embasamento teórico necessário para compreender o estudo realizado neste trabalho.
- Capítulo 3: Neste capítulo é apresentado o estudo de caso realizado neste TCC, o qual verificou se é possível detectar efeitos colaterais indesejáveis entre dispositivos.
- Capítulo 4: Sintetiza a investigação e experimentos realizados no estudo de caso deste trabalho, assim como demonstra o resultado alcançado.

Capítulo 2

Este capítulo tem como objetivo fundamentar as bases necessárias dos campos de estudos utilizados nesta monografia.

FUNDAMENTAÇÃO TEÓRICA

2.1 INTERNET DAS COISAS

A ideia básica da *Internet of things (IoT)*, traduzido para o português como Internet das Coisas é a presença pervasiva de uma variedade de "coisas ou objetos", tais como RFID tags, sensores, telefones móveis, dentre outros. Os quais, através de esquemas de endereçamento único são capazes de interagir com os outros e cooperar com seus vizinhos para alcançar um objetivo em comum (Atzori et al., 2010). Outros exemplos de "coisas ou objetos"podem ser pessoas, geladeiras, televisores, veículos, roupas, medicações, livros, passaportes, contanto que possam ser identificadas unicamente e possam se comunicar com as outras coisas e/ou possam ser acessados remotamente por humanos.

Dentre as diversas definições de IoT pode-se citar duas, a primeira define de maneira mais geral (Internet. . . , 2008), enquanto que a segunda especifica melhor como deve ser o cenário ideal da Internet das Coisas (Sundmaeker et al., 2010).

1. *Internet of Things* significa rede mundial de objetos unicamente endereçáveis e interconectados, seguindo os protocolos dos padrões de comunicação.

2. IoT é parte integrante da futura Internet e pode ser definida como uma infraestrutura de rede global dinâmica com capacidades de autoconfiguração baseada nos padrões e interoperabilidade dos protocolos de comunicação onde coisas físicas e virtuais têm identidade, atributos físicos, personalidade virtual, usam interfaces inteligentes e, são integradas dentro da rede de informações.

Diante deste cenário, pode-se citar exemplos de aplicações na IoT para diversos domínios, tais como, logística e transporte; cuidados com a saúde; ambiente inteligente (casa (seção 2.3), escritório); (Atzori et al., 2010) verificação de procedência alimentícia; dentre outros. Seguem dois exemplos de aplicações, uma na área de cuidados com a saúde e outra na verificação de procedência de alimentos, respectivamente.

- Dispositivos implantáveis com capacidade de comunicação sem fio podem ser utilizados para armazenar registros sobre a saúde de um paciente em situações de risco e podem ser decisivos para salvar a vida do paciente. A capacidade de ter acesso a essas informações nessas circunstâncias fazem com que hospitais possam saber de imediato como tratar um paciente que esta a caminho. Esta possibilidade é especialmente útil para pessoas com diabetes, câncer, problemas de coração na artéria coronária, doenças do pulmão, assim como as pessoas com implantes médicos complexos, tais como, marcapassos, tubos, transplantes de órgãos e aqueles que podem ficar inconscientes e incapazes de comunicar-se durante uma operação (Weber e Weber, 2010).

- Rastreabilidade de produtos alimentícios ajudam os usuários a verificar a origem de um produto, assim como informações de composição química, dentre outros. Mas também previne de doenças não desejadas. Por exemplo, avisos atuais sobre a mercadoria em questão podem ser disponibilizados à medida que o produto sai da origem e vai passando para os outros níveis de consumo, desta forma os consumidores podem evitar o contágio de doenças como gripe aviária e, encefalopatia espongiforme bovina (EEB), mais conhecida como doença da vaca louca (Weber e Weber, 2010).

2.1.1 Serviços Web

Segundo (Dustdar e Schreiner, 2005), um serviço Web é um sistema identificado por uma URL[1], no qual suas interfaces públicas são definidas e descritas usando XML[2] e, suas definições são padronizadas e podem ser descobertas por outros sistemas. Sistemas então podem interagir com os serviços Web utilizando suas definições e descrições. Mensagens XML são transmitidas seguindo os protocolos padrão (definidos por W3C[3]) da Internet para que haja tal interação.

Serviços Web surgiram tendo como principal foco o reúso de aplicações existentes (dentre as quais incluíam código fonte legado) para que pudessem se integrar de forma leve com outras aplicações. Geralmente essa integração tinha como referência o desejo de novas formas de compartilhamento dos serviços ao longo das diversas linhas do negócio ou entre parceiros. Como exemplo, considere o cenário da Figura 2.1 o qual demonstra a utilização de diferentes serviços (*Credit service, Inventory service, Billing service, Shipmente service*) para prover um único serviço, pedido de uma ou mais mercadorias, ambiente comum do E-Commerce.

Serviços Web possuem algumas características:

1. Podem ser ditos *stateless* (sem estado) ou *stateful* (com estado). Se um serviço pode ser invocado repetidamente sem ter que manter um contexto ou estado este

[1] Acrônimo para Uniform Resource Locator e é uma referência (um endereço) para um recurso na Internet.⟨https://docs.oracle.com/javase/tutorial/networking/urls/definition.html⟩

[2] Extensible Markup Language (XML) é uma simples e flexível linguagem de marcação utilizada para codificar documentos através de regras produzidas por humanos, as quais podem ser manipulados (compreendidas) por máquinas (W3C, 2004a), (W3C, 2004b)

[3] ⟨https://www.w3.org/⟩

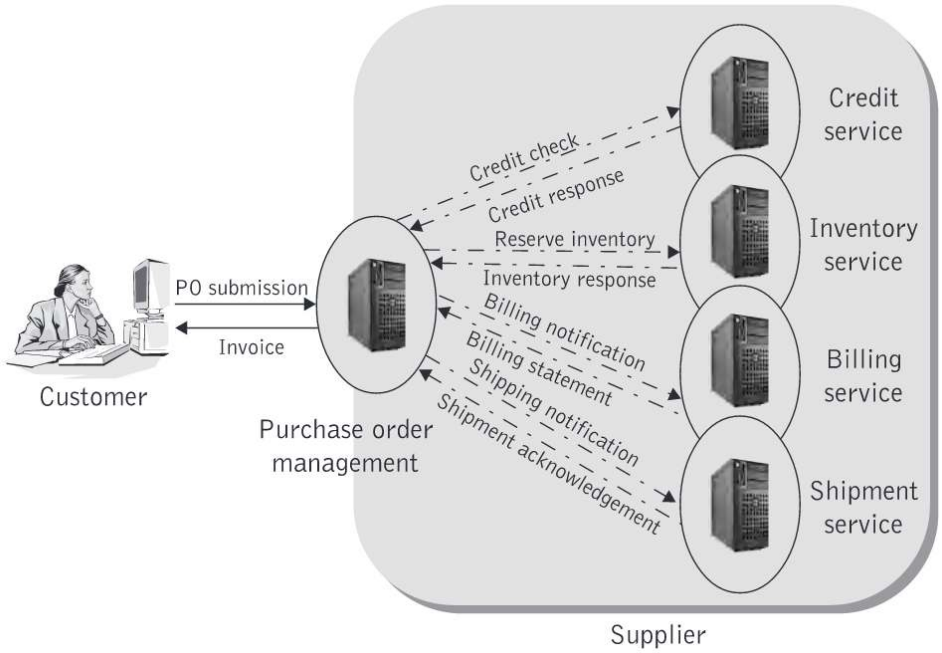

Figura 2.1: Utilização de diferentes serviços Web para prover o serviço de pedido de um E-Commerce. (Papazoglou, 2008b)

é denominado *stateless*. Como exemplo de serviço *stateless* pode-se citar a recuperação de informação de um sensor de temperatura, pois não precisa de nenhum tipo de memória para manter o que aconteceu durante as requisições ao serviço. Já os serviços que precisam manter seus contextos preservados de uma invocação para a próxima requisição, estes são *stateful*. Por exemplo, um usuário quando chama um serviço (Levar compras) de um *Home Network Service* (ver seção 2.3), este então começa sua execução: um veículo (um serviço) leva a carga até uma determinada área próxima ao elevador (um serviço) e, uma garra mecânica (um serviço) pega a carga e coloca dentro do elevador. Observe que o serviço Levar compras é a composição de diferentes serviços e não pode ser chamado e executado diversas vezes continuamente sem a necessidade de manter seu contexto entre os diferentes serviços envolvidos, pois o ambiente não tem mais de um carro, mais de uma garra e elevador. Então a menos que este já tenha terminado toda a sua execução, poderia ser chamado e executado novamente. Por tal motivo este serviço mantêm um estado (em execução ou sem trabalho) o qual depende dos estados dos serviços envolvidos (Papazoglou, 2008b).

2. Podem ser ditos com baixo nível de acoplamento (*loose coupling*) quando os serviços interagem uns com os outros e utilizam-se as tecnologias padrões da Internet, permitindo dessa forma construir pontes entre sistemas que de outra forma exigiriam grandes esforços de desenvolvimento de software. O termo acoplamento indica o nível de dependência entre serviços (Papazoglou, 2008b).

3. Podem ser ditos síncronos e assíncronos. Quando síncrono, clientes realizam suas requisições e ficam sempre aguardando uma resposta. O cliente é dependente de tal resposta para continuar sua computação. Se uma operação for incapaz de ser completada, o resto da computação é impedida de continuar. Enquanto que nos assíncronos o cliente não necessita aguardar uma resposta para continuar a execução de sua aplicação. Quando um cliente invoca um serviço assíncrono, o cliente normalmente envia um documento inteiro, como uma ordem de pagamento ou, uma lista de itens de um carrinho de compra com o tipo de pagamento e endereço de entrega. O serviço aceita o documento inteiro e processa-o, podendo retornar ou não uma mensagem de resultado. A resposta do serviço, se existir alguma, pode acontecer horas ou talvez dias depois (Papazoglou, 2008b).

Serviços Web possuem baixo nível de acoplamento e utilizam padrões para oferecer suas funcionalidades, pois tais serviços tem a intenção de prover comunicação a diferentes tipos de aplicações, que possivelmente são executadas em diferentes plataformas. A *Web Service Description Language*(WSDL) usa o formato XML para descrever os métodos oferecidos por um serviço Web, incluindo parâmetros de entrada e saída, tipos de dados e, protocolo de transporte utilizado (normalmente o HTTP). Além de informações referente ao provedor do serviço, tais como, endereço e contato da empresa desenvolvedora do serviço. O *Simple Object Access Protocol*(SOAP, seção 2.1.2) é utilizado para as trocas de mensagens (formatadas em XML) entre as entidades envolvidas no modelo de serviço Web citado(Dustdar e Schreiner, 2005), como pode ser visto na Figura 2.2 e explicado logo abaixo. Esse tipo de serviço Web é chamado de serviço Web baseado em SOAP (Belqasmi et al., 2011).

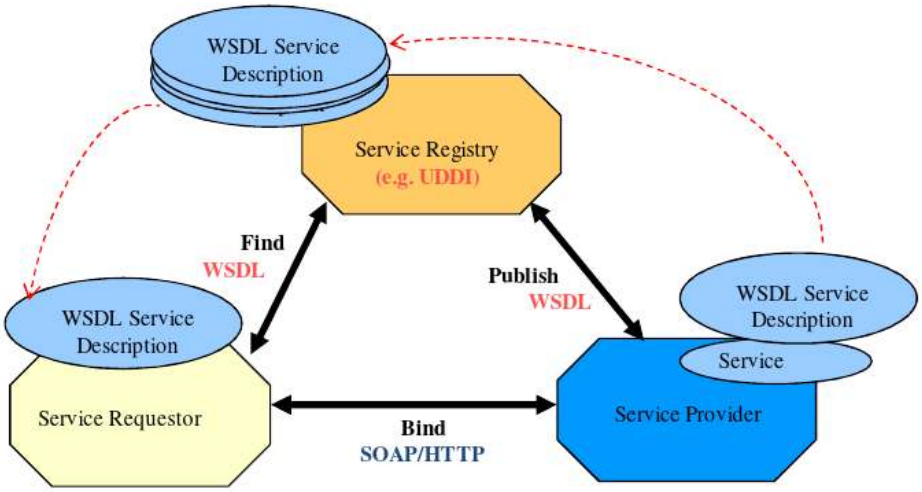

Figura 2.2: Modelo de serviço Web baseado em SOAP. (Belqasmi et al., 2011)

- Provedor (Service Provider) - cria e oferece o serviço Web, este precisa descrever o serviço em um formato padrão, neste caso WSDL e, publica-o (*Publish*) no Registro de serviço.

- Registro de serviço (Service Registry) - contém a descrição publicada pelo Provedor. O registro de serviço mais utilizado é o *Universal Description, Discovery and Integration* (UDDI), este em suas especificações define um conjunto de *Application Program Interfaces*(APIs) tanto para publicação, quanto para descoberta (Belqasmi et al., 2011).

- Consumidor (Service Requestor) - obtém informações do registro, *Find*, e utiliza a descrição do serviço capturada para invocar (*Bind*) o serviço Web.

A descrição realizada da Figura 2.2 foi na verdade a descrição das 3 regras fundamentais do Service Oriented Architecture (SOA), que é um caminho lógico para projetar sistemas de software, os quais podem prover serviços tanto para o usuário final de aplicações quanto para outros serviços distribuídos na rede, através de interfaces que possam ser publicadas e descobertas (Papazoglou, 2008b).

Existe também outro tipo de abordagem de serviços Web, baseada nos princípios arquiteturais REST (ver seção 2.1.3).

