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

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.

33p.: il.

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

"1. Efeitos Colaterais. 2. Internet das Coisas. 3. Home Network System. 4. Serviço WEB. 5.Dispositivos. 6.REST. 7.Rede Neural. 8.Multilayer Perceptron".
I. Claro, Daniela Barreiro. II. UNIVERSIDADE FEDERAL DA BAHIA. INSTITUTO DE MATEMÁTICA. III Título.

CDD 005.13307

TERMO DE APROVAÇÃO

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 julgado adequado à obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Programa de Graduação em Ciência da Computação da Universidade Federal da Bahia.

Salvador, 31 de outubro de 2016

Profa. Dra. Daniela Barreiro Claro
Universidade Federal da Bahia

Prof. Dr. Cássio Vinicius Serafim Prazeres
Universidade Federal da Bahia

Prof. Dr. Raimundo José de Araújo Macêdo
Universidade Federal da Bahia

TODO

AGRADECIMENTOS

O que sabemos é uma gota, o que ignoramos é um oceano.

—ISAAC NEWTON (1687)

RESUMO

Palavras-chave: TODO

ABSTRACT

Keywords: TODO

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

Capítulo

1

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

INTRODUÇÃO

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 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).

No âmbito da economia, por exemplo, o E-Commerce permitiu potencializar as vendas de produtos e serviços, com um faturamento estimado para o ano de 2016 de aproximadamente 56,8 bilhões de reais no Brasil, segundo ecommercenews[1]. Além do benefício direto para sociedade provindo do E-Commerce, onde as pessoas podem realizar pesquisa de preços de serviços e produtos e, adquiri-los de forma cômoda, sem precisar se deslocar até um ponto de venda, a Internet dispõe para a sociedade diversas outras oportunidades, como cursos a distância oferecidos por diversas universidades de todo o mundo, a exemplo dos cursos disponibilizados pela plataforma Coursera[2].

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 (presentes em locais que não tinham e, mais baratas)(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[3], 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

[1] https://ecommercenews.com.br/noticias/pesquisas-noticias/e-commerce-brasileiro-deve-crescer-18-e-faturar-r-568-bilhoes-em-2016

[2] ⟨https://pt.coursera.org/⟩

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

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). 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, tanto a atual realidade, quanto a prospecção futura. Já a segunda especifica melhor como deve ser o cenário ideal da Internet das Coisas, pois já embuti explicitamente os conceitos de capacidades de autoconfiguração, interoperabilidade e interfaces inteligentes.

1. Segundo(Nanosystems, 2008), *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 (Sundmaeker et al., 2010).

Diante deste cenário da IoT, pode-se citar exemplos de aplicações em 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. Abaixo se tem uma breve descrição de duas 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)

Apesar da grande potencialidade da visão 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[4], 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 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, pois cada coisa possui suas próprias interfaces, logo para cada dispositivo um desenvolvimento a parte. 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 (seção 2.1.4), assim sendo se pode utilizar os protocolos WEB como linguagem comum de integração dos dispositivos a Internet.(Franca et al., 2011)

2.1.1 Serviços Web

Segundo(Dustdar e Schreiner, 2005), um serviço Web é um sistema identificado por uma URL[5], no qual suas interfaces públicas são definidas e descritas usando XML[6] e, suas definições podem ser descobertas por outros sistemas. Sistemas então podem interagir com os serviços Web utilizando suas definições e descrições. Através destas utiliza-se mensagens XML que são transmitidas seguindo os protocolos padrão (definidos por W3C[7]) 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. Para melhor entender o propósito dos serviços Web, considere, como exemplo, uma companhia de seguros que decidiu oferecer um serviço de cotação on-line. Então em vez de desenvolver toda a aplicação do início, a empresa utiliza-se do serviço Web de outra empresa (especializada em cálculos de seguro). Este serviço pode, por

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

[5] 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⟩

[6] 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)

[7] ⟨https://www.w3.org/⟩

exemplo, oferecer um formulário de cotação e, após o cliente informar os dados necessários, o serviço Web apresentará uma cotação baseado nos dados informados. Além disso, caso o cliente escolher comprar tal seguro, pode-se ainda utilizar de outro serviço Web, oferecido por uma outra empresa, o qual processará o pagamento de acordo com os dados passados para ele e então retorna o resultado do processamento para o cliente e para a empresa original(que desejou construir o serviço de cotação on-line)(Papazoglou, 2008b). O cenário da Figura 2.1 demonstra um outro exemplo de uso de diferentes serviços para prover um único serviço, o de um pedido de uma ou mais mercadorias, ambiente comum do E-Commerce.