2.1.2 SOAP

SOAP é um protocolo leve que foi desenvolvido com a intenção de prover troca de mensagens estruturadas em um ambiente descentralizado e distribuído. Este usa a tecnologia XML para definir um framework extensível de mensagens, o qual permite a construção de mensagens que podem ser trocadas através de uma variedade de protocolos subjacentes. O framework foi desenvolvido para ser independente de modelo de linguagem de programação ou qualquer outra implementação semântica (Gudgin et al., 2007). Diz-se que é um protocolo leve pelo fato de somente receber e enviar pacotes de protocolos de transportes (por exemplo, HTTP) e, processar as mensagens XML, quando contrastados com os protocolos ORPC (Papazoglou, 2008a). A figura 2.3 seguinte mostra como o SOAP atua no modelo de serviço Web baseado em SOAP (Figura 2.2). Nesta, o *SOAP-based middleware* converte as chamadas de procedimento de/para mensagens XML que são enviadas através do HTTP ou outro protocolo.

Em outras palavras, SOAP é um protocolo (da camada de aplicação) utilizado para transferir mensagens entre instâncias de serviços descritas por interfaces WSDL (ver Figura **??**).

2.1.3 Serviços Web RESTFul

Segundo (Heffelfinger, 2014), Representational State Transfer (REST) é um estilo de arquitetura no qual serviços Web são vistos como recursos e podem ser identificados por Uniform Resources Identifiers (URIs). Serviços Web que são desenvolvidos usando REST são conhecidos como RESTful web services.

Os principais princípios do estilo arquitetural REST são: endereçamento global através de identificação dos recursos, interface uniforme compartilhada por todos os recursos, interações *stateless* entre os serviços e, mensagens auto-descritivas e *hypermedia* como um mecanismo para descoberta descentralizada de recursos por referência (Pautasso, 2014), conforme abaixo.

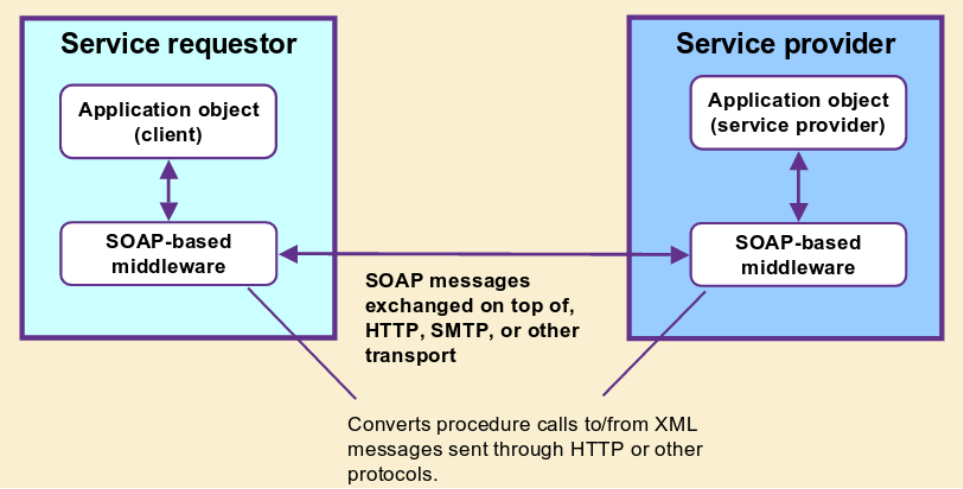

Figura 2.3: SOAP no modelo Modelo de serviço Web baseado em SOAP. (Papazoglou, 2008a)

1. Identificação dos recursos - todos os recursos que são publicados por um serviço Web deveriam ser disponibilizados por um único e estável identificador global (Pautasso, 2014). A exemplo das URIs (Franca et al., 2011).

2. Interface uniforme - todos os recursos interagem através de uma interface uniforme, a qual prover um conjunto de métodos pequeno, genérico e funcionalmente suficiente para suportar todas as possíveis interações entre os serviços. Cada método tem uma semântica bem definida em relação aos efeitos que causará no estado do recurso. No contexto da Web e do protocolo HTTP que é utilizado, "Interface uniforme" pode ser alcançado com os seus métodos (GET, PUT, DELETE, POST, HEAD, OPTIONS, dentre outros) os quais podem ser aplicados para todos os identificadores dos recursos Web (URIs) (Pautasso, 2014).

3. Interações *stateless* - os serviços não podem estabelecer sessões permanentes entre eles. Isto assegura que as requisições para um recurso sejam independentes entre si. No final de cada interação, não há estados compartilhados entre clientes e servidores. Requisições podem resultar em uma mudança de estado do recurso, mas o novo estado é imediatamente visível para todos clientes (Pautasso, 2014).

4. Mensagens auto-descritivas - Serviços interagem através de requisição e mensagem de resposta, que contem tanto os dados (representações dos recursos) e correspondente *meta-data*. As representações podem variar de acordo com o contexto do cliente, interesses e habilidades. Por exemplo, um cliente *mobile* pode obter uma representação do recurso que exige pouco consumo de banda de dados. Da mesma forma, um navegador pode requisitar a representação de uma página Web em uma linguagem específica, de acordo com suas preferências. Esta característica aumenta de maneira significativa o grau de interoperabilidade, pois um cliente pode dinamicamente negociar o mais apropriado formato de representação com o recurso

(também conhecido como media type), em vez de ser forçado a sempre trabalhar com uma determinada representação do recurso. As requisições e mensagens de respostas também devem conter explicitamente *meta-data* sobre sua representação, desta maneira os serviços não precisam assumir algum tipo de acordo de sobrecarga no sentido de como tal representação seria analisada, processada e entendida (Pautasso, 2014).

5. *Hypermedia* - Recursos podem ser relacionados entre si. *Hypermedia* embute referências a outros recursos relacionados ou dentro de suas representações ou em sua correspondente *metadata*. Clientes então podem descobrir os *hyper-links* dos recursos relacionados ao processar suas representações e escolher seguir o link. Como exemplificado em (Franca et al., 2011), um sistema de uma instituição que possui um recurso lista_cursos (lista todos os cursos da instituição), este pode oferecer os *hyper-links* que representam cada recurso que representa um determinado curso. *Hypermedia* auxilia na descoberta descentralizada de recursos e também pode ser utilizada para descoberta e descrição de protocolos de interação entre os serviços. Pelo fato deste ser o princípio menos utilizado pelas APIs que se dizem ser REST-Ful, algumas vezes as APIs disponibilizadas por serviços Web que além das outras restrições contemplam esta, são também chamadas de *Hypermedia APIs* (Pautasso, 2014).

2.1.4 Dispositivos como serviços Web RESTFul

Como abordado em 2.1 um dos desafios referentes a visão da IoT é sua interoperabilidade e, como possível solução se pode disponibilizar os dispositivos como serviços Web, neste caso seguindo os príncipios REST. Desta maneira os dispositivos podem interagir entre si e com outros sistemas na Web.

A disponibilização dos dispositivos como serviços Web RESTFul vêm sendo empregada através de duas abordagens. Na primeira, quando os dispositivos têm recursos suficientes (memória, processamento e largura de banda de rede) servidores Web são embarcados nos próprios dispositivos e estes são disponibilizados como recursos REST. Enquanto que na segunda, quando uma coisa não possui recursos suficientes para tal propósito, utiliza-se de outro dispositivo, por exemplo, um [4]Raspberry Pi ou qualquer outro controlador com recursos suficientes para instalação e execução de um servidor Web. Este então atua como um intermediador entre o objeto e a Internet (Franca et al., 2011).

Adotando esse padrão, os dispositivos podem ter suas propriedades disponíveis através de qualquer navegador, sem a necessidade de instalação de nenhum programa ou driver adicional, como pode ser exemplificado na Figura 2.4. Além disto, mashups físicos[5] podem ser construídos com muito menos esforço do que as existentes abordagens, minimizando a barreira para o desenvolvimento de aplicações com dispositivos (Guinard e Trifa, 2009).

[4]https://www.raspberrypi.org/products/

[5]Aplicativos criados a partir da composição de dados e serviços de dispositivos físicos com outros recursos Web.

Pelo fato das coisas serem disponibilizadas como serviços Web, ainda pode-se utilizar-se dos outros recursos disponíveis na Web, a exemplo de sistemas de cache, balanceamento de carga, indexação e pesquisa (Franca et al., 2011). Assim então promovendo a visão da Internet das Coisas.

Figura 2.4: Dispositivo sensor de obstáculo do elevador disponibilizado como serviço. Faixa de detecção compatível com a largura do elevador utilizado na maquete do cenário (Figura 3.3)

No cenário da IoT os princípios REST têm sido amplamente aplicados na integração dos dispositivos inteligentes a Web, pois esses princípios parecem ser mais adequados para dispositivos com poucos recursos de hardware (Franca et al., 2011). Para deixar claro os motivos da adoção do REST para disponibilização de dispositivos como serviços Web ao invés dos serviços Web baseados em SOAP, será realizada uma comparação a seguir.

Uma das diferenças entre o REST e WS-* SOAP está na forma como o protocolo HTTP é empregado. Com REST, o HTTP é utilizado para prover interfaces uniformes onde cada método (GET, PUT, DELETE, POST, HEAD, OPTIONS, dentre outros) tem uma semântica bem definida em relação aos efeitos que causará no estado do recurso (Franca et al., 2011), enquanto que nos WS-* SOAP cada serviço tem seu próprio conjunto de operações explicitamente declarados dentro de um documento WSDL (Pautasso, 2014). O SOAP utiliza o HTTP como um protocolo de transporte (HTTP é um protocolo da camada de aplicação de rede). As mensagens SOAP são adicionadas ao corpo do HTTP. Nos WS-* SOAP o método POST do HTTP é utilizado na troca de mensagens entre cliente e servidor e, a informação sobre qual funcionalidade deve ser executada está presente na mensagem SOAP e não na requisição HTTP. Os serviços Web baseados em SOAP utilizam a Web como meio de troca de mensagens, enquanto os serviços Web RESTFul são parte da Web, ou seja, é visto como um terreno comum para as aplicações (Franca et al., 2011).

O formato da mensagem SOAP ocasiona um maior consumo de banda devido ao tamanho da mensagem SOAP (ver Figura 2.5), um dos principais motivos para utilizar dispositivos como serviços Web RESTFul, pois muitos dispositivos na IoT possuem baixa capacidade de processamento e de banda de rede. Além disso, este formato pré-definido de mensagem força o cliente a tratar sempre este tipo de mensagem, enquanto utilizando REST é possível oferecer diferentes tipos de representações (JSON, XML, dentre outros)

(Franca et al., 2011).

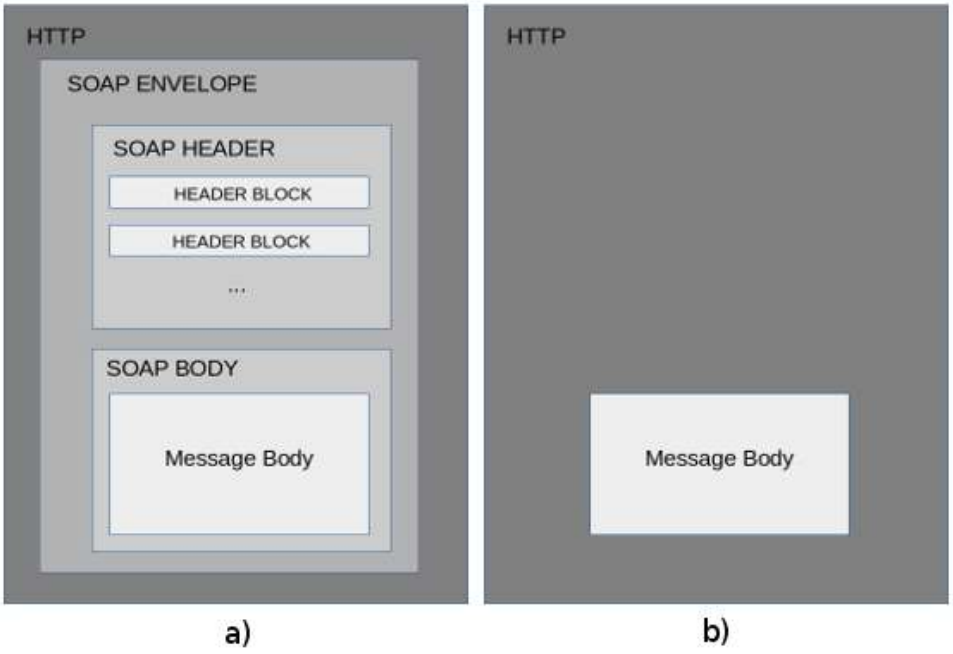

Figura 2.5: a)Mensagem SOAP sobre HTTP. b)Mensagem utilizando REST sobre HTTP. Figura baseada em (Pautasso, 2014).

Em relação ao termo "fraco acoplamento", o mesmo se refere a capacidade de fazer modificações no provedor de serviço sem afetar o cliente, os serviços Web RESTFul são menos acoplados, pois as operações sobre os recursos não mudam, já que tais operações são sobre os métodos do HTTP. Uma ressalva é que se as alterações forem feitas nos parâmetros passados nas mensagens, tanto o SOAP quanto o REST estarão no mesmo nível de acoplamento (Franca et al., 2011).

Por fim, na Figura 2.6 é apresentado uma análise comparativa entre duas aplicações que oferecem as mesmas funcionalidades, sendo que uma foi construída utilizando serviços Web SOAP e a outra utilizando serviços Web baseado nos princípios arquiteturais REST. A aplicação em questão é de conferência multimídia e pode ser usada para criar uma conferência, adicionar ou remover um participante de uma conferência que esteja em progresso, atualizar o tipo de mídia (mudar de audio para vídeo) e deletar uma conferência. Ambas as aplicações foram desenvolvidas em um mesmo ambiente e foram publicadas em um mesmo servidor de aplicação. Vários cenários foram executados e performances foram medidas. Todas as medidas são de tempo (milissegundos) e cada resultado é a média de 10 experimentos. Observe que tanto na mesma máquina quanto em um ambiente distribuído a performance do REST é sempre melhor, sendo superior de 3 até 5 vezes em relação ao SOAP quando em ambiente distribuído (Belqasmi et al., 2012).