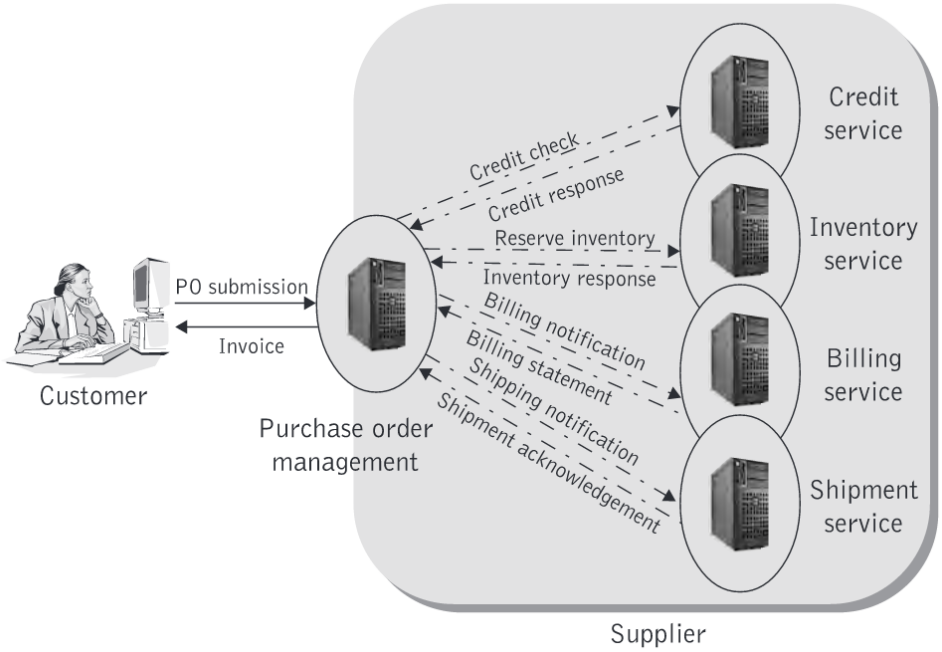

Figura 2.1: Utilização de diferentes serviços Web para prover o serviço de pedido de um E-Commerce.(Papazoglou, 2008b)

O cenário do serviço(Figura 2.1) pode ser descrito da seguinte forma:

1. *Consumer*(Consumidor) escolhe seu(s) produto(s), informa dados do cartão de crédito e outras informações necessárias e, clica no botão comprar. Requisição é enviada ao *Purchase order management*(Gerenciador de pedidos de compra).

2. O gerenciador então verifica se o cartão do cliente tem crédito suficiente através da funcionalidade *Credit check* do serviço *Credit Service* e, o serviço dá uma resposta.

3. Gerenciador capta a resposta e, caso cliente tenha crédito suficiente, inicia o processo de reserva do pedido através da funcionalidade textitReserve inventory do serviço *Invetory service*.

4. Gerenciador capta resposta *Inventory response*, caso o pedido tenha sido reservado com êxito, o gerenciador utiliza o *Billing service* para processar o pagamento.

5. *Billing service* retorna o estado do pagamento, se pagamento foi processado com êxito o gerenciador utiliza agora a funcionalidade *Shipping notification* do serviço *Shipment service* para prover o serviço de entrega.

6. Gerenciador recebe a resposta *Shipment acknowledgement* e informa ao cliente (*Invoice*) a data prevista para entrega do(s) produto(s) do cliente.

No cenário apresentado anteriormente há uma ressalva que deve ser feita, a qual dita uma das características de um serviço Web (serviço assíncrono ou síncrono, descrito logo a seguir), no item 5, por exemplo, o pagamento pode demorar mais de um dia para concluir seu processamento, então o cliente poderia receber um retorno do sistema do tipo, pagamento aguardando processamento, ou seja, a resposta do processamento do pagamento(realizado ou não realizado), poderia vir através de um envio de e-mail ou SMS, somente depois de muito tempo.

Serviços Web possuem algumas características que se fazem importantes saber:

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 é denominado textitstateless. Como exemplo de serviço textitstateless 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 textitHome 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 (textitloose 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).

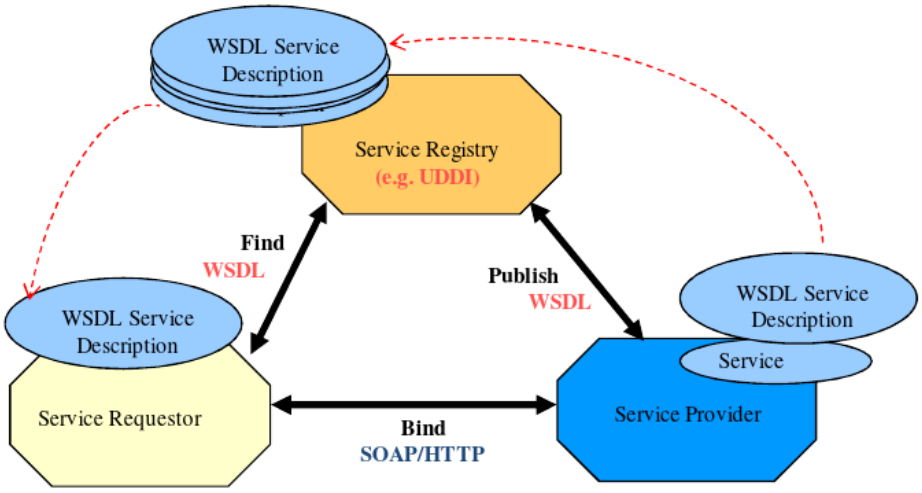

Figura 2.2: Modelo de serviço Web baseado em SOAP.(Belqasmi et al., 2011)

- 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 (ver 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.