Diante do cenário da IoT descrito até agora faz-se observar que muitos objetos quando isolados, nao tem como prover funcionalidade(s) que atendam ao(s) requisito(s) de um

	Baseado em SOAP			Baseado em REST		
Conferência Multimídia API	**Demora no ambiente distribuído**	**Demora na mesma máquina**	**Demora de carga na rede**	**Demora no ambiente distribuído**	**Demora na mesma máquina**	**Demora de carga na rede**
Criar conferência	848.4	381.7	767	171.4	102.7	273
Obter informação de conferência	818.6	335.3	546	172.3	98.6	177
Adicionar participante	1325.3	334.2	578	368.8	103.3	200
Remover participante	1322.3	357	588	382.9	107.2	195
Obter participantes	787.1	342.7	615	167.8	104.8	195
Obter informação de um participante	766.2	346.7	619	169.8	105	204
Finalizar conferência	1508.4	341.4	500	556.6	105.3	204

Figura 2.6: Resultados de performance entre duas aplicações com mesmas funcionalidades, mas uma baseada em SOAP e outra em REST. Figura adaptada de (Belqasmi et al., 2012).

usuario final (outra coisa ou um humano). Entao, para atender a tais requisitos, ha necessidade de compor estes objetos com outros para oferecer nova(s) funcionalidade(s) que atendam aos usuarios finais. Entretanto, a composicao das coisas pode levar a uma interacao de caracterısticas (seção 2.2) a qual pode provocar efeitos colaterais indesejaveis.

2.2 INTERAÇÃO DE CARACTERÍSTICAS

Em desenvolvimento de *software*, uma *feature* (característica) é um componente de adicional funcionalidade ao *software* (Calder et al., 2003), consistindo de um conjunto de requisitos logicamente relacionados e suas especificações, o qual se destina a fornecer um determinado efeito comportamental (NHLABATSI et al., 2008).

Poucas *features* atendem aos requisitos do usuário final quando estão isoladas. Para resolver este problema, características são combinadas para fornecer um novo componente de funcionalidade ao software. Entretanto, quando a composição de *features* leva a algum comportamento não esperado - interação de características, esta pode resultar em efeitos colaterais indesejáveis (NHLABATSI et al., 2008) ou em efeitos colaterais desejáveis (Weiss et al., 2005).

Efeito colateral indesejável é quando há interação de características e esta resulta em um estado inconsistente do sistema, um sistema instável ou dados imprecisos. Já efeito colateral desejável é quando a interação de características resulta num comportamento que ajudará de algum modo a funcionalidade em questão (Weiss et al., 2005), (NHLABATSI et al., 2008).

Existem diversas causas para interação de características, algumas delas são listadas a seguir de forma mais genérica possível (Weiss et al., 2007).

- *Goal Conflit* (Conflito de interesses): Cada característica tem seus próprios objetivos (designadas para fazer algum tipo de processamento, coletar algum dado, dar uma saída específica, dentre outros) a serem alcançados. Entretanto, quando as características são combinadas, os objetivos das duas características podem entrar em conflito e não se pode garantir que todos sejam alcançados.

- *Resource Contention* (Contenção de recurso): *Features* podem estar competindo umas com as outras devido ao acesso a recursos de capacidade limitada, tais como, memória, CPU, largura de banda, acesso a banco de dados, dentre outros. Desta forma, a correta execução de uma característica pode ser comprometida por outra característica que esteja utilizando além da sua cota de um recurso compartilhado.

- *Violation of Assumptions* (Violação de suposições): Desenvolvedores das funcionalidades de software precisam fazer algumas suposições de como outras características trabalham. Estes podem fazer suposições incorretas, por exemplo, devido a semântica ambígua (tal como uso do mesmo conceito de diferentes formas), ou devido a presença de diferentes versões da mesma funcionalidade de software. De forma similar, implementação de características podem ser baseadas em suposições incorretas sobre o contexto de uso. A caracterização de uma *Violation of Assumptions* se dá quando a mudança em uma *feature* quebra a suposição correta que a outra característica tinha sobre esta.

- *Policy Conflict* (Conflito de políticas): Políticas proveem os meios para especificação e modularização do comportamento de uma característica, na medida em que alinha suas capacidades e restrições com os requisitos de seus usuários. A *Policy Conflict* ocorre caso exista políticas (autenticação, ou de privacidade) especificadas em duas *features* que se referem as suas correspondentes operações, e que as políticas não são compatíveis.

Diversas taxonomias foram propostas para descrever os tipos de interação de características. Algumas destas propostas foram produzidas por Cameron e Velthuijsen (domínio das telecomunicações), Kolberg (domínio das *smarthomes*), e Shehata. Este último fez uma síntese das duas taxonomias citadas para uma taxonomia genérica a qual consiste de nove tipos de interação de características. Esta taxonomia tem a pretensão de ser aplicável na maioria dos domínios (NHLABATSI et al., 2008).

A seguir é apresentada e explicada os tipos de interação de características propostos por Shehata em uma versão reduzida da sua taxonomia (NHLABATSI et al., 2008). Para as explicações considere pré-condições como sendo as condições necessárias para que uma característica inicie sua execução e, como pós-condições as ações ou saídas esperadas após o término da execução da característica.

- *Dependence*: A execução de uma característica depende da execução correta da outra.

- *Non-determinism*: Ocorre quando características têm as mesmas pré-condições, mas pós-condições diferentes, ou seja, duas ou mais especificações de requisitos requerem que um domínio compartilhado tenha comportamentos distintos simultaneamente, quando o domínio pode somente ter um comportamento por vez. Domínio aqui significa uma propriedade do ambiente a qual uma especificação de uma característica utiliza para satisfazer seus requerimentos.

- *Negative impact*: Similar a *Non-determinism*, no sentido que as características sobrepõem pré-condições. Mas diferente no sentido que ambas as *features* são executadas e o resultado das suas pós-condições são inconsistentes. As pós-condições de uma característica diminui o efeito das pós-condições da(s) outra(s).

- *Invocation Order*: O comportamento da execução das características em uma determinada sequência é diferente do comportamento destas quando há mudança na sequência.

- *Bypass*: Uma característica F1 *bypasses* a execução de outra característica F2 se F1 muda o estado do ambiente compartilhado de tal forma que o novo estado não atende as pré-condições da outra característica F2.

- *Infinite Looping*: Ocorre quando duas *features* são reciprocamente ligadas em suas pós-condições e disparos de eventos. As características F1 e F2 são reciprocamente ligadas se as pós-condições de F1 cria um disparo de evento para F2 e vice e versa. Suponha que F1 é disparada e inicia sua execução criando um disparo de evento para F2. F2 inicia sua execução e cria um disparo para F1. Este processo então se repete indefinidamente, criando um loop infinito.

Autores em (NHLABATSI et al., 2008) não consideram *dependence* como uma causa de interação de características, pois este diz que *dependence* passa uma fraca noção de interação de características, já que esta implica que uma característica não funciona corretamente isoladamente e, segundo os autores, uma importante propriedade é que a *feature* deva satisfazer seus requisitos quando isolada, mas quando em composição pode acontecer a quebra desses requisitos.

Dentro deste cenário de interação de características, diversos métodos vêm sendo propostos para detecção e/ou resolução dessas interações. Duas abordagens são adotadas para este fim: *Online techniques* (técnicas online) e *Offline techniques* (técnicas offline). Técnicas online são métodos aplicados enquanto o sistema está sendo executado dentro de uma rede, ou seja, as *features* já foram desenvolvidas e estão em fase de produção sendo executadas. São uma combinação de mecanismos de detecção e resolução de interação de características. Já as técnicas offline (abordagens da engenharia de software e métodos formais) são aquelas aplicadas durante a fase desenvolvimento das características (Calder et al., 2003).

Dentre as técnicas offline, a abordagem da engenharia de software vêm propondo diversos caminhos (Thum et al., 2014; Siegmund et al., 2012a; Siegmund et al., 2012b) para auxiliar na remoção de interação de características antes de sua implantação (Calder et al., 2003). Na abordagem dos métodos formais (Almeida et al., 2011) uma variedade de técnicas também têm sido adotadas, tais como, lógicas clássicas e construtivas, autômatos de estado finito e infinito, autômatos estendidos, *petri-nets*, sistemas de transição e, linguagens como SDL, Promela, Z e LOTOS (Calder et al., 2003).

Dentre as técnicas online, *Feature Interaction Manager - FIM* é uma das técnicas que vem sendo adotada em *Home Network Systems ou Smart homes* (Wilson et al., 2005; Wilson et al., 2008; Nakamura et al., 2009). FIM (Calder et al., 2003) é um componente

do sistema introduzido dentro de uma rede com a capacidade de observar e controlar as chamadas a qualquer *feature*. Este trabalho concentra-se nesta técnica para prover detecção de efeitos colaterais indesejáveis.

2.3 HOME NETWORK SYSTEM

Diferentes sensores e aparelhos domésticos a exemplo de lâmpadas, ar-condicionadores, sistema de segurança e entretenimento, além de interagirem dentro do ambiente doméstico, podem ser conectados a Internet e controlados por um *smartphone* ou qualquer outro dispositivo da IoT. Desta forma pode-se não só controlar os dispositivos como também monitorar o ambiente doméstico (temperatura, consumo de energia, dentre outros). Uma casa com essas características é comumente chamada de *Smart Home* e é normalmente implementada como um *Home Network System* (Piyare, 2013).

Um *Home Network System* (HNS) é uma rede doméstica de coisas (aparelhos domésticos e sensores) com capacidade de conectividade de rede e, interface de controle e monitoramento. Nesse tipo de sistema os aparelhos e sensores podem juntos prover novas funcionalidades, as quais atendam as expectativas do usuário, como por exemplo, "Levar compras" (seção 3.1). Os aparelhos e sensores dessa rede podem ser disponibilizados como serviços Web RESTFul. O HNS tem um componente denominado de *Home Server*, este acessa os dispositivos através de APIs e disponibiliza as funcionalidades ao usuário final. O HS também pode atuar de outra forma, detectando e resolvendo efeitos colaterais indesejáveis (seção 2.2), para isto é incorporado ao HS um novo componente de software, o *Feature Interaction Manager* (FIM). O HS ainda pode servir de mediador para redes externas, ou seja, este pode disponibilizar cada serviço de forma única na Internet, para isto, basta ter um endereço IP público estático e oferecer seus serviços através de uma API, por exemplo, uma API que segue os princípios REST. Desta forma, cada dispositivo do HNS pode ser identificado unicamente na chamada Internet das Coisas (Nakamura et al., 2009), (Ikegami et al., 2013). Este trabalho replica um cenário de HNS, conforme descrito na seção 3.1.

Para que o FIM (seção 2.2) possa prover a detecção de efeitos colaterais indesejáveis será proposto (seção 3) utilizar algum procedimento de classificação, tema abordado na seção 2.4 seguinte.

2.4 CLASSIFICAÇÃO - APRENDIZADO SUPERVISIONADO

Em seu sentido mais geral, o termo classificação poderia cobrir qualquer contexto no qual alguma decisão ou predição é realizada baseada nas informações presentes. Classificação então pode ser caracaterizada como um método formal que realiza julgamentos repetidos a cada nova situação. No caso deste trabalho, o termo classificação será restrigindo a situação de se construir um procedimento classificatório com base em um conjunto de dados no qual as classes são conhecidas, comumente chamado de aprendizado supervisionado (Michie et al., 1994). O objetivo final de se construir esse procedimento classificatório é que este possa generalizar para conjuntos de dados que não participaram da sua construção, ou seja, o classificador tem que ser capaz de classificar corretamente

novos dados (Kotsiantis, 2007). O conjunto de dados aqui citado é composto de objetos (instâncias), sendo cada instância representada por um conjunto de atributos e, cada uma é pertencente a uma classe específica (ver Tabela 2.1).

Objeto	**atributo 1**	**atributo 2**	**..**	**atributo n**	**classe**
1	10	3		4	A
2	11	11		9	A
3	14	26		6	B
..				..	

Tabela 2.1: Conjunto de objetos: seus atributos e classe pertencente. Adaptado de (Kotsiantis, 2007)

.

Muitos problemas nas areas da ciência, indústria, medicina, dentre outros, podem ser tratados como problemas de classificação. A exemplo de diagnósticos médicos (classificar tumores como benigno ou maligno), classificação de transações de cartão de crédito como legítima ou fraudulenta, controle de qualidade, reconhecimento de fala (Zhang, 2000).

De modo geral, a metodologia adotada para construção de um classificador é como segue e pode ser visualizado na Figura 2.7.

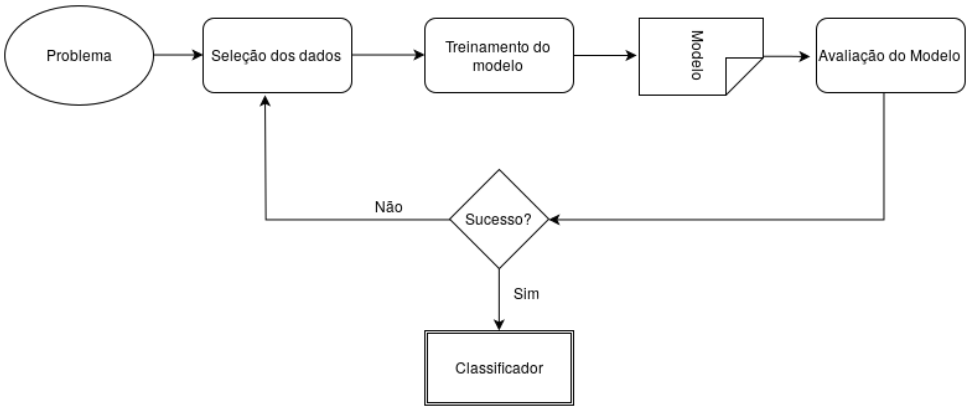

Figura 2.7: Processo de construção de um classificador em aprendizado supervisionado. Baseado em (Kotsiantis, 2007) e proft[6].

1. Problema: primeiramente se deve saber qual problema quer resolver.

2. Seleção dos dados: coleta do conjunto de dados (*dataset*). Se existir uma pessoa com alto conhecimento sobre os atributos, este pode sugestionar quais atributos são mais significativos para o problema. Caso contrário o mais simples método é mensurar todos os atributos na esperança que as características corretas possam ser isoladas (Kotsiantis, 2007).

[6] ⟨http://en.proft.me/2015/12/24/types-machine-learning-algorithms⟩

3. Treinamento do modelo: treina-se o dataset da etapa anterior sobre um algoritmo de classificação (árvores de decisão, *Fisher's linear discriminants*, redes neurais - *Multi Layer Perceptron (MLP)* (Michie et al., 1994)). Então obtêm-se um modelo, que é o algoritmo de classificação treinado com o dataset.

4. Avaliação do Modelo: é baseada em métodos avaliativos (ver seção 2.4.2), os quais têm o propósito de verificar a generalização do modelo, ou seja, verificar o quão bom é o modelo do classificador quando submetido a instâncias desconhecidas (que não fazem parte do modelo). Uma das medidas utilizadas por tais métodos é a taxa de acurácia (percentagem das predições feitas corretamente dividida pelo número total de predições realizadas) (Kotsiantis, 2007).

5. Classificador: se o modelo foi validado na etapa anterior, este generaliza o suficiente para o problema proposto, então este modelo pode ser utilizado como o classificador final (pode ser implantado em um sistema), caso contrário reinicia-se o processo a partir da "seleção de dados" reavaliando as decisões de cada etapa.

Dentre os algoritmos citados na etapa de treinamento da Figura 2.7, as redes neurais são modelos que atendem bem tanto problemas de classificação que não são linearmente separáveis (problemas que se adequam mais a realidade), quanto problemas que são linearmente seperáveis, entretanto, utilizar-se das redes neurais para problemas linearmente separáveis é totalmente desaconselhável, já que pode ser muito mais custoso computacionalmente.(Zhang, 2000)(Elizondo, 2006) A Figura 2.8 ilustra problema linearmente separável e problema não separável linearmente. Apesar de alguns problemas serem não linearmente separáveis, estes podem ser separados de forma linear sem perdas substânciais (Figura 3.8).

Algumas comparações (Firdausi et al., 2010; Arora e Suman, 2012; Karthikeyan e Thangaraju, 2013) vêm sendo realizadas entre J48 (árvore de decisão, implementação WEKA (Hall et al., 2009) do algoritmo C4.5 (Quinlan, 1993)) e MLP (rede neural). Nestas, J48 obteve resultados melhores ou próximos para problemas de classificação binária (entre duas classes) e mostrou-se ser muito menos custoso computacionalmente. Sendo assim, o uso do algoritmo J48 parece ser uma boa escolha e, este será utilizado como base do *DECORATE*, ver seção 2.4.1, o qual será o método de classificação utilizado neste trabalho.

2.4.1 DECORATE - um método ensemble

Um método *ensemble* combina as decisões de diversos classificadores para chegar numa decisão final. A diversidade das hipotéses dos classificadores do *ensemble* é tida como um importante fator para obter uma boa generalização. O DECORATE *(Diverse Ensemble Creation by Oppositional Relabeling of Artificial Training Examples)* constrói diversas hipóteses utilizando datasets de treino adicionais gerados artificialmente. Resultados experimentais demonstraram que o DECORATE tendo como algoritmo base o J48 coseguiu obter taxas de acurácia satisfatórias, mais altas que outros métodos *ensemble* e, melhores do que se o J48 fosse utilizado de forma isolada. O DECORATE também se saiu muito

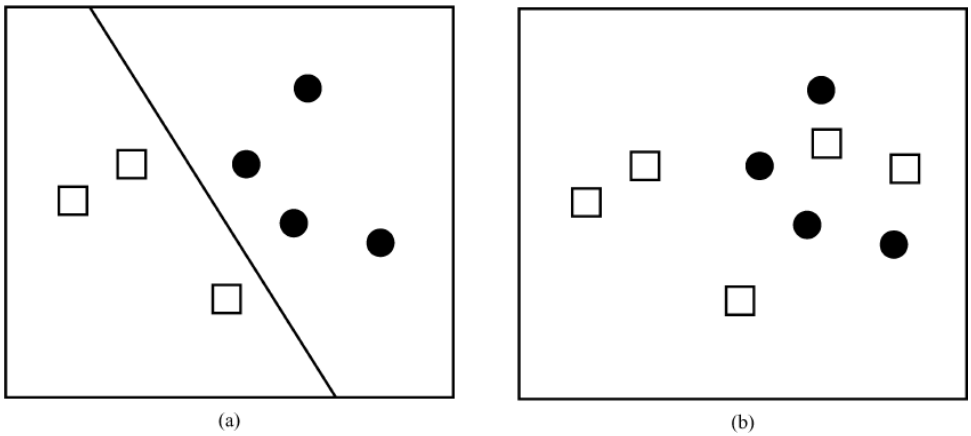

Figura 2.8: (a) Problema linearmente separável. (b) Problema não separável linearmente. (Elizondo, 2006)

bem em *datasets* pequenos, nos quais obteve um bom grau de generalização (Melville e Mooney, 2003; Melville e Mooney, 2004).

No DECORATE, o *ensemble* é gerado de forma iterativa, ou seja, a cada iteração é gerado um classificador e este é adicionado ao *ensemble*. Na primeira iteração o *ensemble* contém o classificador treinado no *dataset* original. A partir da segunda iteração os classificadores são gerados com o *dataset* original mais dados artificiais. O número de exemplos artificiais a serem gerados é especificado como um fator em cima do tamanho do *dataset* de treino. Caso a adição do novo classificador ao *ensemble* aumente sua taxa de erro, este é ignorado, senão é adicionado ao *ensemble*. Este procedimento é repetido até que se alcance um número de *committee* (membro do comitê julgador ou classificador) desejado ou exceda o limite de iterações (Melville e Mooney, 2003).

Para classificar um exemplo não rotulado, x, utiliza-se a seguinte fórmula 2.1, onde C_i é um classificador pertencente ao *ensemble* C^*, $|C^*|$ representa a quantidade de classificadores no *ensemble* e $P_{C_{i,y}}(x)$ é a probabilidade de x pertencer a classe y de acordo com o classificador C_i.

$$P_y(x) = \frac{\sum_{C_i \in C^*} P_{C_{i,y}}(x)}{|C^*|} \tag{2.1}$$

2.4.2 Métodos avaliativos

Métodos avaliativos têm o propósito de verificar a generalização do modelo, ou seja, verificar quão bom é o modelo do classificador quando submetido a instâncias desconhecidas (que não fazem parte do modelo). Para isto, utiliza-se de medidas estatísticas para auxiliar na tarefa (Witten e Frank, 2005). Como por exemplo, acurácia, *precision*, *recall*, dentre outras. As medidas estatísticas citadas são baseadas em quatro componentes: verdadeiro positivo (*True Positive - TP*), verdadeiro negativo (*True Negative - TN*), falso positivo (*False Positive- FP*) e falso negativo (*False Negative- FN*) (Davis e Goadrich, 2006).

- Verdadeiro positivo (TP): instância que pertence a classe positiva e que foi corretamente classificada como positiva.
- Verdadeiro negativo (TN): instância que pertence a classe negativa e que foi classificada corretamente como negativa.
- Falso positivo (FP): instância que pertence a classe negativa e que foi classificada incorretamente como positiva.
- Falso negativo (FN): instância que pertence a classe positiva e que foi classificada incorretamente como negativa.
- *Accuracy*: taxa correta de classificação em relação a todos os exemplos, a qual pode ser visualizada e obtida através da fórmula 2.2 (Metz, 1978).
- *Precision*: mede a taxa de exemplos classificados como positivos que realmente são positivos. Ou seja, dentre todos os exemplos classificados como positivos, quais realmente são positivos? Esta relação pode ser visualizada e obtida através da fórmula 2.3 (Davis e Goadrich, 2006).
- *Recall* ou *True Positive Rate - TPR*: mede a taxa de exemplos positivos que foram corretamente classificados. Ou seja, dentre os exemplos classificados como positivos (que realmente são positivos) e os classificados como negativos (mas que são positivos), qual a taxa de exemplos positivos classificados corretamente? Esta relação pode ser visualizada e obtida através da fórmula 2.4 (Davis e Goadrich, 2006).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

Outra medida utilizada é o desvio padrão, utilizada para verificar a dispersão de um conjunto de dados medido, por exemplo, o desvio padrão de cada conjunto das medidas avaliativas citadas anteriormente (*Precision*, *Recall*, *Accuracy*) quando aplicado o *stratified-k-fold-cross-validation*. Um desvio padrão baixo indica que os elementos do conjunto têm valores próximos ao esperado, enquanto que valores altos indicam que os elementos estão dispersos dentro de uma faixa grande de valores (Kerr et al., 2002), (Bland e Altman, 1996).

O desvio padrão pode ser medido de acordo com a fórmula 2.5, que é a $\sqrt{variância}$. Onde x_i é um elemento do conjunto, $\bar{x}$ é a média aritmética dos i elementos, e n é a quantidade de elementos do conjunto. Usualmente o desvio padrão é também calculado para somente um subconjunto das medidas, desta forma o valor produzido é uma estimativa do desvio padrão para a população dos dados em questão. Para este caso em específico,

a fórmula 2.5 sofre uma pequena alteração (fórmula 2.6, ao invés de n, tem-se n-*1*), a fim de corrigir os possíveis erros gerados por tal estimativa (Kerr et al., 2002).

$$Desvio\ padrão\ (S) = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n}} \tag{2.5}$$

$$Desvio\ padrão\ (S) = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}} \tag{2.6}$$

Uma exemplificação da mensuração do desvio padrão pode ser realizada com os dados da Tabela 2.2 e aplicação da fórmula 2.5. Os elementos x_i (*recall*) da população foram gerados utilizando *stratified-10-fold-cross-validation* em cima de um dos *datasets* gerados através de tetes iniciais realizados neste trabalho com o classificador MLP. A soma dos $x_i - \bar{x}$, na segunda coluna da tabela, gerou um valor diferente de zero devido a aproximação realizada para duas casas decimais, o correto seria 0 (zero). A aplicação da fórmula neste cenário pode ser visualizada em 2.7.

$$Desvio\ padrão\ (S) = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n}} = \sqrt{\frac{4173.42}{10}} = 24.43\% \tag{2.7}$$

$x_i(\%)$	$x_i - \bar{x}(\%)$	$(x_i - \bar{x})^2(\%)^2$
100.00	19.17	367.49
66.67	14.16	200.50
100.00	19.17	367.49
100.00	19.17	367.49
75.00	5.83	33.99
100.00	19.17	367.49
66.67	14.16	200.50
50.00	30.83	950.49
50.00	30.83	950.49
100.00	19.17	367.49
$\sum$ =808.34	0.04	4173.42

Tabela 2.2: Medidas *recall* mensuradas a partir da aplicação da técnica *stratified-10-fold-cross-validation* com o classificador MLP sobre um dos *datasets* gerados nos testes iniciais deste trabalho. Tabela baseada em (Kerr et al., 2002).

Um método bastante simples para avaliar um modelo é dividir o dateset em dois grandes conjuntos (um para a construção do modelo - treino, e outro para testar o modelo - teste). Então se faz pelo menos uma medição estatística (por exemplo, acurácia) de cada instância do *dataset* de teste. Assim, pode-se utilizar o mesmo procedimento com outro modelo e realizar comparações. Mas esta técnica só é aceitável quando se tem um conjunto de dados grande o suficiente para realizar este tipo de repartição (Kotsiantis, 2007). Quando não se tem um dataset de tamanho avantajado, recorre-se a outros tipos

de métodos para verificar a generalização do modelo, tais como, *randomized-holdout* e *stratified-k-fold-cross-validation* (Witten e Frank, 2005).

No método *randomized-holdout* uma parte do dataset é escolhida randomicamente para treino e o restante é utilizado para teste, normalmente considerando a proporção de dois terço (treino) e o restante para teste. Este processo então é repetido diversas vezes e é computada a média das medidas estátisticas de cada repetição. Em cada repetição deve-se randomizar o dataset de maneira distinta (Witten e Frank, 2005).

No método *stratified-k-fold-cross-validation* é escolhido um número fixo k de *folds* (compartimentos ou pastas). O *dataset* é dividido randomicamente em k pastas iguais (ou aproximadamente), nas quais cada classe é representada aproximadamente na mesma proporção (estratificação - preserva a representatividade dos dados). Um compartimento é então utilizado para teste e, os k-1 restante são utilizado para treino, como pode-se observar na Figura 2.9. Daí computa-se uma medida estatística. Então o procedimento é executado k vezes. Por fim têm-se uma media das medidas estatísticas das k repetições. Assim como no *randomized-holdout* é recomendável que se repita o *stratified-k-fold-cross-validation* diversas vezes, com randomização diferente para cada repetição. Este método é um dos mais utilizados quando se tem uma base de dados pequena (Witten e Frank, 2005).

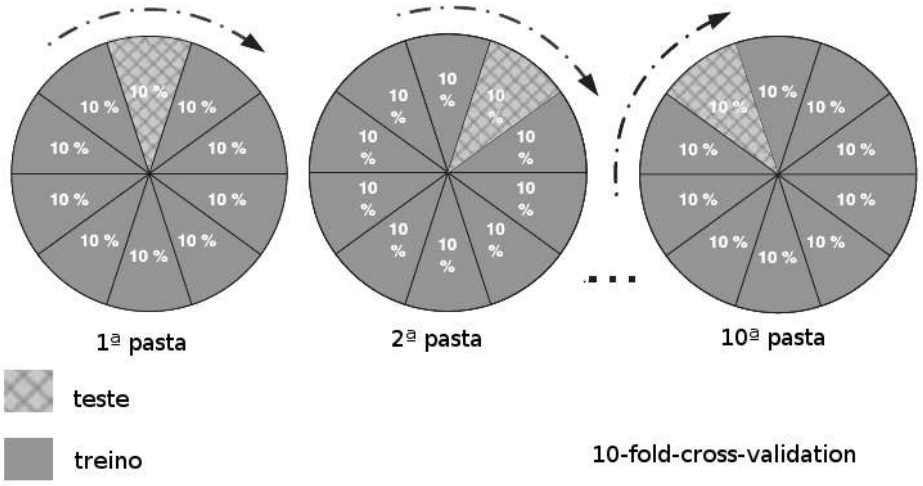

Figura 2.9: k-fold-cross-validation. Adaptado de (Olson e Delen, 2008).