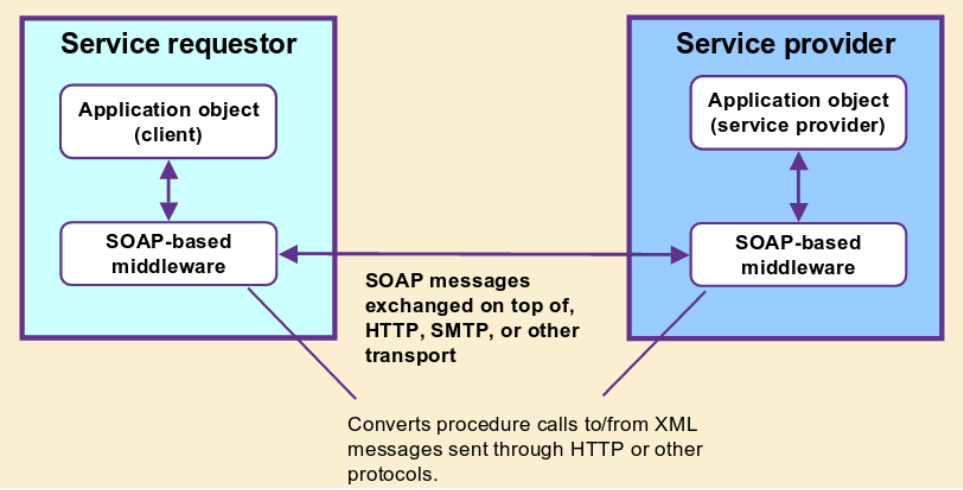

Figura 2.3: SOAP no modelo Modelo de serviço Web baseado em SOAP.(Papazoglou, 2008a)

Em outras palavras, SOAP é um protocolo de rede de aplicação utilizado para transferir mensagens entre instâncias de serviços descritas por interfaces WSDL (ver Figura 2.4).

Um documento XML SOAP é chamado de SOAP mensagem (ou SOAP envelope) e é usualmente transportado sobre outros protocolos de aplicação, mais frequentemente o HTTP. Mas existem outros protocolos de aplicação que podem ser utilizados: SMTP[8], FTP[9], RMI[10]. A mensagem SOAP, representada na Figura 2.4 é colocada no corpo da mensagem HTTP e então é encaminhada para seu destino. Na próxima camada, da hierarquia ilustrada na figura 2.4 (camada de transporte no modelo OSI[11]), a mensagem HTTP torna-se segmentos de bytes a serem enviados através do stream do TCP. Do outro lado (destinatário) um *HTTP listener*(fica a espera de novas mensagens) passa o corpo da mensagem HTTP para um processador de mensagens SOAP, o qual entende a sintaxe das mensagens e então é capaz de processá-las.(Papazoglou, 2008b)

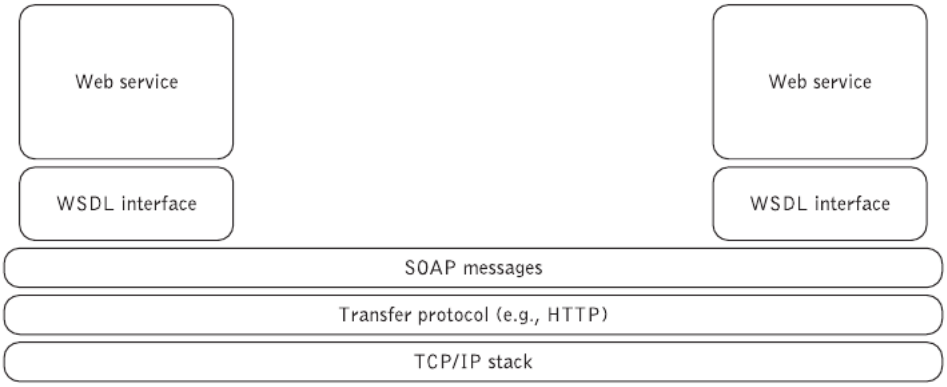

Figura 2.4: Comunicação dos serviços Web e mensagem SOAP.(Papazoglou, 2008b)

Uma mensagem SOAP (ver Figura 2.5) é composta de um elemento <Envelope>(elemento raiz), o qual contém um <Header>(opcional) e um obrigatório <Body>. Caso um <Header>seja utilizado, este deve ser um filho imediato de <Envelope>, e preceder o <Body>. <Body>contém os dados principais da aplicação (dados arbitrários em um XML ou parâmetros para uma chamada de método). Uma mensagem SOAP pode ter uma declaração XML, <?xml version="1.0" encoding="UTF-8"?>, a qual indica a versão do XML usado e o formato da codificação. Para que a tag <env:Header ou Body ou Envelope>seja identificada como pertencente ao SOAP *namespace*, esta deve ser referenciada ao SOAP *namespace* URI, xmlns:env="http://www.w3.org/2003/05/soap-envelope". Se uma aplicação SOAP recebe uma mensagem que é baseada em outro *namespace*, isto gerará um erro.(Papazoglou, 2008b)

O envelope SOAP pode especificar um conjunto de regras que serializam os dados XML da aplicação. Assim, tanto o Provedor quanto o Consumidor devem concordar com as regras de codificação (normalmente baixando o mesmo esquema XML que define eles).

[8]Simple Mail Transfer Protocol é um protocolo utilizado para transportar e-mail de forma confiável e eficiente. ⟨https://www.ietf.org/rfc/rfc0821.txt⟩

[9]⟨https://tools.ietf.org/html/rfc959⟩

[10]⟨https://docs.oracle.com/javase/tutorial/rmi/⟩

[11]Open Systems Interconnection (OSI) - modelo de referência para comunicação dividido em sete camadas.(Kurose e Ross, 2013)

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:env="http://www.w3.org/2003/05/soap-envelope">

   <env:Header> <!-- optional -->
    <!-- header blocks go here . . . -->
   </env:Header>

   <env:Body>
    <!-- payload or Fault element goes here . . . -->
   </env:Body>
</env:Envelope>
```

Figura 2.5: Estrutura da mensagem SOAP.(Papazoglou, 2008b)

Para tal finalidade ambos podem utilizar-se do atributo global(pode ser declarado em qualquer nível do documento XML) *encodingStyle*, este define qual estilo de codificação deve ser utilizado no elemento no qual foi definido e em seus filhos (ver Figura 2.6). Com o intuito de favorecer a flexibilidade SOAP permite que as aplicações definam seu próprio estilo de codificação, mas na maioria dos casos as regras de codificação padrão são indicadas.(Papazoglou, 2008b)