Normalmente para o valor de k, no método descrito anteriormente, utiliza-se k=10. Este número de k tornou-se padrão após uma quantidade enorme de testes em datasets utilizando diferentes técnicas de classificação, os quais demonstraram k=10 como sendo aproximadamente a melhor escolha. A mesma quantidade (dez) têm sido escolhida como padrão para o número de repetições do *stratified-k-fold-cross-validation* (Witten e Frank, 2005).

No próximo capítulo (3) será apresentada a proposta deste trabalho para detectar efeitos colaterais indesejáveis entre dispositivos no cenário descrito na seção 3.1.

Capítulo

3

Estudo de caso no Home Network System.

ABORDAGEM PROPOSTA

Diante da visão da IoT propõe-se um estudo de caso no HNS (seção 2.3), com o intuito de verificar se é possível detectar efeitos colaterais indesejáveis. Para isto é idealizado um cenário, "Levar Compras" (seção 3.1), dentro de um HNS (Figura 3.1). Neste, cada dispositivo da casa (veículo terrestre, elevador, garra mecânica) é disponibilizado como um serviço Web RESTFul. O HS do HNS é um servidor Web RESTFul com um FIM (seção2.2) integrado, no qual será implantado um classificador para detecção de efeitos colaterais indesejáveis aplicavél ao cenário "Levar Compras", caso possa se construir um modelo com um grau de generalização aceitável (seção 3.3). A discussão do motivo de se utilizar dispositivos como serviços Web e, RESTFul em vez de SOAP foi realizada na seção 2.1.4.

3.1 CENÁRIO LEVAR COMPRAS

O cenário "Levar compras" refere-se a funcionalidade (*feature* ou característica) oferecida pelo HS. Esta característica é provida através da composição dos serviços do elevador, veículo e garra. O cenário em questão pode ser visualizado na Figura 3.2 e é descrito a seguir.

1. Um usuário da casa (morador ou visitante), organiza as compras numa cesta (que tem formato expansível) e a coloca sobre o veículo terrestre.

2. O usuário então aciona o serviço "Levar compras" (disponibilizado pelo HS) através do seu *smartphone*. O serviço então começa sua execução.

3. O FIM solicita informações de restrições do elevador e veículo terrestre, assim como quais objetos compõem a cesta. Este então verifica se a cesta colocada sobre o carro quebrou alguma restrição do mesmo e se a massa total dos objetos excedem a carga máxima do elevador.

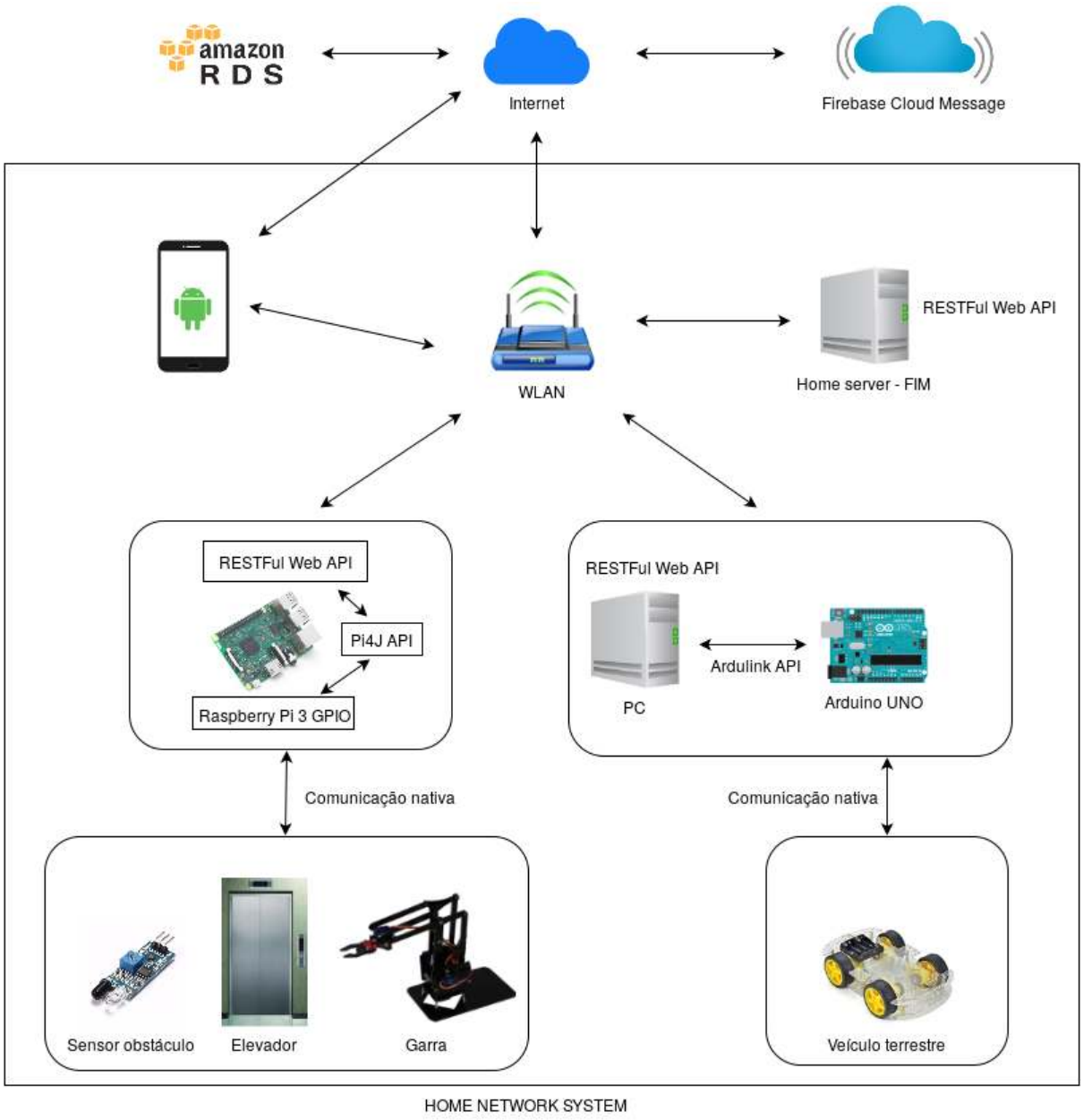

Figura 3.1: Modelo do HNS utilizado neste trabalho.

4. Se nenhuma destas restrições foram quebradas o FIM solicita ao veículo terrestre que leve a as compras até as proximidades do elevador.

5. Assim que o veículo chega no seu destino, este envia uma mensagem ao FIM informando de sua chegada.

6. O FIM então requisita o elevador para levar as compras.

7. O elevador agora solicita a garra mecânica que pegue o cesto e coloque-o dentro do elevador.

8. Quando a garra completa a execução do seu serviço, esta avisa ao elevador.

9. Caso não haja nenhuma restrição, como por exemplo, alguma coisa que possa impedir a porta do elevador de fechar, então o elevador inicia o transporte das compras ao seu destino final e manda uma notificação ao HS.

10. HS Captura token referente ao *smartphone* com Android[1] do usuário da casa.

11. Por fim, o HS solicita ao serviço de notificação *Firebase Cloud Message*[2] que envie uma notificação para o usuário da casa, que a recebe no seu *smartphone*. Assim completa-se a execução do serviço "Levar compras".

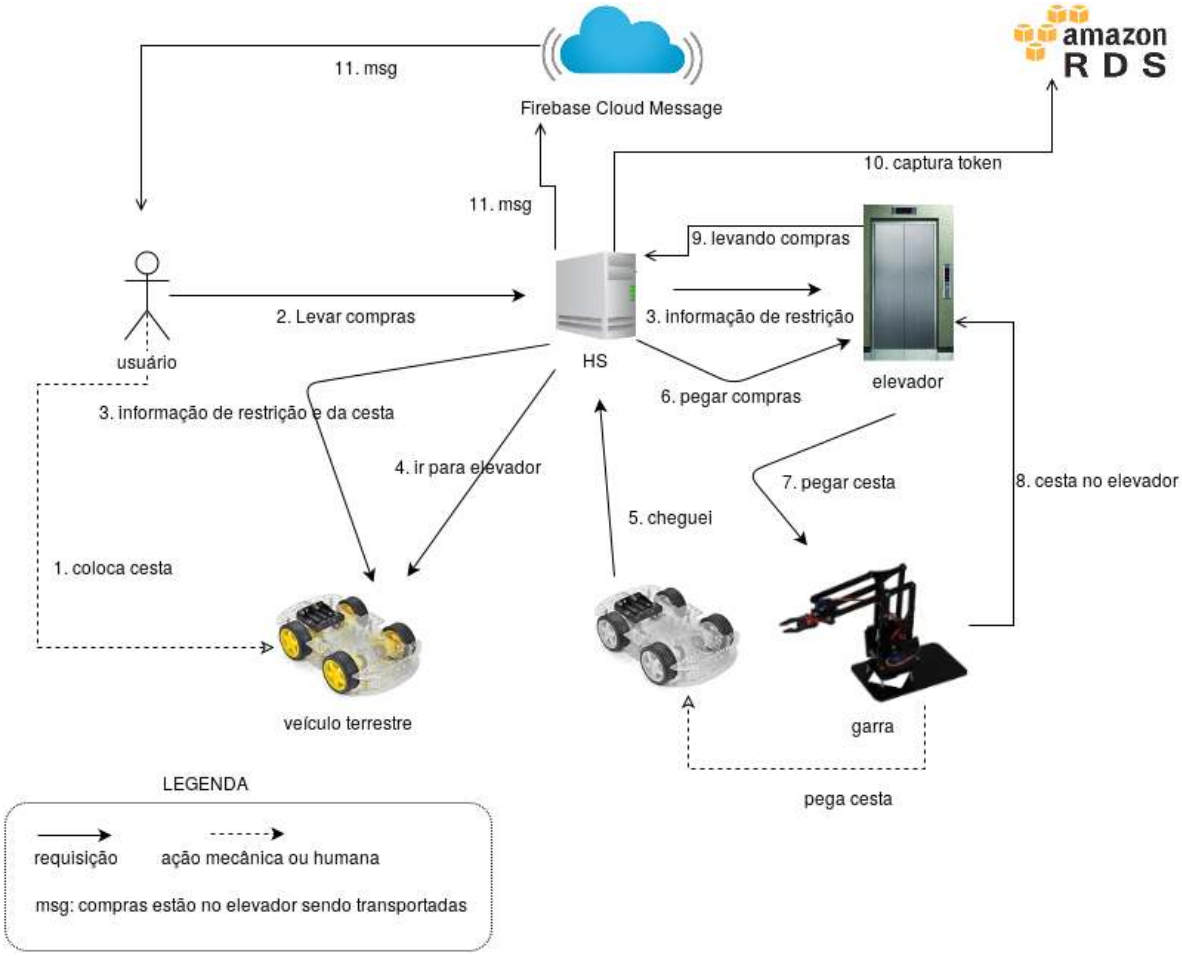

Figura 3.2: Cenário Levar compras.

No cenário descrito da Figura 3.2 o elevador não contém motores nem algum outro tipo de sistema elétrico ou mecânico, este é representado por uma caixa de papelão e

[1] ⟨https://www.android.com/intl/pt-BR_br/⟩
[2] ⟨https://firebase.google.com/docs/cloud-messaging/?hl=pt-br⟩

um sensor de obstáculo acoplado. O veículo terrestre é representado por um programa carregado no Arduino UNO[3], o qual escuta através de sua porta serial mensagens provindas do PC (Figura 3.2) e as processa simulando as funcionalidades "Ir para elevador" e "cheguei". Pela mesma porta serial o veículo terrestre encaminha suas mensagens ao PC. Os dispositivos veículo, elevador e garra podem ser visualizados na Figura 3.3.

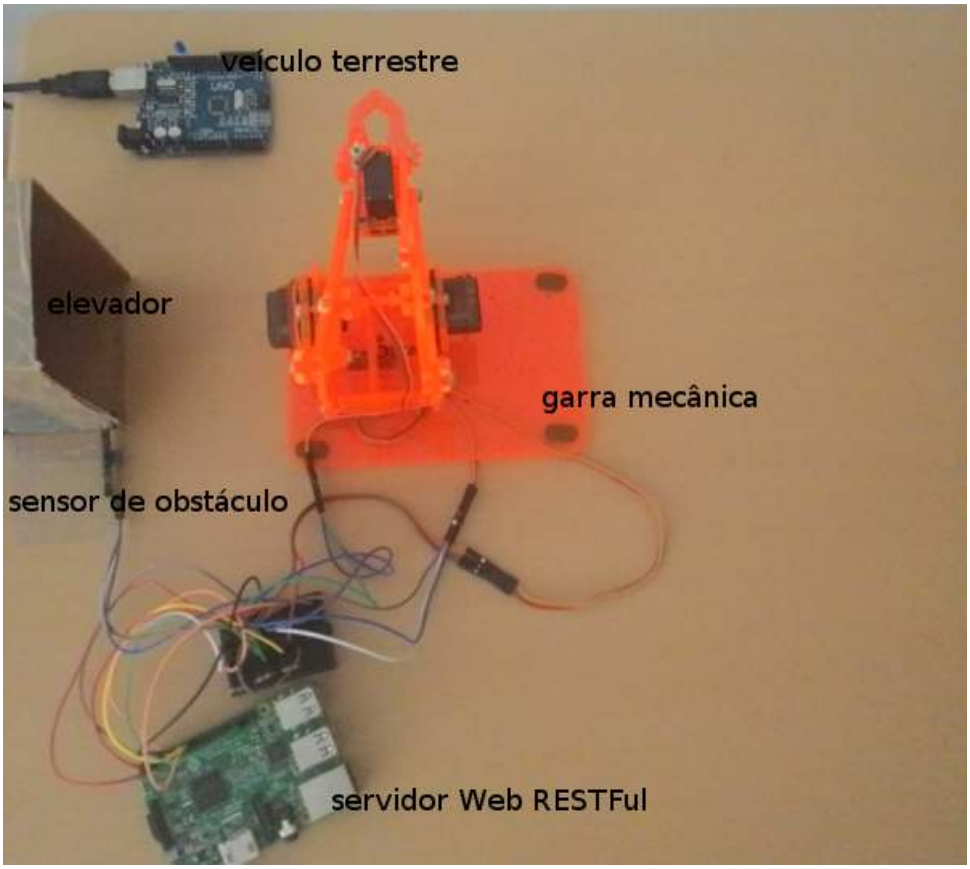

Figura 3.3: Dispositivos reais do cenário "Levar compras".