```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    env:encodingStyle="http://schemas.xmlsoap.org/soap/
                                              encoding/">
        …
</env:Envelope>
```

Figura 2.6: Estrutura da mensagem SOAP com declaração do atributo encodingStyle.(Papazoglou, 2008b)

O envelope SOAP pode conter no máximo um elemento <Header>, o qual contém todas as dicas que são relevantes para seus destinatários finais ou pontos de transporte intermediários. Também pode conter informação de onde o documento deve ser encaminhado, origem, e pode conter assinaturas digitais. O propósito do <Header>é encapsular extensões ao formato da mensagem sem combiná-las aos dados principais de carga da aplicação ou modificar a estrutura fundamental SOAP. Isto permite adição de especificações de características e funcionalidades, tais como segurança, transporte, referências a objetos, atributos QoS[12], dentre outros, sem quebra das especificações da mensagem SOAP. Cada bloco dentro do <Header>deve ter seu próprio *namespace*, assim como já explanado anteriormente, o *namespace* permite que as aplicações SOAP identifiquem o bloco e os processe de acordo com as regras definidas no *namespace* em questão(ver Figura

[12] *Quality of Service* garante largura de banda a cada instância de uma determinada aplicação.(Kurose e Ross, 2013)

2.7).(Papazoglou, 2008b)

```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
      …
    <env:Header>
       <tx:transaction-id
           xmlns:tx="http://www.transaction.com/transaction"
             env:mustUnderstand="true">
                512
       </tx:transaction-id>
       <notary:token xmlns:notary="http://www.notarization-
                                        services.com/token"
              env:mustUnderstand="true">
                 GRAAL-5YF3
       </notary:token>
    </env:Header>
       …
</env:Envelope>
```

Figura 2.7: Estrutura da mensagem SOAP, Header.(Papazoglou, 2008b)

O <Body>pode ter um número arbitrário de filhos (chamados de entradas do <Body>), como também pode conter nenhum. Todas as entradas do <Body>que são filhos imediatos devem ter o *namespace*. Por padrão, o conteúdo do <Body>pode ser um XML qualquer sem especificação de regras. <Body>ou contém os dados principais da aplicação ou uma mensagem de erro, mas não os dois. Ou ainda pode ser vazio.(Papazoglou, 2008b)

SOAP suporta dois tipos de estilo de comunicação, RPC (Remote Procedure Call) e documento (ou mensagem). Numa comunicação no estilo RPC/Literal, o cliente expressa sua requisição como uma chamada de método passando um conjunto de parâmetros. O método então retorna uma resposta. Algumas regras nesse estilo de comunicação devem ser seguidas e são explanadas abaixo e exemplificadas na Figura 2.8.

- Uma URI deve identificar o endereço de destino onde o método será chamado, que pode ser identificado na declaração da tag <Envelope>, xmlns:m=“http://www.plastics$_s upply.c$ $prices''.Deveserpassadoonomedométodo, <m : GetProductPriceResponse>e, seusparâmetros$ $id>450R6OP<product - id>.$

- Uma mensagem de resposta RPC deve conter um valor de retorno e quaisquer parâmetros de saída ou um erro. Esta é modelada como uma estrutura única com um campo para cada parâmetro na assinatura do método, exemplificado em <product-price>134.32 <product-price>.

Em uma comunicação no estilo documento, SOAP é utilizado para trocar documentos contendo qualquer tipo de dados XML no <Body>, com o XML não codificado. No estilo de mensagem *Document/Literal*, o <Body>contém um elemento XML bem formado com dados da aplicação de qualquer tipo. O ambiente de execução SOAP aceita o elemento <Body>da maneira com que chegou e o trata sobre a aplicação a qual foi destinado.

a)

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
       <tx:Transaction-id
           xmlns:t="http://www.transaction.com/transactions"
           env:mustUnderstand='1'>
              512
       </tx:Transaction-id>
    </env:Header>
    <env:Body>
       <m:GetProductPrice>
          <product-id> 450R6OP </product-id >
       </m:GetProductPrice >
    </env:Body>
</env:Envelope>
```

b)

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
       <--! - Optional context information -->
    </env:Header>
    <env:Body>
       <m:GetProductPriceResponse>
          <product-price> 134.32 </product-price>
       </m:GetProductPriceResponse>
    </env:Body>
</env:Envelope>
```

Figura 2.8: a)Estilo SOAP RPC <Body>. b) Estilo SOAP RPC <Body>- resposta(Papazoglou, 2008b)

Pode ou não ter uma resposta associado com a mensagem. A Figura 2.9 é um exemplo de mensagem neste estilo, a qual exemplifica os dados de um pedido de compra. Observa-se que este tipo de envio de dados não é muito bem conduzido para o estilo RPC, como o demonstrado na Figura 2.8. Em vez disso, a aplicação almeja transferir os dados da aplicação de uma única vez a um serviço para processamento futuro.(Papazoglou, 2008b)

As requisições SOAP normalmente utilizam o protocolo de transporte HTTP. Quando utilizado as requisições SOAP são transportadas dentro do corpo do método POST do HTTP e suas respostas são retornadas nas respostas do HTTP (ver Figuras 2.10 e 2.11).

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, mensagens auto-descritivas e *hypermedia* como um mecanismo para descoberta descentralizada de recursos por referência(Pautasso, 2014). Na lista abaixo se têm uma explanação a respeito dos princípios citados.

1. Identificação dos recursos - todos os recursos que são publicados por um serviço Web

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">

    <env:Header>
      <tx:Transaction-id
          xmlns:t="http://www.transaction.com/transactions"
          env:mustUnderstand='1'>
              512
    </env:Header>
    <env:Body>
      <po:PurchaseOrder orderDate="2004-12-02"
          xmlns:m="http://www.plastics_supply.com/POs">
          <po:from>
            <po:accountName> RightPlastics </po:accountName>
            <po:accountNumber> PSC-0343-02 </po:accountNumber>
          </po:from>
          <po:to>
            <po:supplierName> Plastic Supplies Inc.
                                          </po:supplierName>
            <po:supplierAddress> Yara Valley Melbourne
                                        </po:supplierAddress>
          </po:to>
          <po:product>
            <po:product-name> injection molder </po:product-name>
            <po:product-model> G-100T </po:product-model>
            <po:quantity> 2 </po:quantity>
          </po:product>
      </ po:PurchaseOrder >
    </env:Body>
</env:Envelope>
```