3.1.1 Execução do cenário Levar compras

De acordo com a Figura 3.2 o primeiro passo para inicializar a execução do cenário é colocar a cesta no veículo terrestre. Apesar do veículo ser simulado através de um software embarcado no Arduino, este necessita das informações dos objetos que se deseja transportar. Sendo assim, as informações da cesta são passadas quando o usuário executa o passo 2 (Levar compras). O usuário digita o "id da cesta" (no banco de dados) que se deseja transportar e clica em um botão para inicializar o serviço. Todas as cestas são compostas com miniaturas de objetos que normalmente são encontrados em supermercados, exemplos de cestas podem ser visualizados na Figura 3.4. Todos os objetos e cestas

[3] ⟨https://www.arduino.cc/en/Main/ArduinoBoardUno⟩

estão cadastradas no banco de dados na nuvem (amazon RDS[4]), o qual foi apresentado na Figura 3.1. A lista de objetos que podem fazer parte de uma cesta é demonstrada na Figura 3.5, observe que para cada objeto é cadastrado suas características de massa, largura, comprimento, altura e diâmetro.

Figura 3.4: Exemplo de cestas do cenário "Levar compras".

Os próximos passos (3, 4, 5, e 6) do cenário podem ser executados sem mais problemas, pois já têm toda a informação necessária e não necessita de nenhuma intervenção humana. Isto já não acontece com o passo 7 (pegar cesta), pois este necessita de intervenção humana para a garra poder pegar a cesta e continuar a realizar o seu serviço, é preciso posicionar a cesta no lugar correto para a garra prender a cesta, como pode ser visto na Figura 3.6. Então, após a garra executar seu serviço esta manda uma mensagem para o elevador e, caso não tenha nada obstruindo a porta do elevador este inicia o transporte da cesta até o local desejado e manda uma mensagem para o FIM. Daí em diante a execução do cenário continua da mesma forma com já explanado.

3.1.2 Observação do problema

Para a execução deste cenário foram confeccionadas e cadastradas no banco 59 cestas distintas. Antes de registrar as cestas no banco sempre era observado se a cesta cabia

[4] ⟨https://aws.amazon.com/pt/rds/postgresql/⟩

largura	altura	comprimento	massa	diâmetro	nome
0	0.031	0	0.003	0.009	Refrigerante nº 4 - Soda
0.01	0.014	0.03	0.001	0	Caixa Pasta de dente - vermelha.
0.004	0.02	0.014	0.001	0	Batata
0.004	0.02	0.014	0.001	0	Café
0	0.015	0	0.003	0.008	Café Classic
0.005	0.02	0.011	0.001	0	Cerveja em lata III
0	0.028	0	0.001	0.007	Champanhe - Azul
0.004	0.02	0.014	0.001	0	Farinha de Trigo
0	0.03	0	0.001	0.008	Garrafa de Rum
0.012	0.016	0.004	0.004	0	Goiabada
0	0.03	0	0.001	0.008	Garrafa de Vodka Special
0.027	0.014	0.018	0.001	0	Pote de sorvete
0	0.032	0	0.002	0.007	Vinho branco
0.012	0.03	0.006	0.002	0	Limpa vidros
0.021	0.015	0.007	0.002	0	Sabonete - II
0.015	0.024	0.008	0.001	0	Sabão em pó - II
0.019	0.03	0.039	0.018	0	Baú simples
0.046	0.061	0.017	0.006	0	Prateleira de acrílico pequena
0.016	0.025	0.004	0.001	0	Farinha de trigo
0	0.03	0	0.003	0.022	Barril em madeira
0.013	0.017	0.0063	0.002	0	Fermento em pó
0.077	0.0197	0.038	0.001	0	Colchão p/ berço azul
0.01	0.0432	0	0.001	0	Jogo de esfregões
0.0181	0.0258	0.0017	0.001	0	Pão de Queijo
0.022	0.0412	0.0135	0.001	0	Rastelo

Figura 3.5: Lista dos objetos que podem ser utilizados para compor uma cesta.

dentro do elevador quando colocadas de forma manual, e se também não ultrapassava os limites da porta, o limite de detecção do sensor de obstáculos. Então a cesta só era registrada caso coubesse no elevador e não estivesse em área que pudesse obstruir a porta.

O cenário foi executado 59 vezes, uma para cada cesta, e pode-se observar que em algumas vezes o comportamento da garra provocava um efeito colateral indesejável (seção 2.2) na funcionalidade “Levar compras”, pois a garra durante a execução do seu serviço (pegar cesta, mover de uma lado a outro , largar cesta) acabava por vezes não conseguindo que a cesta ficasse disposta dentro do elevador de uma maneira em que este pudesse continuar com o serviço de “Levar compras”, mesmo que para todos esses casos se esperasse sempre como resultado o elevador levar as compras sem problemas, já que se era sabido que as cestas cabiam no elevador e, nenhuma delas feria as restrições de massa do carro nem do elevador. Tanto o veículo quanto o elevador sempre funcionaram de maneira esperada nestas execuções, pois o veículo sempre levava a carga para o local destinado avisando quando chegasse, e o elevador sempre acionava o serviço da garra, levava a carga e avisava ao FIM quando não tinha obstáculo na porta, enquanto deixava de levar quando tinha, funcionando de maneira correta quanto as especificações de sua funcionalidade de locomoção, mas indo contra o comportamento esperado da funcionalidade pai “Levar carga”.

Concluindo as observações feitas até agora, o comportamento da garra junto com os tipos de objetos presentes na cesta faziam com que algumas vezes ocorressem locomoção destes durante o transporte da garra, e também no momento em que os objetos eram postos no compartimento do elevador. Nesse momento, os objetos as vezes geravam outras dimensões de cesta e/ou deslocamento da mesma, devido a natureza dos objetos e da forma como os objetos eram soltos pela garra. Sendo assim, para tentar detectar esses efeitos colaterais indesejáveis através do DECORATE, utilizou-se dos valores das grandezas físicas dos objetos (altura, massa, largura, comprimento, diâmetro) para gerar características que pudessem ser interessantes para a construção de modelos de classi-

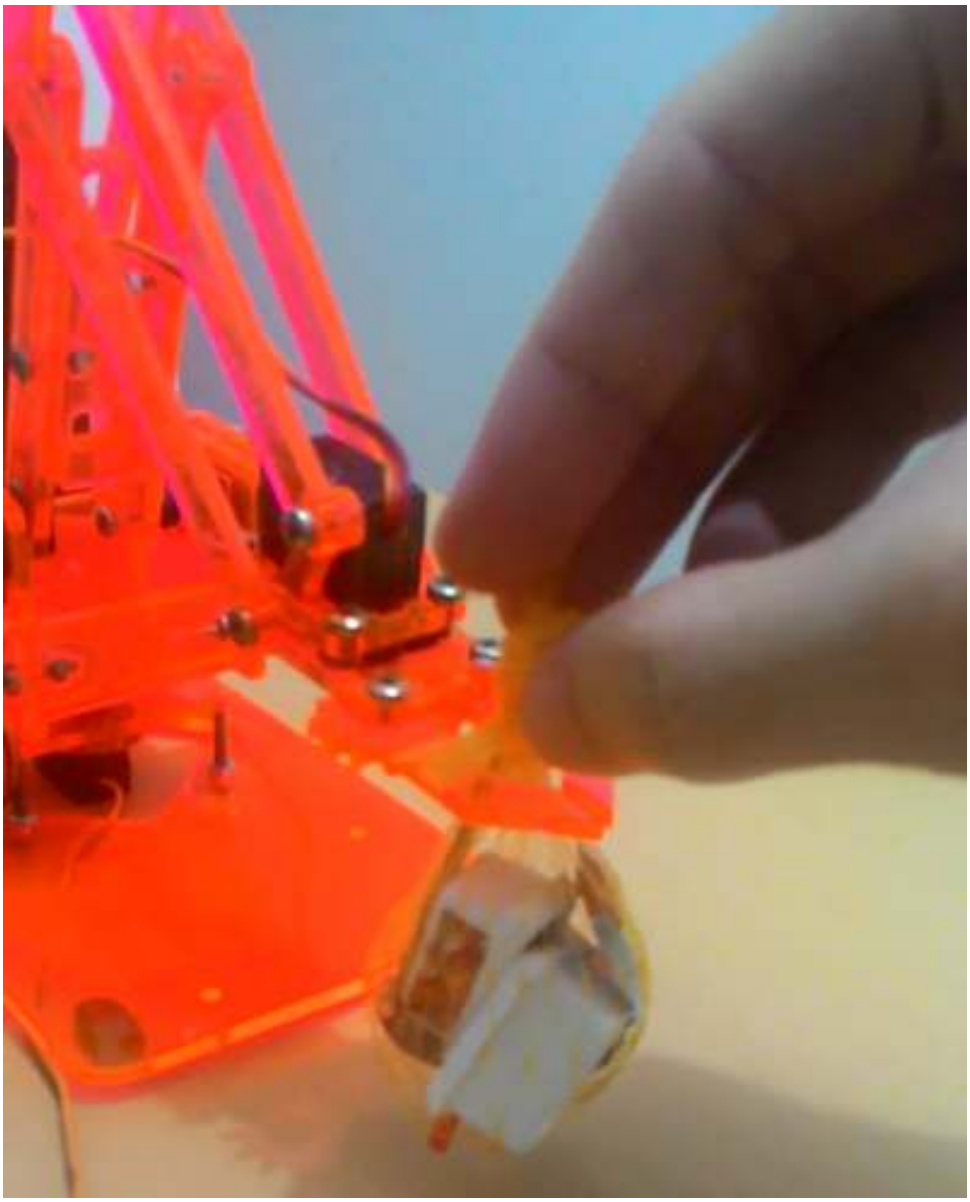

Figura 3.6: Cesta posicionada no local adequado por um humano, para que a garra possa prender a cesta adequadamente.

ficadores satisfatórios (seção 3.3). Vale salientar que esta informação da cesta é uma resposta do veículo terrestre referente a requisição "3" da Figura 3.2, ou seja, é uma das informações que os dispositivos e/ou serviços trocam entre si durante a excecução de algum serviço.

Como todas as cestas estavam previamente cadastradas no banco de dados, a estas foram atribuídos valores de "é efeito colateral" ou "não é efeito colateral", a medida que se observava cada execução do cenário. Desta forma, será considerado para detecção de efeitos colaterais indesejáveis as informações das cestas.

3.2 SELEÇÃO DOS DADOS

Os dados coletados na etapa de execução do cenário "Levar carga", 59 cestas, ainda estão muito brutos, observe que uma cesta deve conter pelo menos um objeto, como também pode conter muitos. Supomos que a cesta tenha 10 objetos, então o total de características da cesta seria $10.5 = 50$, pois cada objeto possui cinco características. Utilizar as características dos objetos desta forma, além de provavelmente não predizer nada em um modelo de classificação, terá um custo computacional muito elevado. Para diminuir a quantidade de atributos e melhor qualificá-los gerou-se características a partir de todos os atributos dos objetos da cesta: média das alturas, dos comprimentos, das larguras, dos diâmetros e das massas, assim como o total de cada uma dessas medidas, e o total de itens. De ínicio então têm-se a seleção de 11 parâmetros.

Com a seleção prévia desses atributos, constriu-se um arquivo "arff", que é o tipo de arquivo padrão utilizado como *dataset* pelo Weka (Hall et al., 2009). Este *dataset* é composto de 59 instâncias (referente as 59 cestas), cada uma com 11 atributos mais um que identifica a qual classe pertence. Então têm-se 11 atributos númericos e um nominal (a classe a qual a instância pertence).

O próximo passo utilizado nesse processo de seleção de atributos foi o de visualizar estes novos parâmetros par a par, e verificar quais pares parecem melhor separar as classes "é efeito colateral" ou "não é efeito colateral". A Figura 3.7 passa a ideia de tal visualização. Dentre essas plotagens foram selecionadas três da Figura 3.8, as quais aparentam melhor separar as classes, são respectivamente os pares ordenados (somaLargura, somaMassa) - "sl x sm", (somaDiâmetro, somaLargura) - "sd x sl" e, (médiaAltura, somaLargura) - "ma x sl". Então, nesta fase de seleção dos atributos, estes foram os pares selecionados para construção e validação do modelo (seção 3.3 seguinte). Observe que o espaço de *features* foi reduzido a duas características.

3.3 VALIDAÇÃO DO MODELO DE CLASSIFICADOR

Diante dos pares de atributos selecionados na etapa anterior (seção 3.2), utilizou-se o método *stratified-10-fold-cross-validation* (repetido dez vezes) juntamente com o *ensemble DECORATE* a fim de se construir e verificar se é possível generalizar pelo menos um modelo.