Figura 2.9: Exemplo de um <Body>no estilo documento SOAP.(Papazoglou, 2008b)

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

```
POST /Purchase Order HTTP/1.1
Host: http://www.plastics_supply.com  <! - Service provider -- >
Content-Type:application/soap+xml;
charset = "utf-8"
Content-Length: nnnn

<?xml version="1.0" ?>
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
       <tx:Transaction-id
           xmlns:t="http://www.transaction.com/transactions"
           env:mustUnderstand='1'>
               512
       </tx:Transaction-id>
    </env:Header>
    <env:Body>
       <m:GetProductPrice>
          <product-id>  450R6OP  </product-id >
       </m:GetProductPrice >
    </env:Body>
</env:Envelope>
```

Figura 2.10: Requisição SOAP no estilo RPC dentro do corpo da requisição HTTP.(Papazoglou, 2008b)

```
HTTP/1.1 200 OK
Content-Type:application/soap+xml;
charset = "utf-8"
Content-Length: nnnn

<?xml version="1.0" ?>
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
       <--! - Optional context information -->
    </env:Header>
    <env:Body>
       <m:GetProductPriceResponse>
          <product-price> 134.32 </product-price>
       </m:GetProductPriceResponse>
    </env:Body>
</env:Envelope>
```

Figura 2.11: Resposta SOAP no estilo RPC dentro do corpo da resposta HTTP.(Papazoglou, 2008b)

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. `\textit{Hypermedia} - Recursos podem ser relacionados entre si. \textit{Hyperm`

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 RESTFul. 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 [13]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 oferecendo a coisa como um recurso REST.(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 2.12. Além disto, mashups físicos[14] podem ser construídos com muito menos esforço do que as existentes abordagens, quebrando drasticamente a barreira para o desenvolvimento de aplicações com dispositivos(Guinard e Trifa, 2009). 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.

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

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

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

Figura 2.12: Dispositivo sensor de obstáculo do elevador disponibilizado como serviço. Faixa de detecção compatível com a largura do elevador utilizado na maquete do experimento 4

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 2.13), 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)

Tratando o termo “fraco acoplamento” somente como 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)

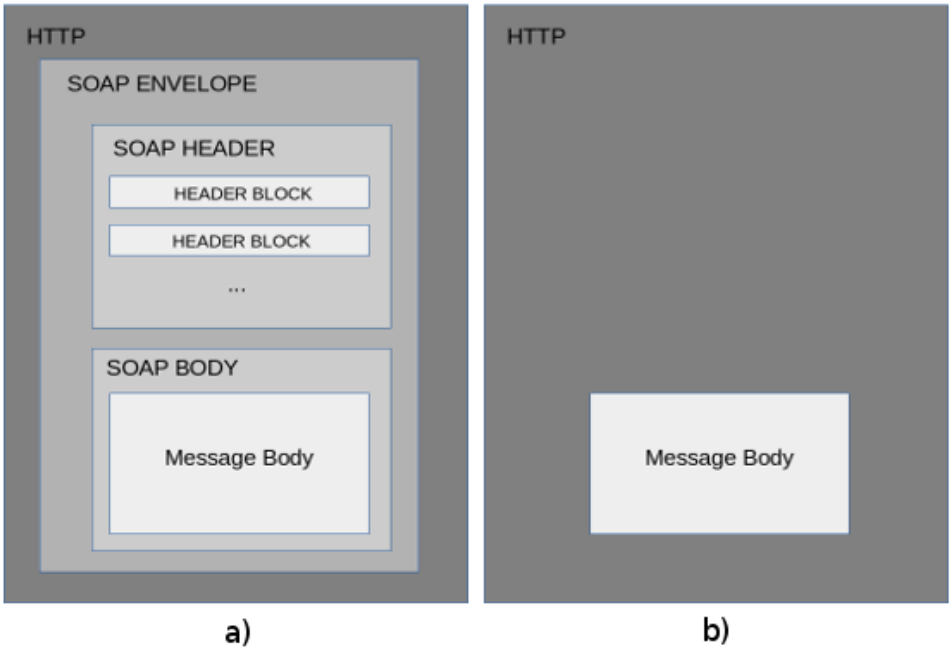

Figura 2.13: a)Mensagem SOAP sobre HTTP. b)Mensagem utilizando REST sobre HTTP. Figura baseada em (Pautasso, 2014)

Por fim, na Figura 2.14 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)

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

	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 2.14: 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)

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 desejado é quando a interação de características resulta num comportamento que ajudará de alguma 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 compremetida 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 basedas 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 textitPolicy Conflict ocorre caso exista políticas (autenticação, ou de privacidade) especificadas em duas *features* que referem-se as suas correspondentes operações, e que as politícas 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 sítese 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 experadas 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 repete-se indefinidamente, criando um loop infinito.