Em cada validação constrói-se um modelo com diferentes frações do dataset, variando de 16 instâncias a 59 instâncias. Os parâmetros do DECORATE escolhidos neste trabalho foram semelhantes a um dos testes avaliativos realizados em (Melville e Mooney, 2004), nos quais comparam *DECORATE* com outros métodos classificatórios, em diferentes tamanhos de dataset e diferentes parâmetros do DECORATE. Para as avaliações neste trabalho o número de iterações é setado em 50, o número máximo de classificadores participantes no *ensemble* é setado em 14, o classificador base utilizado é o J48 e, a quantidade de exemplos artificiais gerados em cada iteração é setado para 100% (valores setados de 50% a 100% não faz variar muito o resultado (Melville e Mooney, 2004)) do tamanho do dataset de treino. Os resultados das avaliações podem ser vistos na Figura 3.9. Nesta percebe-se um domínio quase absoluto da curva pontuada de aprendizado verde

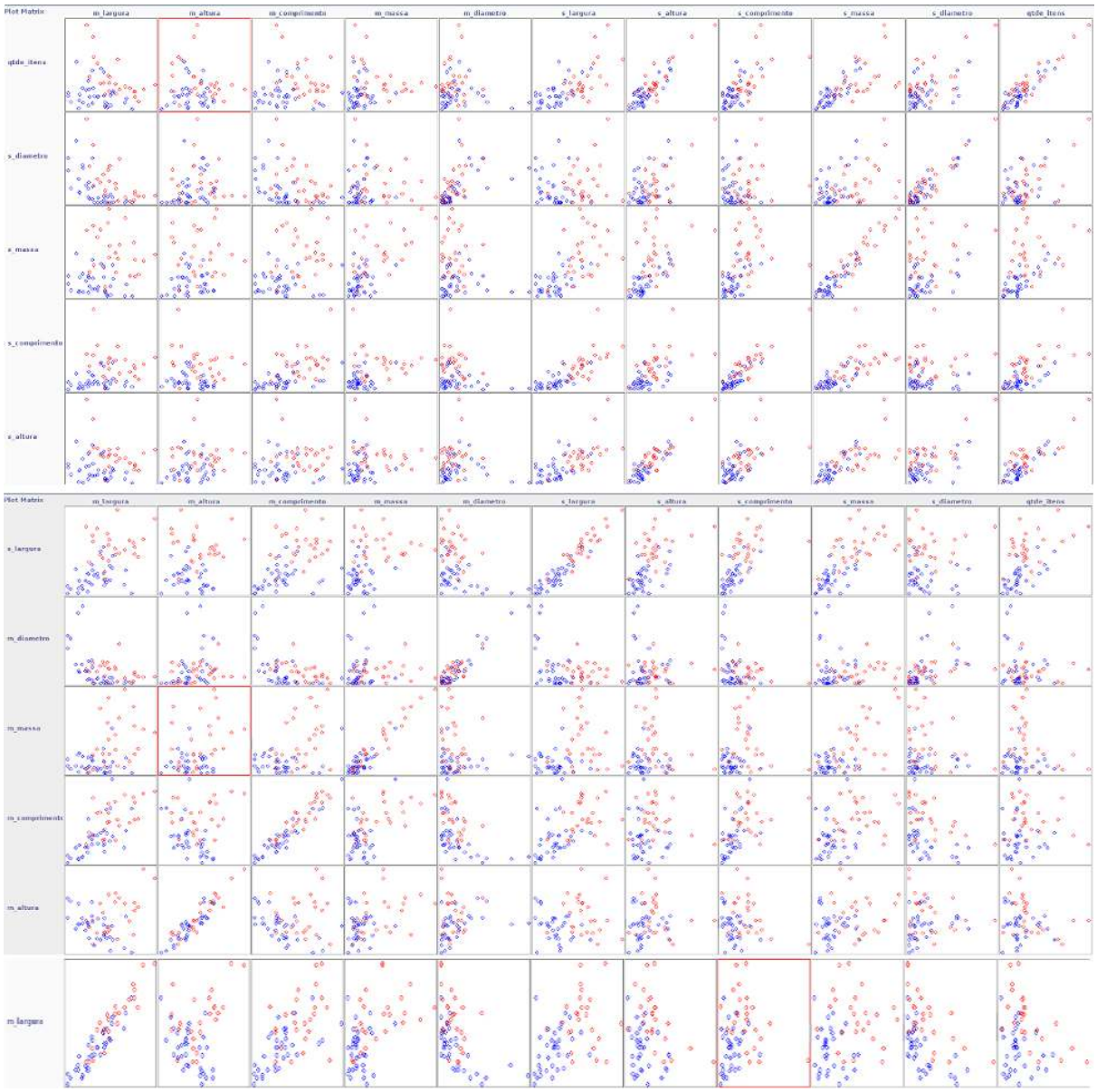

Figura 3.7: Plotagem do espaço de *features* par a par do dataset de 59 instâncias.

(médiaAltura, somaLargura - “ma x sl”). Por esso motivo essa será a curva analisada para escolher um modelo o mais adequado possível para implantar no FIM. Como o principal motivo da implantação do classificador no FIM é detectar efeitos colaterais indesejáveis, então pretende-se que o modelo tenha um alto índice de *recall*, pois não se deseja que se tenha muitos falso negativos, ou seja, muitos casos classificados como não sendo efeito colateral, mas que na verdade era. Visualizando a curva percebe-se que esta têm um crescimento acentuado e depois torna-se estável a partir do dataset com 30 instâncias. Logo fará-se uma análise a partir deste ponto. Os maiores valores de *recall* partindo deste ponto são aqueles quando o *dataset* tem 40 ou mais instâncias, para todos estes pontos considera que se obteve um bom grau de generalização. Estes valores podem ser

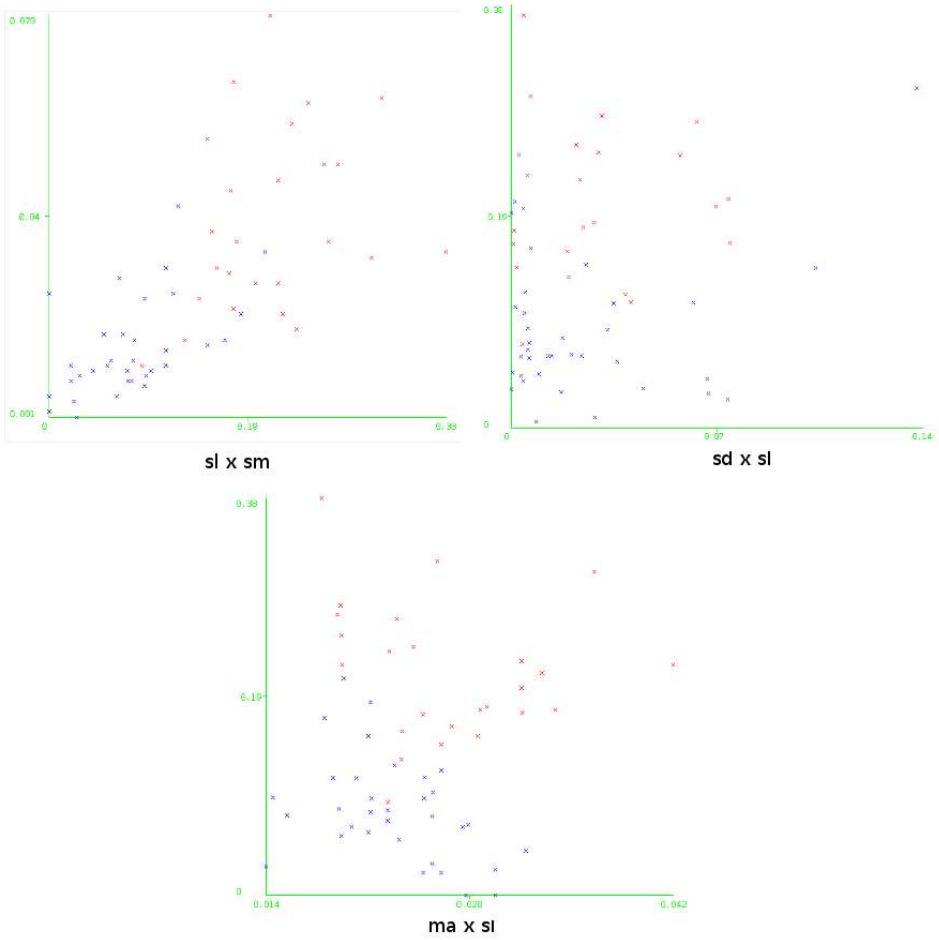

Figura 3.8: Separação das classes entre os atributos (somaLargura, somaMassa) - "sl x sm", (somaDiâmetro, somaLargura) - "sd x sl" e, (médiaAltura, somaLargura).

visualizados na Tabela 3.1.

$n^o i$	**40**	**45**	**50**	**55**	**58**
A, DP	92.5, 0	92.75, 1.78	88.8, 1.6	91.13, 1.56	92.33, 1.89
P, DP	93.5, 2.29	99.17, 1.71	93.17, 2.93	92.17, 3.25	92.5, 2.81
R, DP	91.69, 2.56	90.25, 2.21	86.33, 1.91	89.71, 2.97	92.05, 3.06

Tabela 3.1: Valores das curvas "ma x sl" da Figura 3.9 a partir do *dataset* com 40 instâncias. A (*Accuracy*), P (*Precision*, R (*Recall*), DP (Desvio Padrão), i (instâncias). Os valores A, P, R e DV estão em %.

Finalizando a análise decidiu-se que o melhor ponto para poder se construir um classificador é o com $n^o i = 45$, pois além de se ter um alto grau de *recall* e *accuracy* têm-se também *precision* quase de 100%. Então, para se construir um classificador final com base neste modelo escolheu-se randomicamente 45 instâncias do *dataset* de 59 instâncias

e aplicou-se *stratified-10-fold-cross-validation* (repetido dez vezes). Este procedimento foi repetido diversas vezes (Figura 3.10) até que se obtivesse um modelo que satisfizesse a expressão 3.1. O modelo que satisfez essa expressão obteve resultados que podem ser vistos na Tabela 3.2 e este será o modelo final de classificador que será implantado no FIM.

$$(accuracy \geq (92.75 - 1.78)) \wedge (precision \geq (99.17 - 1.71)) \wedge (recall \geq 90.25) \qquad (3.1)$$

		DP %
Accuracy %	98.05	1.81
Precision %	98.5	3.2
***Recall* %**	97.67	2

Tabela 3.2: Valores da validação do modelo final. DP (Desvio Padrão).

3.4 RESULTADOS

Como provado na seção 3.3 é possível construir um modelo de classificador que tenha um alto grau de generalização para o problema observado na seção 3.1.2. Sendo assim este foi implantado no FIM do HNS (Figura 3.1) afim de detectar efeitos colaterais indesejáveis no cenário da Figura 3.2.

O classificador atua exatamente no momento em que o serviço veículo responde a requisição do FIM na etapa 3 da Figura (3.2). Nesta etapa o veículo primeiramente captura o "id_cesta" (banco de dados) passado a ele pelo FIM. Então o veículo utiliza uma funcionalidade do serviço FIM que captura a cesta correspondente ao "id_cesta", transforma-a em um arquivo "arff" contendo somente uma instância com os parâmetros correspondentes a "somaLargura, mediaAltura, classe", sendo a classe com valor não rotulado (valor esquecido), ou seja, não sabe-se a qual classe pertence ("é efeito colateral" ou "não é efeito colateral"). O FIM então salva este arquivo e retorna o caminho do arquivo como resposta para o dispositivo veículo. O veículo então responde a requisição 3, sendo um dos dados o caminho do arquivo "arff" referente a cesta. Neste momento o FIM utiliza o classificador (uma funcionalidade interna) para predizer se "é efeito colateral" ou "não é efeito colateral" passando como parâmetro ao classificador o caminho do arquivo "arff". Caso o classificador classifique como "é efeito colateral" o FIM interrompe o serviço e manda uma mensagem para o usuário. Caso contrário o fluxo do cenário (Figura 3.2) continua.

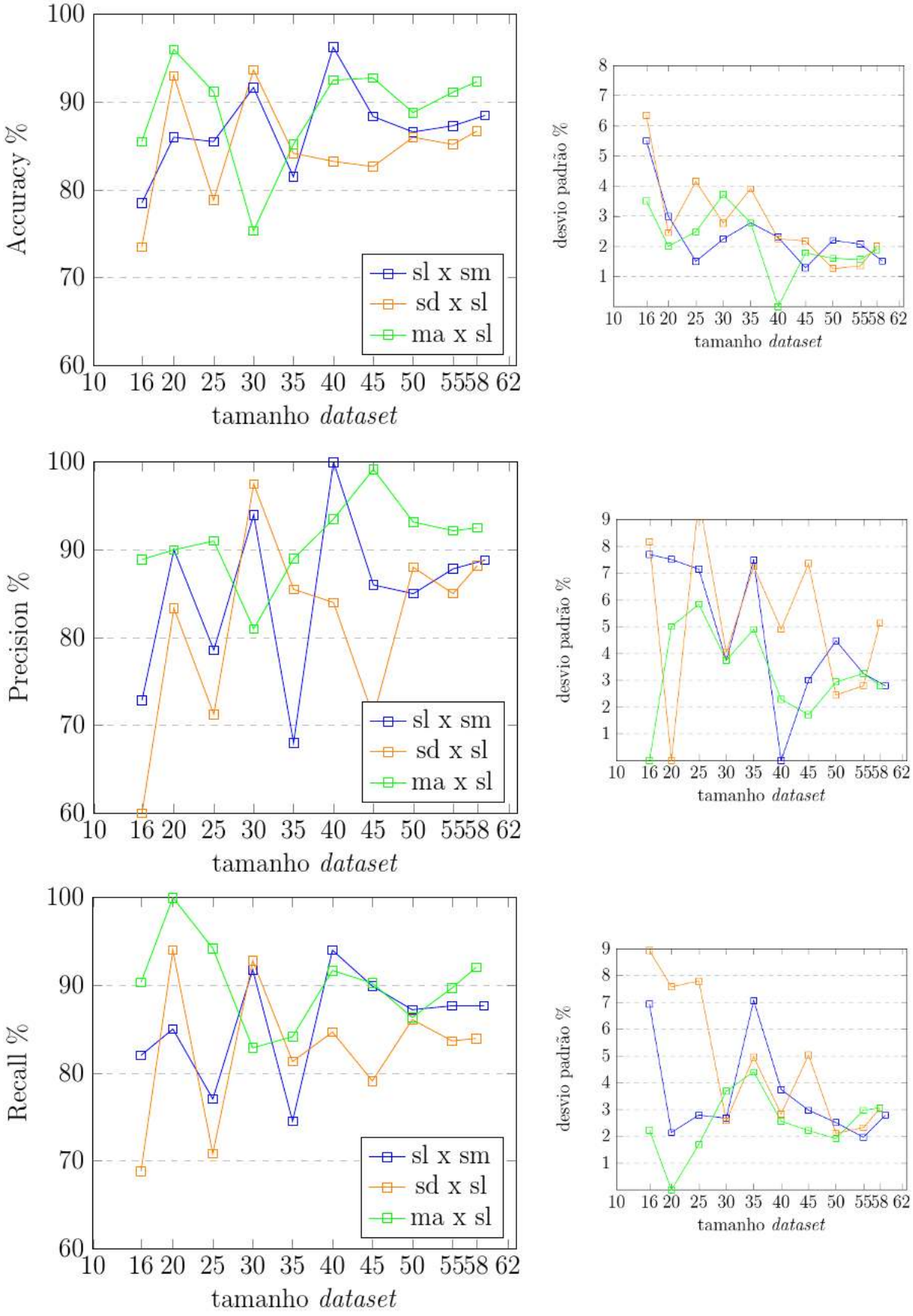

Figura 3.9: *Accuracy*, Precision e *Recall* obtidas utilizando stratified-10-fold-crossvalidation (repetido 10 vezes) e DECORATE sobre diferentes frações do *dataset* com os pares de parâmetros selecionados na seção 3.2.