(NHLABATSI et al., 2008) não considera *dependence* como uma causa de interação de características, pois este diz que *dependence* passa um fraca noção de interação de características, já que esta implica que uma característica não funcione corretamente isoladamente e, segundo o autor, 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)

A engenharia de software concentra-se na criação dos serviços, então esta se preocupa com suas especificações, desenvolvimento, teste e implantação. Desta forma a criação dos serviços podem ser adaptados para auxiliar na remoção de interação de características antes de sua implantação(Calder et al., 2003). Algumas técnicas dentro do cenário da engenharia de software vêm sendo adotadas, como em (Thum et al., 2014), (Siegmund et al., 2012a) e (Siegmund et al., 2012b).

Métodos formais(Almeida et al., 2011) têm sido empregados em diversos caminhos para analisar a interação de características. Isto implica uma variedade de técnicas para tal análise, tais como, descrição formal, modelagem e raciocínio. Estas técnicas incluem: lógicas clássicas e construtivas, autômatos de estado finito e infinito, automatos estendidos, *petri-nets*, sistemas de transição e, linguagens como SDL, Promela, Z, LOTOS.(Calder et al., 2003)

Dentre as técnicas online, *Feature Interaction Manager - FIM* é uma das técnicas que vem sendo adotadas 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*.

2.3 HOME NETWORK SYSTEM

Como explanado anteriormente IoT pode ser descrita como a presença pervasiva de objetos do cotidiano, como *smartphones*, televisores, sensores, dentre outros, que juntos são interligados para prover novas formas de comunicação entre as pessoas e entre os

próprios objetos. Observe que tal visão também pode ser empregada dentro do ambiente de uma casa, tornando possível o desenvolvimento de um espaço inteligente, confortável e que promove uma melhor qualidade de vida as pessoas. 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 4). 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 (sessão2.2), 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 o FIM 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) O modelo de HNS com as características citadas aqui pode ser visusalizado na Figura 4.1.

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).

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

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)

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 2.15.

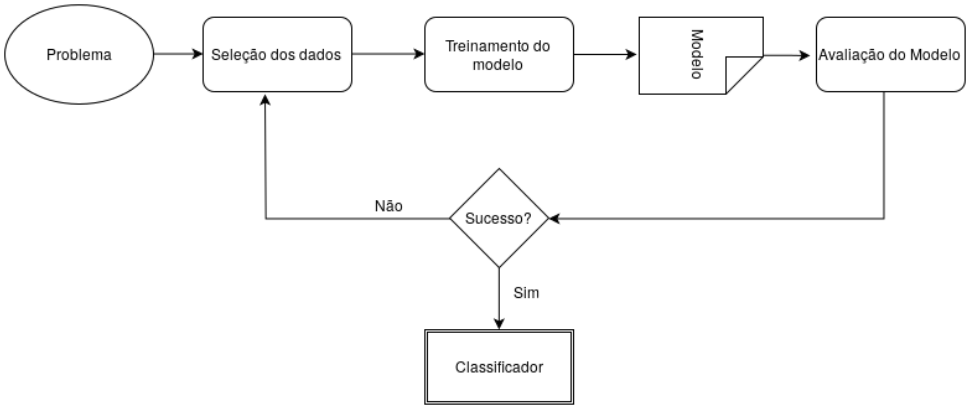

Figura 2.15: Processo de construção de um classificador em aprendizado supervisionado. Baseado em (Kotsiantis, 2007) e [15]proft.

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)

3. Treinamento do modelo: treina-se o dataset da etapa anterior sobre um algoritmo de classificação (árvores de decisão, *Fisher's linear discriminants*,(Michie et al., 1994), redes neurais (seção 2.4.1). 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 2.15, 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 2.16, ilustra problema linearmente separável e problema não separável linearmente.

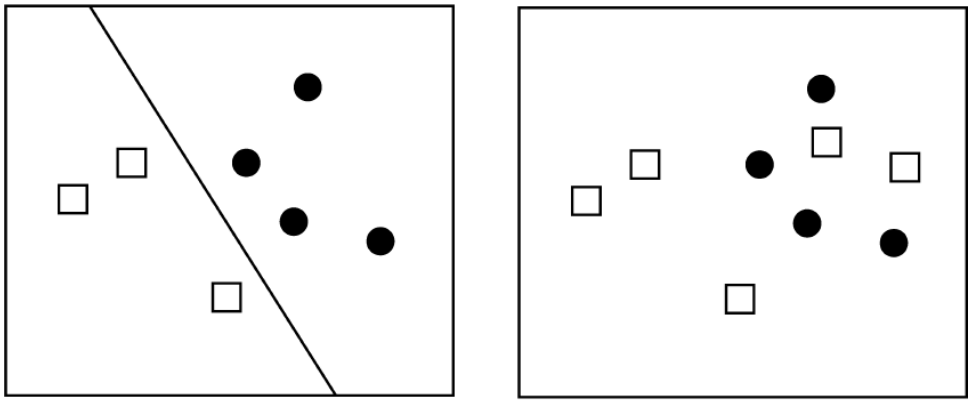

Figura 2.16: (a) Problema linearmente separável. (b) Problema não separável linearmente.(Elizondo, 2006)

2.4.1 Redes neurais - multilayer perceptron

A area de conhecimento das redes neurais surgiu de diversas fontes, variando da fascinção da humanidade de compreender e emular o cérebro humano, para questões de copiar habilidades humanas como a fala e uso da linguagem, para prática comercial, científica, disciplinas de engenharia de reconhecimento de padrões, modelagem e predição.

pág 3; ver definição da implementação do weka; pág 84-93;

2.4.2 Métodos avaliativos

Métodos avaliativos 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). Para isto, utiliza-se de medidas estatísticas para auxiliar na tarefa. 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*).

- Verdadeiro positivo (TP): instância que pertence a classe positiva e que foi classificada como positiva.
- Verdadeiro negativo (TN): instância que pertence a classe negativa e que foi classificada como negativa.
- Falso positivo (FP): instância que pertence a classe negativa e que foi classificada como positiva.
- Falso negativo (FN): instância que pertence a classe positiva e que foi classificada como negativa.
- **

**[1] Powers, D.M.W., 2011. Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation. Journal of Machine Learning Technologies, 2(1), 37-63.

**[2] Christopher, D. M.; Prabhakar, R.; Hinrich, S., 2009. An introduction to information retrieval. Cambridge University Press. Cambridge, England.

**[3] Manning, C.. Natural Language Processing Lecture: Precision, Recall, and the F measure. Disponível em: https://class.coursera.org/nlp/lecture/142. Acessado maio de 2015.

- Precision: É mensurada da seguinte forma, TP/(TP+FP), indica ...
- Recall: É mensurada da seguinte forma, TP/(TP+FN), indica ...
- Accuracy: É mensurada da seguinte forma, TP+TN/(TP+TN+FP+FN), indica ...

Um método bastante simples para avaliar um modelo é dividir o dateset em dois grandes conjuntos (um para a construção do modelo - treino, e ou outro para testar o modelo - teste). Então dividi-se o dataset de treino em N partes e faz pelo menos uma medição estatística (por exemplo, acurácia) de cada N sobre o*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, supomos numa 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 2.17. Daí computa-se uma medida estatística. Então o procedimento é executado k vezes nos diferentes datasets de treino (observe que eles têm muito em comum). 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 randomizações diferentes para cada repetição.(Witten e Frank, 2005)

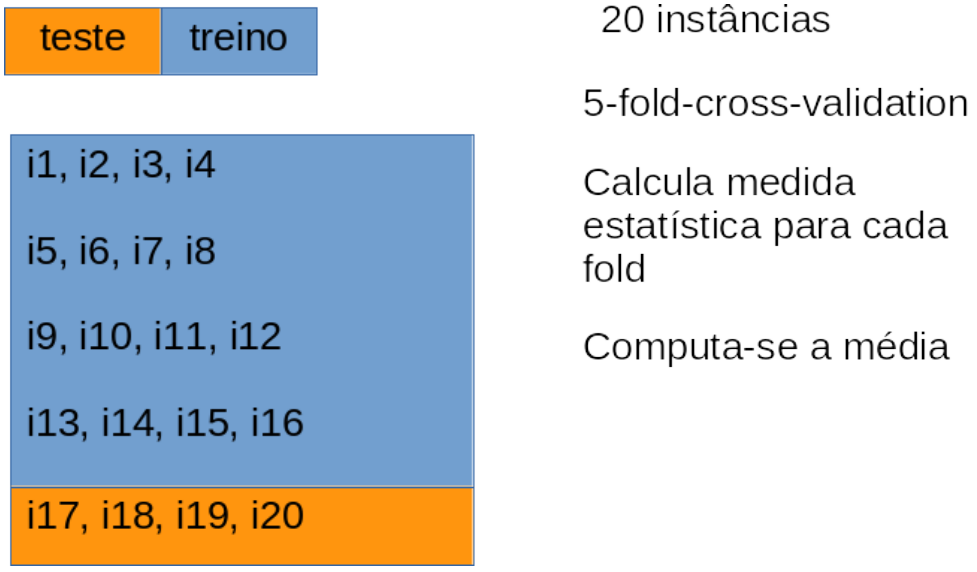

Figura 2.17: k-fold-cross-validation. Adaptado de(Keller,)

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.(Witten e Frank, 2005)

Capítulo

3

//TODO

PROPOSTA

Capítulo

4

//TODO

EXPERIMENTOS

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Pe- ter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1. pag 215 manual$_weka(radomizacao)$

**Coloca o desvio padrão tb na comparação das medidas.

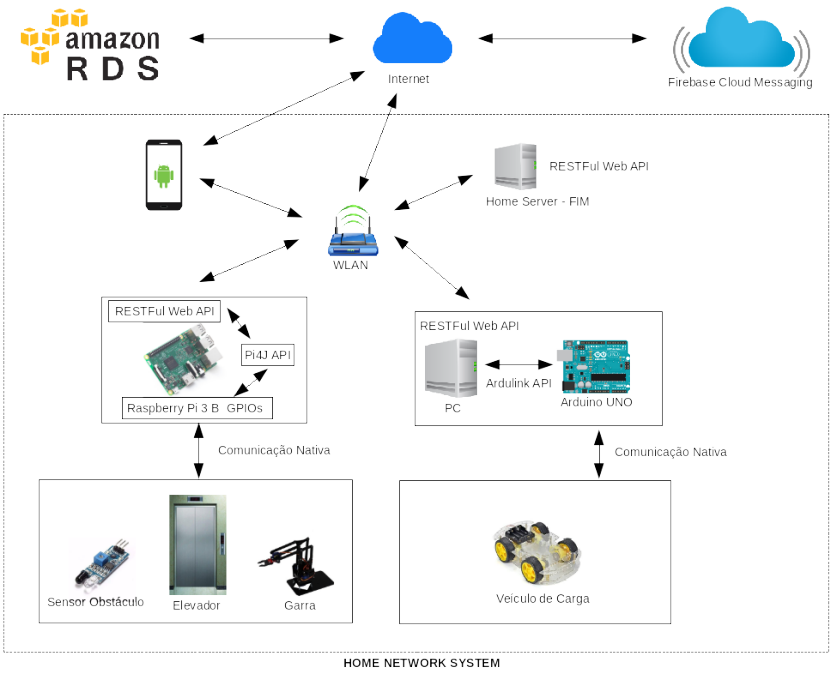

Figura 4.1: Modelo do HNS utilizado neste trabalho.