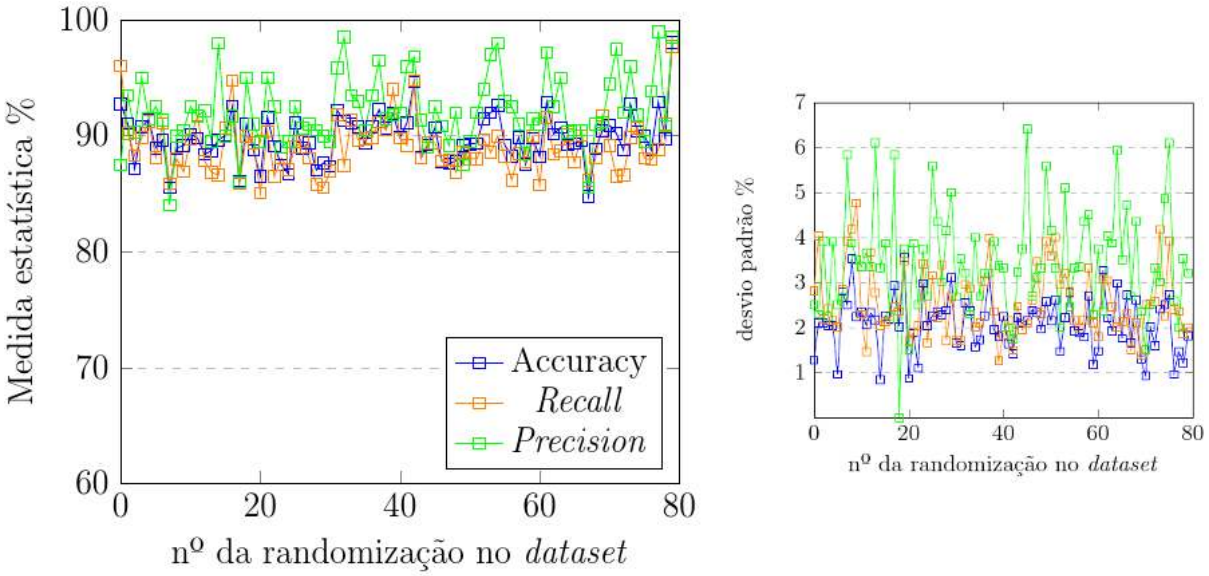

Figura 3.10: *stratified-10-fold-cross-validation* (repetido dez vezes). 45 instâncias foram escolhidas randomicamente em cima do *dataset* de 59 instâncias. Procedimento repetido até que satisfizesse a expressão 3.1

Capítulo

4

Síntese da investigação e dos experimentos realizados nesta monografia

CONCLUSÃO

O presente trabalho demonstrou através de um cenário ("Levar compras" - seção 3.1) no HNS que quando dispositivos são compostos para prover uma única funcionalidade, o comportamento de pelo menos um destes pode provocar interação de características com efeitos colaterais indesejáveis. Foi proposto detectar tais efeitos colaterais indesejáveis diante do cenário apresentado utilizando-se das mensagens trocadas entres os dispositivos, mais especificamente as informações dos objetos que estavam a ser transportados pelo veículo. A partir dessas informações foi realizado um processo classificatório (seções 3.1.2, 3.2, 3.3) a fim de se construir um modelo final de classificador que tivesse alto grau de generalização para o problema de detectar efeitos colaterais indesejáveis no cenário em questão. Por fim, tal modelo foi construído e implantado no componente de software, FIM, do HS, mostrando que é possível detectar efeitos colaterais indesejáveis entre dispositivos no cenário da IoT.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, J. et al. An overview of formal methods tools and techniques. In: _____. *Rigorous Software Development: An Introduction to Program Verification*. London: Springer London, 2011. p. 15–44. ISBN 978-0-85729-018-2. Disponível em: ⟨http://dx.doi.org/10.1007/978-0-85729-018-2_2⟩.

ARORA, R.; SUMAN, S. Comparative analysis of classification algorithms on different datasets using weka. *International Journal of Computer Applications*, v. 54, n. 13, 2012.

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

BELQASMI, F.; FU, C.; GLITHO, R. Restful web services for service provisioning in next generation networks: A survey. *IEEE COMMUNICATIONS MAGAZINE, NETWORK and SERVICE MANAGEMENT*, 2011.

BELQASMI, F. et al. Soap-based web services vs. restful web services for multimedia conferencing applications: A case study. *IEEE INTERNET COMPUTING*, 2012. Disponível em: ⟨http://spectrum.library.concordia.ca/980040/1/PersonalCopy-SOAPvsREST.pdf⟩.

BLAND, J.; ALTMAN, D. Measurement error. *BMJ*, v. 313, p. 744, 1996.

CALDER, M. et al. Feature interaction: a critical review and considered forecast. *Computer Networks*, v. 41, p. 115–141, 2003. Disponível em: ⟨http://eprints.gla.ac.uk/2874/1/feature1calder.pdf⟩.

CHANDRAKANTH, S. et al. Internet of things. *International Journal of Innovations and Advancement in Computer Science IJIACS*, v. 3, October 2014.

DAVIS, J.; GOADRICH, M. The relationship between precision-recall and roc curves. In: *Proceedings of the 23rd International Conference on Machine Learning*. New York, NY, USA: ACM, 2006. (ICML '06), p. 233–240. ISBN 1-59593-383-2. Disponível em: ⟨http://doi.acm.org/10.1145/1143844.1143874⟩.

DUSTDAR, S.; SCHREINER, W. A survey on web services composition. *Int. J. Web and Grid Services*, v. 1, p. 1–30, 2005.

ELIZONDO, D. The linear separability problem: Some testing methods. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, v. 17, n. 2, March 2006. Disponível em: ⟨http://sci2s.ugr.es/keel/pdf/specific/articulo/IEEETNN06.pdf⟩.

FIRDAUSI, I. et al. Analysis of machine learning techniques used in behavior-based malware detection. In: *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*. [S.l.: s.n.], 2010. p. 201–203.

FRANCA, T. et al. Web das coisas: Conectando dispositivos físicos ao mundo digital. In: . Porto Alegre, RS: SBC: [s.n.], 2011. v. 1, p. 103–146.

GUDGIN, M. et al. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. 2007. Disponível em: ⟨https://www.w3.org/TR/soap12⟩.

GUINARD, D.; TRIFA, V. Towards the web of things: Web mashups for embedded devices. In: *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web, International World Wide Web Conferences*. Madrid, Spain: [s.n.], 2009.

HALL, M. et al. The weka data mining software: An update. *SIGKDD Explorations*, v. 11, 2009.

HEFFELFINGER, D. *Java EE 7 with GlassFish 4 Application Server*. Third. [S.l.]: Packt Publishing Ltd, 2014.

IKEGAMI, K.; MATSUMOTO, S.; NAKAMURA, M. Considering impacts and requirements for better understanding of environment interactions in home network services. *Computer Networks*, Elsevier, v. 57, p. 2442–2453, 2013.

INTERNET of Things in 2020: Roadmap for the Future. INFSO D.4 Networked Enterprise and RFID INFSO G.2 Micro and Nanosystems, in: Co-operation with the Working Group RFID of the ETP EPOSS. 2008.

KARTHIKEYAN, T.; THANGARAJU, P. Analysis of classification algorithms applied to hepatitis patients. *International Journal of Computer Applications*, v. 62, n. 15, 2013.

KERR, A.; HALL, H.; KOZUB, S. *Doing Statistics with SPSS*. [S.l.]: SAGE Publications, 2002. ISBN 0 7619 7384 2.

KOTSIANTIS, S. Supervised machine learning: A review of classification techniques. *Informatica*, v. 31, p. 249–268, 2007. Disponível em: ⟨http://www.informatica.si/index.php/informatica/article/view/148/140⟩.

MELVILLE, P.; MOONEY, R. J. Constructing diverse classifier ensembles using artificial training examples. In: *Eighteenth International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 2003. p. 505–510.

MELVILLE, P.; MOONEY, R. J. Creating diversity in ensembles using artificial data. *Information Fusion: Special Issue on Diversity in Multiclassifier Systems*, 2004. Submitted.

METZ, C. Basic principles of roc analysis. *Seminars in Nuclear Medicine*, v. 8, n. 4, p. 283–298, 1978. ISSN 0001-2998. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0001299878800142⟩.

MICHIE, D.; SPIEGELHALTER, D.; TAYLOR, C. (Ed.). *Machine Learning, Neural and Statistical Classification*. [s.n.], 1994. Disponível em: ⟨http://www1.maths.leeds.ac.uk/~charles/statlog/⟩.

NAKAMURA, M. et al. Considering online feature interaction detection and resolution for integrated services in home network system. *Feature Interactions in Software and Communication Systems X*, p. 191–206, 2009.

NHLABATSI, A.; LANEY, R.; NUSEIBEH, B. Feature interaction: the security threat from within software systems. *Progress in Informatics*, n. 5, p. 75–89, 2008. Disponível em: ⟨http://www.nii.jp/pi/n5/5_75.pdf⟩.

OLSON, D.; DELEN, D. *Advanced Data Mining Techniques*. [S.l.]: Springer-Verlag Berlin Heidelberg, 2008. ISBN 978-3-540-76916-3.

PAPAZOGLOU, M. *Slides from Web Services: Principles and Technology*. 2008. Disponível em: ⟨https://www.cs.colorado.edu/~kena/classes/7818/f08/lectures/lecture_3_soap.pdf⟩.

PAPAZOGLOU, M. *Web Services: Principles and Technology*. [S.l.]: Pearson, 2008. ISBN 978-0-321-15555-9.

PAUTASSO, C. Restful web services: Principles, patterns, emerging technologies. In: *Springer Science+Business Media*. New York: [s.n.], 2014. Disponível em: ⟨http://vis.uky.edu/~cheung/courses/ee586/papers/Pautasso2014.pdf⟩.

PIYARE, R. Internet of things: Ubiquitous home control and monitoring system using android based smart phone. *International Journal of Internet of Things*, Elsevier, p. 5–11, 2013.

QUINLAN, R. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

SIEGMUND, N. et al. Predicting performance via automated feature-interaction detection. In: *34th International Conference on Software Engineering (ICSE '12)*. Piscataway, NJ, USA: IEEE Press, 2012. p. 167–177.

SIEGMUND, N. et al. Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, v. 20, n. 3, p. 487–517, 2012. ISSN 1573-1367. Disponível em: ⟨http://dx.doi.org/10.1007/s11219-011-9152-9⟩.

SUNDMAEKER, H. et al. (Ed.). *Vision and Challenges for Realising the Internet of Things*. [S.l.]: European Union, 2010. ISBN 978-92-79-15088-3.

THUM, T. et al. A classification and survey of analysis strategies for software product lines. *ACM Comput. Surv.*, v. 47, May 2014.

W3C. *Extensible Markup Language (XML)*. 2004. Disponível em: ⟨https://www.w3.org/XML/⟩.

W3C. *XML Schema*. 2004. Disponível em: ⟨https://www.w3.org/XML/Schema⟩.

WEBER, R. H.; WEBER, R. *Internet of Things - Legal Perspectives*. [S.l.]: Springer, 2010.

WEISS, M.; ESFANDIARI, B.; LUO, Y. Towards a classification of web service feature interactions. *Computer Networks*, v. 51, p. 359–381, 2007.

WEISS, M.; ORESHKIN, A.; ESFANDIARI, B. Invocation order matters: Functional feature interactions of web services. In: *Proceedings of the First International Workshop on Engineering Service Compositions*. Amsterdam, The Netherlands: [s.n.], 2005. p. 69–76.

WILSON, M.; MAGILL, E.; KOLBERG, M. An online approach for the service interaction problem in home automation. *Consumer Communications and Networking Conference*, IEEE, 2005.

WILSON, M.; MAGILL, E.; KOLBERG, M. Considering side eects in service interactions in home automation - an online approach. In: _____. *Feature Interactions in Software and Communication Systems IX*. [S.l.]: IOS Press, 2008. p. 172–187. ISBN 978-1-58603-845-8.

WITTEN, I.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*. second. [S.l.]: ELSEVIER, 2005. ISBN 0-12-088407-0.

ZHANG, G. Neural networks for classification: A survey. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C: APPLICATIONS AND REVIEWS*, v. 30, n. 4, November 2000.