**Ve se da para usar o Experimenter ou o Evaluation no fonte.

VE SE DA TEMPO PARA RODA UM OUTRO CLASSIFICADOR E VERIFICAR OS RESULTADOS **trees.SimpleCart** trees.DecisionStump functions.SMO

trees.BFTree trees.LMT trees.FT **ver se dar tempo de gerar curva roc(verifdicar o motivo da roc, é viavel para este cenário) e precision e recall dos resultados dos 3 modelos**

MOTIVO: escolher os melhores parametros para o modelo em questão, neste caso, rede neural.

machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for three reasons:

simplification of models to make them easier to interpret by researchers/users,[1] shorter training times, enhanced generalization by reducing overfitting[2](formally, reduction of variance[1])

Procedimentos utilizados - experimento: -gera uma plotMatriz dos dados do dataset -verifica quais dados poderiam ser melhor separados por uma curva(par a par)

-neste caso, visualmente são esse abaixo: $t_num_widthet_num_diameter(quemnãotemwidth, temdiam$

-testa com alguma combinação distinta dos parâmetros acima $t_num_width, t_num_mass, t_num_diamete$

-Verifica a generalização utilizando técnicas estaitisticas, tais como, stratifyed-k-crossvalidation e/ou randomized-holdout

-escolhe o melhor modelo baseado no que você quer do modelo, neste caso o que tiver melhor recall, seguido de accuracy. POIS É MELHOR ERRAR DIZENDO QUE É, DO QUE ERRAR DIZENDO QUE NÃO É.

-gera um classificador com todo o dataset verificado na etapa anterior. -testa esse classificador com um dataset distinto do anterior. -DEMONSTRA OS RESULTADOS E COMPARA COM OS TESTES ANTERIORES.

Capítulo

5

Síntese da investigação e dos experimentos realizados nesta monografia

CONCLUSÃO

Solutions for specific feature interactions can then be generalized to other interactions of the same category.

The approaches described above require a-priori data about features and their potential interactions. Since this is a serious drawback with on-line approaches, approaches which circumvent that requirement have been developed. One way of achieving this is to introduce, during run-time, a separate phase of "collecting" information before the actual operation of the approach. During this phase the behaviour of the feature is observed and the information is stored in a database. Aggoun and Combes [3] propose a "pre-deployment" phase where a passive observer gathers information about the feature behaviour in the network. The gathered information is then used by the active observer (essentially a feature manager) in the operation phase of the service to detect and resolve interactions. Similarly, Tsang and Magill [113] gather behaviour "signatures" of features in an iso- lated on-line environment (with just the feature under observation being active) and store those in a database. The feature manager then accesses this database during live network operation to detect and resolve interactions.

On-line techniques offer strengths which no other technique possesses. They address both detection and resolution. While this is not often the case with other approaches (e.g. formal methods), automated resolution is especially critical with on-line tech- niques. At design time, manually changing specifications is sufficient. However, feature interactions detected at run-time need to be resolved instantaneously to keep the in- tegrity of the network. On the other hand, work reported so far suggests that run-time resolution is difficult. This is due, in part,to the limited amount of information avail- able. Making an informed decision without much knowledge on the features is hard. Not surprisingly, approaches requiring a-priori knowledge offer the best resolutions. However, the biggest potential of on-line techniques lies in their ability to cope with additional services in the network at runtime and thus allowing for open and expand- able systems. A-priori knowledge is a major obstacle to use this potential. Thus, in terms of feature interaction resolution, many approaches opted for the obvious choice: terminating the call. However,

is this a "bad" resolution? A single call is very cheap. Thus terminating a call might at most annoy the affected users. On the other hand a complex algorithm finding a good resolution is possibly very expensive in terms of its run-time behaviour. Further, the resolution might still confuse the users and mismatch their expectations. To progress work on this issue work is ongoing to augment run-time approaches with feature independent information from off-line techniques to guide the resolution. These approaches are referred to as hybrids. There is very little published work on hybrid approaches to date, however, two examples are the work by Calder et al. [23,105] and by Aggoun and Combes [3].

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, J. et al. An overview of formal methods tools andtechniques. 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⟩.

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⟩.

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.

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⟩.

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.

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.

KELLER, F. *Evaluation: Connectionist and Statistical Language Processing*. Disponível em: ⟨http://www.coli.uni-saarland.de/~crocker/Teaching/Connectionist/lecture11_4up.pdf⟩.

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⟩.

KUROSE, j.; ROSS, k. *COMPUTER NETWORKING A Top-Down Approach*. [S.l.]: PEARSON, 2013. ISBN 978-0-13-285620-1.

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.

NANOSYSTEMS, i. C.-o. w. t. W. G. R. o. t. E. E. I. D. N. E. . R. I. G. M. . *Internet of Things in 2020: Roadmap for the Future*. 2008.

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⟩.

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.

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.